Kerberos Administration Guide

Release 1.22.3

MIT

CONTENTS

1 Installation guide 1

2 Configuration Files 11

3 Realm configuration decisions 45

4 Database administration 51

5 Database types 61

6 Account lockout 65

7 Configuring Kerberos with OpenLDAP back-end 69

8 Application servers 71

9 Host configuration 75

10 Backups of secure hosts 79

11 PKINIT configuration 81

12 OTP Preauthentication 87

13 SPAKE Preauthentication 89

14 Addressing dictionary attack risks 91

15 Principal names and DNS 93

16 Encryption types 95

17 HTTPS proxy configuration 99

18 Authentication indicators 101

19 Administration programs 103

20 MIT Kerberos defaults 143

21 Environment variables 147

22 Troubleshooting 149

23 Advanced topics 151

24 Various links 159

Index 161

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Contents

1.1.1 Installing KDCs

When setting up Kerberos in a production environment, it is best to have multiple replica KDCs alongside with a primary KDC to ensure the continued availability of the Kerberized services. Each KDC contains a copy of the Kerberos database. The primary KDC contains the writable copy of the realm database, which it replicates to the replica KDCs at regular intervals. All database changes (such as password changes) are made on the primary KDC. Replica KDCs provide Kerberos ticket-granting services, but not database administration, when the primary KDC is unavailable. MIT recommends that you install all of your KDCs to be able to function as either the primary or one of the replicas. This will enable you to easily switch your primary KDC with one of the replicas if necessary (see *Switching primary and replica KDCs*). This installation procedure is based on that recommendation.

Warning:

- The Kerberos system relies on the availability of correct time information. Ensure that the primary and all replica KDCs have properly synchronized clocks.
- It is best to install and run KDCs on secured and dedicated hardware with limited access. If your KDC is also a file server, FTP server, Web server, or even just a client machine, someone who obtained root access through a security hole in any of those areas could potentially gain access to the Kerberos database.

Install and configure the primary KDC

Install Kerberos either from the OS-provided packages or from the source (See do_build).

Note: For the purpose of this document we will use the following names:

```
kerberos.mit.edu     - primary KDC
kerberos-1.mit.edu   - replica KDC
ATHENA.MIT.EDU       - realm name
.k5.ATHENA.MIT.EDU   - stash file
admin/admin          - admin principal
```

See *MIT Kerberos defaults* for the default names and locations of the relevant to this topic files. Adjust the names and paths to your system environment.

Edit KDC configuration files

Modify the configuration files, *krb5.conf* and *kdc.conf*, to reflect the correct information (such as domain-realm mappings and Kerberos servers names) for your realm. (See *MIT Kerberos defaults* for the recommended default locations for these files).

Most of the tags in the configuration have default values that will work well for most sites. There are some tags in the *krb5.conf* file whose values must be specified, and this section will explain those.

If the locations for these configuration files differs from the default ones, set **KRB5_CONFIG** and **KRB5_KDC_PROFILE** environment variables to point to the krb5.conf and kdc.conf respectively. For example:

```
export KRB5_CONFIG=/yourdir/krb5.conf
export KRB5_KDC_PROFILE=/yourdir/kdc.conf
```

krb5.conf

If you are not using DNS TXT records (see *Mapping hostnames onto Kerberos realms*), you must specify the **default_realm** in the *[libdefaults]* section. If you are not using DNS URI or SRV records (see *Hostnames for KDCs* and *KDC Discovery*), you must include the **kdc** tag for each *realm* in the *[realms]* section. To communicate with the kadmin server in each realm, the **admin_server** tag must be set in the *[realms]* section.

An example krb5.conf file:

```
[libdefaults]
    default_realm = ATHENA.MIT.EDU

[realms]
    ATHENA.MIT.EDU = {
        kdc = kerberos.mit.edu
        kdc = kerberos-1.mit.edu
        admin_server = kerberos.mit.edu
    }
```

kdc.conf

The kdc.conf file can be used to control the listening ports of the KDC and kadmind, as well as realm-specific defaults, the database type and location, and logging.

An example kdc.conf file:

```
[kdcdefaults]
    kdc_listen = 88
    kdc_tcp_listen = 88

[realms]
    ATHENA.MIT.EDU = {
        kadmind_port = 749
        max_life = 12h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        master_key_type = aes256-cts
        supported_enctypes = aes256-cts:normal aes128-cts:normal
        # If the default location does not suit your setup,
```

(continues on next page)

(continued from previous page)

```
        # explicitly configure the following values:
        #    database_name = /var/krb5kdc/principal
        #    key_stash_file = /var/krb5kdc/.k5.ATHENA.MIT.EDU
        #    acl_file = /var/krb5kdc/kadm5.acl
    }

[logging]
    # By default, the KDC and kadmind will log output using
    # syslog.  You can instead send log output to files like this:
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmin.log
    default = FILE:/var/log/krb5lib.log
```

Replace `ATHENA.MIT.EDU` and `kerberos.mit.edu` with the name of your Kerberos realm and server respectively.

Note: You have to have write permission on the target directories (these directories must exist) used by **database_name**, **key_stash_file**, and **acl_file**.

Create the KDC database

You will use the *kdb5_util* command on the primary KDC to create the Kerberos database and the optional stash_definition.

Note: If you choose not to install a stash file, the KDC will prompt you for the master key each time it starts up. This means that the KDC will not be able to start automatically, such as after a system reboot.

---

kdb5_util will prompt you for the master password for the Kerberos database. This password can be any string. A good password is one you can remember, but that no one else can guess. Examples of bad passwords are words that can be found in a dictionary, any common or popular name, especially a famous person (or cartoon character), your username in any form (e.g., forward, backward, repeated twice, etc.), and any of the sample passwords that appear in this manual. One example of a password which might be good if it did not appear in this manual is "MITiys4K5!", which represents the sentence "MIT is your source for Kerberos 5!" (It's the first letter of each word, substituting the numeral "4" for the word "for", and includes the punctuation mark at the end.)

The following is an example of how to create a Kerberos database and stash file on the primary KDC, using the *kdb5_util* command. Replace `ATHENA.MIT.EDU` with the name of your Kerberos realm:

```
shell% kdb5_util create -r ATHENA.MIT.EDU -s

Initializing database '/usr/local/var/krb5kdc/principal' for realm 'ATHENA.MIT.EDU',
master key name 'K/M@ATHENA.MIT.EDU'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:  <= Type the master password.
Re-enter KDC database master key to verify:  <= Type it again.
shell%
```

This will create five files in *LOCALSTATEDIR*`/krb5kdc` (or at the locations specified in *kdc.conf*):

- two Kerberos database files, `principal`, and `principal.ok`

- the Kerberos administrative database file, `principal.kadm5`
- the administrative database lock file, `principal.kadm5.lock`
- the stash file, in this example `.k5.ATHENA.MIT.EDU`. If you do not want a stash file, run the above command without the **-s** option.

For more information on administrating Kerberos database see *Operations on the Kerberos database*.

Add administrators to the ACL file

Next, you need create an Access Control List (ACL) file and put the Kerberos principal of at least one of the administrators into it. This file is used by the *kadmind* daemon to control which principals may view and make privileged modifications to the Kerberos database files. The ACL filename is determined by the **acl_file** variable in *kdc.conf*; the default is *LOCALSTATEDIR*`/krb5kdc/kadm5.acl`.

For more information on Kerberos ACL file see *kadm5.acl*.

Add administrators to the Kerberos database

Next you need to add administrative principals (i.e., principals who are allowed to administer Kerberos database) to the Kerberos database. You *must* add at least one principal now to allow communication between the Kerberos administration daemon kadmind and the kadmin program over the network for further administration. To do this, use the kadmin.local utility on the primary KDC. kadmin.local is designed to be run on the primary KDC host without using Kerberos authentication to an admin server; instead, it must have read and write access to the Kerberos database on the local filesystem.

The administrative principals you create should be the ones you added to the ACL file (see *Add administrators to the ACL file*).

In the following example, the administrative principal `admin/admin` is created:

```
shell% kadmin.local

kadmin.local: addprinc admin/admin@ATHENA.MIT.EDU

No policy specified for "admin/admin@ATHENA.MIT.EDU";
assigning "default".
Enter password for principal admin/admin@ATHENA.MIT.EDU:  <= Enter a password.
Re-enter password for principal admin/admin@ATHENA.MIT.EDU:  <= Type it again.
Principal "admin/admin@ATHENA.MIT.EDU" created.
kadmin.local:
```

Start the Kerberos daemons on the primary KDC

At this point, you are ready to start the Kerberos KDC (*krb5kdc*) and administrative daemons on the primary KDC. To do so, type:

```
shell% krb5kdc
shell% kadmind
```

Each server daemon will fork and run in the background.

Note: Assuming you want these daemons to start up automatically at boot time, you can add them to the KDC's `/etc/rc` or `/etc/inittab` file. You need to have a stash_definition in order to do this.

You can verify that they started properly by checking for their startup messages in the logging locations you defined in *krb5.conf* (see *[logging]*). For example:

```
shell% tail /var/log/krb5kdc.log
Dec 02 12:35:47 beeblebrox krb5kdc[3187](info): commencing operation
shell% tail /var/log/kadmin.log
Dec 02 12:35:52 beeblebrox kadmind[3189](info): starting
```

Any errors the daemons encounter while starting will also be listed in the logging output.

As an additional verification, check if kinit(1) succeeds against the principals that you have created on the previous step (*Add administrators to the Kerberos database*). Run:

```
shell% kinit admin/admin@ATHENA.MIT.EDU
```

Install the replica KDCs

You are now ready to start configuring the replica KDCs.

Note: Assuming you are setting the KDCs up so that you can easily switch the primary KDC with one of the replicas, you should perform each of these steps on the primary KDC as well as the replica KDCs, unless these instructions specify otherwise.

Create host keytabs for replica KDCs

Each KDC needs a `host` key in the Kerberos database. These keys are used for mutual authentication when propagating the database dump file from the primary KDC to the secondary KDC servers.

On the primary KDC, connect to administrative interface and create the host principal for each of the KDCs' `host` services. For example, if the primary KDC were called `kerberos.mit.edu`, and you had a replica KDC named `kerberos-1.mit.edu`, you would type the following:

```
shell% kadmin
kadmin: addprinc -randkey host/kerberos.mit.edu
No policy specified for "host/kerberos.mit.edu@ATHENA.MIT.EDU"; assigning "default"
Principal "host/kerberos.mit.edu@ATHENA.MIT.EDU" created.

kadmin: addprinc -randkey host/kerberos-1.mit.edu
No policy specified for "host/kerberos-1.mit.edu@ATHENA.MIT.EDU"; assigning "default"
Principal "host/kerberos-1.mit.edu@ATHENA.MIT.EDU" created.
```

It is not strictly necessary to have the primary KDC server in the Kerberos database, but it can be handy if you want to be able to swap the primary KDC with one of the replicas.

Next, extract `host` random keys for all participating KDCs and store them in each host's default keytab file. Ideally, you should extract each keytab locally on its own KDC. If this is not feasible, you should use an encrypted session to send them across the network. To extract a keytab directly on a replica KDC called `kerberos-1.mit.edu`, you would execute the following command:

```
kadmin: ktadd host/kerberos-1.mit.edu
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type aes256-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type aes128-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type aes256-cts-hmac-sha384-192 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type arcfour-hmac added to keytab FILE:/etc/krb5.keytab.
```

If you are instead extracting a keytab for the replica KDC called `kerberos-1.mit.edu` on the primary KDC, you should use a dedicated temporary keytab file for that machine's keytab:

```
kadmin: ktadd -k /tmp/kerberos-1.keytab host/kerberos-1.mit.edu
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type aes256-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
Entry for principal host/kerberos-1.mit.edu with kvno 2, encryption
    type aes128-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab.
```

The file `/tmp/kerberos-1.keytab` can then be installed as `/etc/krb5.keytab` on the host `kerberos-1.mit.edu`.

Configure replica KDCs

Database propagation copies the contents of the primary's database, but does not propagate configuration files, stash files, or the kadm5 ACL file. The following files must be copied by hand to each replica (see *MIT Kerberos defaults* for the default locations for these files):

- krb5.conf
- kdc.conf
- kadm5.acl
- master key stash file

Move the copied files into their appropriate directories, exactly as on the primary KDC. kadm5.acl is only needed to allow a replica to swap with the primary KDC.

The database is propagated from the primary KDC to the replica KDCs via the *kpropd* daemon. You must explicitly specify the principals which are allowed to provide Kerberos dump updates on the replica machine with a new database. Create a file named kpropd.acl in the KDC state directory containing the `host` principals for each of the KDCs:

```
host/kerberos.mit.edu@ATHENA.MIT.EDU
host/kerberos-1.mit.edu@ATHENA.MIT.EDU
```

Note: If you expect that the primary and replica KDCs will be switched at some point of time, list the host principals from all participating KDC servers in kpropd.acl files on all of the KDCs. Otherwise, you only need to list the primary KDC's host principal in the kpropd.acl files of the replica KDCs.

Then, add the following line to `/etc/inetd.conf` on each KDC (adjust the path to kpropd):

```
krb5_prop stream tcp nowait root /usr/local/sbin/kpropd kpropd
```

You also need to add the following line to `/etc/services` on each KDC, if it is not already present (assuming that the default port is used):

```
krb5_prop       754/tcp              # Kerberos replica propagation
```

Restart inetd daemon.

Alternatively, start *kpropd* as a stand-alone daemon. This is required when incremental propagation is enabled.

Now that the replica KDC is able to accept database propagation, you'll need to propagate the database from the primary server.

NOTE: Do not start the replica KDC yet; you still do not have a copy of the primary's database.

Propagate the database to each replica KDC

First, create a dump file of the database on the primary KDC, as follows:

```
shell% kdb5_util dump /usr/local/var/krb5kdc/replica_datatrans
```

Then, manually propagate the database to each replica KDC, as in the following example:

```
shell% kprop -f /usr/local/var/krb5kdc/replica_datatrans kerberos-1.mit.edu

Database propagation to kerberos-1.mit.edu: SUCCEEDED
```

You will need a script to dump and propagate the database. The following is an example of a Bourne shell script that will do this.

Note: Remember that you need to replace `/usr/local/var/krb5kdc` with the name of the KDC state directory.

```
#!/bin/sh

kdclist = "kerberos-1.mit.edu kerberos-2.mit.edu"

kdb5_util dump /usr/local/var/krb5kdc/replica_datatrans

for kdc in $kdclist
do
    kprop -f /usr/local/var/krb5kdc/replica_datatrans $kdc
done
```

You will need to set up a cron job to run this script at the intervals you decided on earlier (see *Database propagation*).

Now that the replica KDC has a copy of the Kerberos database, you can start the krb5kdc daemon:

```
shell% krb5kdc
```

As with the primary KDC, you will probably want to add this command to the KDCs' `/etc/rc` or `/etc/inittab` files, so they will start the krb5kdc daemon automatically at boot time.

Propagation failed?

You may encounter the following error messages. For a more detailed discussion on possible causes and solutions click on the error link to be redirected to *Troubleshooting* section.

1. *kprop: No route to host while connecting to server*
2. *kprop: Connection refused while connecting to server*
3. *kprop: Server rejected authentication (during sendauth exchange) while authenticating to server*

Add Kerberos principals to the database

Once your KDCs are set up and running, you are ready to use *kadmin* to load principals for your users, hosts, and other services into the Kerberos database. This procedure is described fully in *Principals*.

You may occasionally want to use one of your replica KDCs as the primary. This might happen if you are upgrading the primary KDC, or if your primary KDC has a disk crash. See the following section for the instructions.

Switching primary and replica KDCs

You may occasionally want to use one of your replica KDCs as the primary. This might happen if you are upgrading the primary KDC, or if your primary KDC has a disk crash.

Assuming you have configured all of your KDCs to be able to function as either the primary KDC or a replica KDC (as this document recommends), all you need to do to make the changeover is:

If the primary KDC is still running, do the following on the *old* primary KDC:

1. Kill the kadmind process.
2. Disable the cron job that propagates the database.
3. Run your database propagation script manually, to ensure that the replicas all have the latest copy of the database (see *Propagate the database to each replica KDC*).

On the *new* primary KDC:

1. Start the *kadmind* daemon (see *Start the Kerberos daemons on the primary KDC*).
2. Set up the cron job to propagate the database (see *Propagate the database to each replica KDC*).
3. Switch the CNAMEs of the old and new primary KDCs. If you can't do this, you'll need to change the *krb5.conf* file on every client machine in your Kerberos realm.

Incremental database propagation

If you expect your Kerberos database to become large, you may wish to set up incremental propagation to replica KDCs. See *Incremental database propagation* for details.

1.1.2 Installing and configuring UNIX client machines

The Kerberized client programs include kinit(1), klist(1), kdestroy(1), and kpasswd(1). All of these programs are in the directory *BINDIR*.

You can often integrate Kerberos with the login system on client machines, typically through the use of PAM. The details vary by operating system, and should be covered in your operating system's documentation. If you do this, you will need to make sure your users know to use their Kerberos passwords when they log in.

You will also need to educate your users to use the ticket management programs kinit, klist, and kdestroy. If you do not have Kerberos password changing integrated into the native password program (again, typically through PAM), you will need to educate users to use kpasswd in place of its non-Kerberos counterparts passwd.

Client machine configuration files

Each machine running Kerberos should have a *krb5.conf* file. At a minimum, it should define a **default_realm** setting in *[libdefaults]*. If you are not using DNS SRV records (*Hostnames for KDCs*) or URI records (*KDC Discovery*), it must also contain a *[realms]* section containing information for your realm's KDCs.

Consider setting **rdns** to false in order to reduce your dependence on precisely correct DNS information for service hostnames. Turning this flag off means that service hostnames will be canonicalized through forward name resolution (which adds your domain name to unqualified hostnames, and resolves CNAME records in DNS), but not through reverse address lookup. The default value of this flag is true for historical reasons only.

If you anticipate users frequently logging into remote hosts (e.g., using ssh) using forwardable credentials, consider setting **forwardable** to true so that users obtain forwardable tickets by default. Otherwise users will need to use `kinit -f` to get forwardable tickets.

Consider adjusting the **ticket_lifetime** setting to match the likely length of sessions for your users. For instance, if most of your users will be logging in for an eight-hour workday, you could set the default to ten hours so that tickets obtained in the morning expire shortly after the end of the workday. Users can still manually request longer tickets when necessary, up to the maximum allowed by each user's principal record on the KDC.

If a client host may access services in different realms, it may be useful to define a *[domain_realm]* mapping so that clients know which hosts belong to which realms. However, if your clients and KDC are running release 1.7 or later, it is also reasonable to leave this section out on client machines and just define it in the KDC's krb5.conf.

1.1.3 UNIX Application Servers

An application server is a host that provides one or more services over the network. Application servers can be "secure" or "insecure." A "secure" host is set up to require authentication from every client connecting to it. An "insecure" host will still provide Kerberos authentication, but will also allow unauthenticated clients to connect.

If you have Kerberos V5 installed on all of your client machines, MIT recommends that you make your hosts secure, to take advantage of the security that Kerberos authentication affords. However, if you have some clients that do not have Kerberos V5 installed, you can run an insecure server, and still take advantage of Kerberos V5's single sign-on capability.

The keytab file

All Kerberos server machines need a keytab file to authenticate to the KDC. By default on UNIX-like systems this file is named *DEFKTNAME*. The keytab file is an local copy of the host's key. The keytab file is a potential point of entry for a break-in, and if compromised, would allow unrestricted access to its host. The keytab file should be readable only by root, and should exist only on the machine's local disk. The file should not be part of any backup of the machine, unless access to the backup data is secured as tightly as access to the machine's root password.

In order to generate a keytab for a host, the host must have a principal in the Kerberos database. The procedure for adding hosts to the database is described fully in *Principals*. (See *Create host keytabs for replica KDCs* for a brief description.) The keytab is generated by running *kadmin* and issuing the *ktadd* command.

For example, to generate a keytab file to allow the host `trillium.mit.edu` to authenticate for the services host, ftp, and pop, the administrator `joeadmin` would issue the command (on `trillium.mit.edu`):

```
trillium% kadmin
Authenticating as principal root/admin@ATHENA.MIT.EDU with password.
Password for root/admin@ATHENA.MIT.EDU:
kadmin: ktadd host/trillium.mit.edu ftp/trillium.mit.edu pop/trillium.mit.edu
Entry for principal host/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption type␣
↪aes256-cts-hmac-sha384-192 added to keytab FILE:/etc/krb5.keytab.
kadmin: Entry for principal ftp/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption␣
↪type aes256-cts-hmac-sha384-192 added to keytab FILE:/etc/krb5.keytab.
kadmin: Entry for principal pop/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption␣
↪type aes256-cts-hmac-sha384-192 added to keytab FILE:/etc/krb5.keytab.
kadmin: quit
trillium%
```

If you generate the keytab file on another host, you need to get a copy of the keytab file onto the destination host (`trillium`, in the above example) without sending it unencrypted over the network.

Some advice about secure hosts

Kerberos V5 can protect your host from certain types of break-ins, but it is possible to install Kerberos V5 and still leave your host vulnerable to attack. Obviously an installation guide is not the place to try to include an exhaustive list of countermeasures for every possible attack, but it is worth noting some of the larger holes and how to close them.

We recommend that backups of secure machines exclude the keytab file (*DEFKTNAME*). If this is not possible, the backups should at least be done locally, rather than over a network, and the backup tapes should be physically secured.

The keytab file and any programs run by root, including the Kerberos V5 binaries, should be kept on local disk. The keytab file should be readable only by root.

1.2 Additional references

1. Debian: Setting up MIT Kerberos 5
2. Solaris: Configuring the Kerberos Service

CHAPTER

TWO

CONFIGURATION FILES

Kerberos uses configuration files to allow administrators to specify settings on a per-machine basis. *krb5.conf* applies to all applications using the Kerboros library, on clients and servers. For KDC-specific applications, additional settings can be specified in *kdc.conf*; the two files are merged into a configuration profile used by applications accessing the KDC database directly. *kadm5.acl* is also only used on the KDC, it controls permissions for modifying the KDC database.

2.1 Contents

2.1.1 krb5.conf

The krb5.conf file contains Kerberos configuration information, including the locations of KDCs and admin servers for the Kerberos realms of interest, defaults for the current realm and for Kerberos applications, and mappings of hostnames onto Kerberos realms. Normally, you should install your krb5.conf file in the directory `/etc`. You can override the default location by setting the environment variable **KRB5_CONFIG**. Multiple colon-separated filenames may be specified in **KRB5_CONFIG**; all files which are present will be read. Starting in release 1.14, directory names can also be specified in **KRB5_CONFIG**; all files within the directory whose names consist solely of alphanumeric characters, dashes, or underscores will be read.

Structure

The krb5.conf file is set up in the style of a Windows INI file. Lines beginning with '#' or ';' (possibly after initial whitespace) are ignored as comments. Sections are headed by the section name, in square brackets. Each section may contain zero or more relations, of the form:

```
foo = bar
```

or:

```
fubar = {
    foo = bar
    baz = quux
}
```

The krb5.conf file can include other files using either of the following directives at the beginning of a line:

```
include FILENAME
includedir DIRNAME
```

FILENAME or *DIRNAME* should be an absolute path. The named file or directory must exist and be readable. Including a directory includes all files within the directory whose names consist solely of alphanumeric characters, dashes, or underscores. Starting in release 1.15, files with names ending in ".conf" are also included, unless the name begins with ".". Included profile files are syntactically independent of their parents, so each included file must begin with a section header. Starting in release 1.17, files are read in alphanumeric order; in previous releases, they may be read in any order.

Placing a '*' after the closing bracket of a section name indicates that the section is *final*, meaning that if the same section appears again later, it will be ignored. A subsection can be marked as final by placing a '*' after either the tag name or the closing brace. A relation can be marked as final by placing a '*' after the tag name. Prior to release 1.22, only sections and subsections can be marked as final, and the flag only causes values to be ignored if they appear in later files specified in **KRB5_CONFIG**, not if they appear later within the same file or an included file.

The krb5.conf file can specify that configuration should be obtained from a loadable module, rather than the file itself, using the following directive at the beginning of a line before any section headers:

```
module MODULEPATH:RESIDUAL
```

MODULEPATH may be relative to the library path of the krb5 installation, or it may be an absolute path. *RESIDUAL* is provided to the module at initialization time. If krb5.conf uses a module directive, *kdc.conf* should also use one if it exists.

Sections

The krb5.conf file may contain the following sections:

[libdefaults]	Settings used by the Kerberos V5 library
[realms]	Realm-specific contact information and settings
[domain_realm]	Maps server hostnames to Kerberos realms
[capaths]	Authentication paths for non-hierarchical cross-realm
[appdefaults]	Settings used by some Kerberos V5 applications
[plugins]	Controls plugin module registration

Additionally, krb5.conf may include any of the relations described in *kdc.conf*, but it is not a recommended practice.

[libdefaults]

The libdefaults section may contain any of the following relations:

allow_des3
: Permit the KDC to issue tickets with des3-cbc-sha1 session keys. In future releases, this flag will allow des3-cbc-sha1 to be used at all. The default value for this tag is false. (Added in release 1.21.)

allow_rc4
: Permit the KDC to issue tickets with arcfour-hmac session keys. In future releases, this flag will allow arcfour-hmac to be used at all. The default value for this tag is false. (Added in release 1.21.)

allow_weak_crypto
: If this flag is set to false, then weak encryption types (as noted in *Encryption types* in *kdc.conf*) will be filtered out of the lists **default_tgs_enctypes**, **default_tkt_enctypes**, and **permitted_enctypes**. The default value for this tag is false.

canonicalize
: If this flag is set to true, initial ticket requests to the KDC will request canonicalization of the client principal

name, and answers with different client principals than the requested principal will be accepted. The default value is false.

ccache_type
This parameter determines the format of credential cache types created by kinit(1) or other programs. The default value is 4, which represents the most current format. Smaller values can be used for compatibility with very old implementations of Kerberos which interact with credential caches on the same host.

clockskew
Sets the maximum allowable amount of clockskew in seconds that the library will tolerate before assuming that a Kerberos message is invalid. The default value is 300 seconds, or five minutes.

The clockskew setting is also used when evaluating ticket start and expiration times. For example, tickets that have reached their expiration time can still be used (and renewed if they are renewable tickets) if they have been expired for a shorter duration than the **clockskew** setting.

default_ccache_name
This relation specifies the name of the default credential cache. The default is *DEFCCNAME*. This relation is subject to parameter expansion (see below). New in release 1.11.

default_client_keytab_name
This relation specifies the name of the default keytab for obtaining client credentials. The default is *DEFCKTNAME*. This relation is subject to parameter expansion (see below). New in release 1.11.

default_keytab_name
This relation specifies the default keytab name to be used by application servers such as sshd. The default is *DEFKTNAME*. This relation is subject to parameter expansion (see below).

default_rcache_name
This relation specifies the name of the default replay cache. The default is `dfl:`. This relation is subject to parameter expansion (see below). New in release 1.18.

default_realm
Identifies the default Kerberos realm for the client. Set its value to your Kerberos realm. If this value is not set, then a realm must be specified with every Kerberos principal when invoking programs such as kinit(1).

default_tgs_enctypes
Identifies the supported list of session key encryption types that the client should request when making a TGS-REQ, in order of preference from highest to lowest. The list may be delimited with commas or whitespace. See *Encryption types* in *kdc.conf* for a list of the accepted values for this tag. Starting in release 1.18, the default value is the value of **permitted_enctypes**. For previous releases or if **permitted_enctypes** is not set, the default value is `aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 aes128-cts-hmac-sha256-128 des3-cbc-sha1 arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac`.

Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

default_tkt_enctypes
Identifies the supported list of session key encryption types that the client should request when making an AS-REQ, in order of preference from highest to lowest. The format is the same as for default_tgs_enctypes. Starting in release 1.18, the default value is the value of **permitted_enctypes**. For previous releases or if **permitted_enctypes** is not set, the default value is `aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 aes128-cts-hmac-sha256-128 des3-cbc-sha1 arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac`.

Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

dns_canonicalize_hostname
Indicate whether name lookups will be used to canonicalize hostnames for use in service principal names. Setting this flag to false can improve security by reducing reliance on DNS, but means that short hostnames will not be canonicalized to fully-qualified hostnames. If this option is set to `fallback` (new in release 1.18), DNS canonicalization will only be performed the server hostname is not found with the original name when requesting credentials. The default value is true.

dns_lookup_kdc
Indicate whether DNS SRV records should be used to locate the KDCs and other servers for a realm, if they are not listed in the krb5.conf information for the realm. (Note that the admin_server entry must be in the krb5.conf realm information in order to contact kadmind, because the DNS implementation for kadmin is incomplete.)

Enabling this option does open up a type of denial-of-service attack, if someone spoofs the DNS records and redirects you to another server. However, it's no worse than a denial of service, because that fake KDC will be unable to decode anything you send it (besides the initial ticket request, which has no encrypted data), and anything the fake KDC sends will not be trusted without verification using some secret that it won't know.

dns_lookup_realm
Indicate whether DNS TXT records should be used to map hostnames to realm names for hostnames not listed in the [domain_realm] section, and to determine the default realm if **default_realm** is not set. The default value is false.

dns_uri_lookup
Indicate whether DNS URI records should be used to locate the KDCs and other servers for a realm, if they are not listed in the krb5.conf information for the realm. SRV records are used as a fallback if no URI records were found. The default value is true. New in release 1.15.

enforce_ok_as_delegate
If this flag to true, GSSAPI credential delegation will be disabled when the `ok-as-delegate` flag is not set in the service ticket. If this flag is false, the `ok-as-delegate` ticket flag is only enforced when an application specifically requests enforcement. The default value is false.

err_fmt
This relation allows for custom error message formatting. If a value is set, error messages will be formatted by substituting a normal error message for %M and an error code for %C in the value.

extra_addresses
This allows a computer to use multiple local addresses, in order to allow Kerberos to work in a network that uses NATs while still using address-restricted tickets. The addresses should be in a comma-separated list. This option has no effect if **noaddresses** is true.

forwardable
If this flag is true, initial tickets will be forwardable by default, if allowed by the KDC. The default value is false.

ignore_acceptor_hostname
When accepting GSSAPI or krb5 security contexts for host-based service principals, ignore any hostname passed by the calling application, and allow clients to authenticate to any service principal in the keytab matching the service name and realm name (if given). This option can improve the administrative flexibility of server applications on multihomed hosts, but could compromise the security of virtual hosting environments. The default value is false. New in release 1.10.

k5login_authoritative
If this flag is true, principals must be listed in a local user's k5login file to be granted login access, if a .k5login(5) file exists. If this flag is false, a principal may still be granted login access through other mechanisms even if a k5login file exists but does not list the principal. The default value is true.

k5login_directory
If set, the library will look for a local user's k5login file within the named directory, with a filename corresponding

to the local username. If not set, the library will look for k5login files in the user's home directory, with the filename .k5login. For security reasons, .k5login files must be owned by the local user or by root.

kcm_mach_service
On macOS only, determines the name of the bootstrap service used to contact the KCM daemon for the KCM credential cache type. If the value is -, Mach RPC will not be used to contact the KCM daemon. The default value is `org.h5l.kcm`.

kcm_socket
Determines the path to the Unix domain socket used to access the KCM daemon for the KCM credential cache type. If the value is -, Unix domain sockets will not be used to contact the KCM daemon. The default value is `/var/run/.heim_org.h5l.kcm-socket`.

kdc_default_options
Default KDC options (Xored for multiple values) when requesting initial tickets. By default it is set to 0x00000010 (KDC_OPT_RENEWABLE_OK).

kdc_timesync
Accepted values for this relation are 1 or 0. If it is nonzero, client machines will compute the difference between their time and the time returned by the KDC in the timestamps in the tickets and use this value to correct for an inaccurate system clock when requesting service tickets or authenticating to services. This corrective factor is only used by the Kerberos library; it is not used to change the system clock. The default value is 1.

noaddresses
If this flag is true, requests for initial tickets will not be made with address restrictions set, allowing the tickets to be used across NATs. The default value is true.

permitted_enctypes
Identifies the encryption types that servers will permit for session keys and for ticket and authenticator encryption, ordered by preference from highest to lowest. Starting in release 1.18, this tag also acts as the default value for **default_tgs_enctypes** and **default_tkt_enctypes**. The default value for this tag is `aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 aes128-cts-hmac-sha256-128 des3-cbc-sha1 arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac`.

plugin_base_dir
If set, determines the base directory where krb5 plugins are located. The default value is the `krb5/plugins` subdirectory of the krb5 library directory. This relation is subject to parameter expansion (see below) in release 1.17 and later.

preferred_preauth_types
This allows you to set the preferred preauthentication types which the client will attempt before others which may be advertised by a KDC. The default value for this setting is "17, 16, 15, 14", which forces libkrb5 to attempt to use PKINIT if it is supported.

proxiable
If this flag is true, initial tickets will be proxiable by default, if allowed by the KDC. The default value is false.

qualify_shortname
If this string is set, it determines the domain suffix for single-component hostnames when DNS canonicalization is not used (either because **dns_canonicalize_hostname** is false or because forward canonicalization failed). The default value is the first search domain of the system's DNS configuration. To disable qualification of shortnames, set this relation to the empty string with `qualify_shortname = ""`. (New in release 1.18.)

rdns
If this flag is true, reverse name lookup will be used in addition to forward name lookup to canonicalizing hostnames for use in service principal names. If **dns_canonicalize_hostname** is set to false, this flag has no effect. The default value is true.

realm_try_domains
Indicate whether a host's domain components should be used to determine the Kerberos realm of the host. The value of this variable is an integer: -1 means not to search, 0 means to try the host's domain itself, 1 means to also try the domain's immediate parent, and so forth. The library's usual mechanism for locating Kerberos realms is used to determine whether a domain is a valid realm, which may involve consulting DNS if **dns_lookup_kdc** is set. The default is not to search domain components.

renew_lifetime
(duration string.) Sets the default renewable lifetime for initial ticket requests. The default value is 0.

request_timeout
(duration string.) Sets the maximum total time for KDC and password change requests. This timeout does not affect the intervals between requests, so setting a low timeout may result in fewer requests being attempted and/or some servers not being contacted. A value of 0 indicates no specific maximum, in which case requests will time out if no server responds after several tries. The default value is 0. (New in release 1.22.)

spake_preauth_groups
A whitespace or comma-separated list of words which specifies the groups allowed for SPAKE preauthentication. The possible values are:

edwards25519	Edwards25519 curve (**RFC 7748**)
P-256	NIST P-256 curve (**RFC 5480**)
P-384	NIST P-384 curve (**RFC 5480**)
P-521	NIST P-521 curve (**RFC 5480**)

The default value for the client is `edwards25519`. The default value for the KDC is empty. New in release 1.17.

ticket_lifetime
(duration string.) Sets the default lifetime for initial ticket requests. The default value is 1 day.

udp_preference_limit
When sending a message to the KDC, the library will try using TCP before UDP if the size of the message is above **udp_preference_limit**. If the message is smaller than **udp_preference_limit**, then UDP will be tried before TCP. Regardless of the size, both protocols will be tried if the first attempt fails.

verify_ap_req_nofail
If this flag is true, then an attempt to verify initial credentials will fail if the client machine does not have a keytab. The default value is false.

client_aware_channel_bindings
If this flag is true, then all application protocol authentication requests will be flagged to indicate that the application supports channel bindings when operating over a secure channel. The default value is false.

[realms]

Each tag in the [realms] section of the file is the name of a Kerberos realm. The value of the tag is a subsection with relations that define the properties of that particular realm. For each realm, the following tags may be specified in the realm's subsection:

admin_server
Identifies the host where the administration server is running. Typically, this is the primary Kerberos server. This tag must be given a value in order to communicate with the *kadmind* server for the realm.

auth_to_local
This tag allows you to set a general rule for mapping principal names to local user names. It will be used if there is not an explicit mapping for the principal name that is being translated. The possible values are:

RULE:*exp*
The local name will be formulated from *exp*.

The format for *exp* is **[***n***:***string***](***regexp***)s/***pattern***/***replacement***/g**. The integer *n* indicates how many components the target principal should have. If this matches, then a string will be formed from *string*, substituting the realm of the principal for `$0` and the *n*'th component of the principal for `$n` (e.g., if the principal was `johndoe/admin` then `[2:$2$1foo]` would result in the string `adminjohndoefoo`). If this string matches *regexp*, then the `s//[g]` substitution command will be run over the string. The optional **g** will cause the substitution to be global over the *string*, instead of replacing only the first match in the *string*.

DEFAULT
The principal name will be used as the local user name. If the principal has more than one component or is not in the default realm, this rule is not applicable and the conversion will fail.

For example:

```
[realms]
    ATHENA.MIT.EDU = {
        auth_to_local = RULE:[2:$1](johndoe)s/^.*$/guest/
        auth_to_local = RULE:[2:$1;$2](^.*;admin$)s/;admin$//
        auth_to_local = RULE:[2:$2](^.*;root)s/^.*$/root/
        auth_to_local = DEFAULT
    }
```

would result in any principal without `root` or `admin` as the second component to be translated with the default rule. A principal with a second component of `admin` will become its first component. `root` will be used as the local name for any principal with a second component of `root`. The exception to these two rules are any principals `johndoe/*`, which will always get the local name `guest`.

auth_to_local_names
This subsection allows you to set explicit mappings from principal names to local user names. The tag is the mapping name, and the value is the corresponding local user name.

default_domain
This tag specifies the domain used to expand hostnames when translating Kerberos 4 service principals to Kerberos 5 principals (for example, when converting `rcmd.hostname` to `host/hostname.domain`).

disable_encrypted_timestamp
If this flag is true, the client will not perform encrypted timestamp preauthentication if requested by the KDC. Setting this flag can help to prevent dictionary attacks by active attackers, if the realm's KDCs support SPAKE preauthentication or if initial authentication always uses another mechanism or always uses FAST. This flag persists across client referrals during initial authentication. This flag does not prevent the KDC from offering encrypted timestamp. New in release 1.17.

http_anchors
When KDCs and kpasswd servers are accessed through HTTPS proxies, this tag can be used to specify the location of the CA certificate which should be trusted to issue the certificate for a proxy server. If left unspecified, the system-wide default set of CA certificates is used.

The syntax for values is similar to that of values for the **pkinit_anchors** tag:

FILE: *filename*

filename is assumed to be the name of an OpenSSL-style ca-bundle file.

DIR: *dirname*

dirname is assumed to be an directory which contains CA certificates. All files in the directory will be examined; if they contain certificates (in PEM format), they will be used.

ENV: *envvar*

envvar specifies the name of an environment variable which has been set to a value conforming to one of the previous values. For example, `ENV:X509_PROXY_CA`, where environment variable `X509_PROXY_CA` has been set to `FILE:/tmp/my_proxy.pem`.

kdc
The name or address of a host running a KDC for the realm, or a UNIX domain socket path of a locally running KDC. An optional port number, separated from the hostname by a colon, may be included. If the name or address contains colons (for example, if it is an IPv6 address), enclose it in square brackets to distinguish the colon from a port separator. For your computer to be able to communicate with the KDC for each realm, this tag must be given a value in each realm subsection in the configuration file, or there must be DNS SRV records specifying the KDCs.

kpasswd_server
The location of the password change server for the realm, using the same syntax as **kdc**. If there is no such entry, DNS will be queried (unless forbidden by **dns_lookup_kdc**). Finally, port 464 on the **admin_server** host will be tried.

master_kdc
The name for **primary_kdc** prior to release 1.19. Its value is used as a fallback if **primary_kdc** is not specified.

primary_kdc
Identifies the primary KDC(s). Currently, this tag is used in only one case: If an attempt to get credentials fails because of an invalid password, the client software will attempt to contact the primary KDC, in case the user's password has just been changed, and the updated database has not been propagated to the replica servers yet. New in release 1.19.

sitename
Specifies the name of the host's site for the purpose of DNS-based KDC discovery for this realm. New in release 1.22.

v4_instance_convert
This subsection allows the administrator to configure exceptions to the **default_domain** mapping rule. It contains V4 instances (the tag name) which should be translated to some specific hostname (the tag value) as the second component in a Kerberos V5 principal name.

v4_realm
This relation is used by the krb524 library routines when converting a V5 principal name to a V4 principal name. It is used when the V4 realm name and the V5 realm name are not the same, but still share the same principal names and passwords. The tag value is the Kerberos V4 realm name.

[domain_realm]

The [domain_realm] section provides a translation from hostnames to Kerberos realms. Each tag is a domain name, providing the mapping for that domain and all subdomains. If the tag begins with a period (`.`) then it applies only to subdomains. The Kerberos realm may be identified either in the *realms* section or using DNS SRV records. Tag names should be in lower case. For example:

```
[domain_realm]
    crash.mit.edu = TEST.ATHENA.MIT.EDU
    .dev.mit.edu = TEST.ATHENA.MIT.EDU
    mit.edu = ATHENA.MIT.EDU
```

maps the host with the name `crash.mit.edu` into the `TEST.ATHENA.MIT.EDU` realm. The second entry maps all hosts under the domain `dev.mit.edu` into the `TEST.ATHENA.MIT.EDU` realm, but not the host with the name `dev.mit.edu`. That host is matched by the third entry, which maps the host `mit.edu` and all hosts under the domain `mit.edu` that do not match a preceding rule into the realm `ATHENA.MIT.EDU`.

If no translation entry applies to a hostname used for a service principal for a service ticket request, the library will try to get a referral to the appropriate realm from the client realm's KDC. If that does not succeed, the host's realm is considered to be the hostname's domain portion converted to uppercase, unless the **realm_try_domains** setting in [libdefaults] causes a different parent domain to be used.

[capaths]

In order to perform direct (non-hierarchical) cross-realm authentication, configuration is needed to determine the authentication paths between realms.

A client will use this section to find the authentication path between its realm and the realm of the server. The server will use this section to verify the authentication path used by the client, by checking the transited field of the received ticket.

There is a tag for each participating client realm, and each tag has subtags for each of the server realms. The value of the subtags is an intermediate realm which may participate in the cross-realm authentication. The subtags may be repeated if there is more then one intermediate realm. A value of "." means that the two realms share keys directly, and no intermediate realms should be allowed to participate.

Only those entries which will be needed on the client or the server need to be present. A client needs a tag for its local realm with subtags for all the realms of servers it will need to authenticate to. A server needs a tag for each realm of the clients it will serve, with a subtag of the server realm.

For example, `ANL.GOV`, `PNL.GOV`, and `NERSC.GOV` all wish to use the `ES.NET` realm as an intermediate realm. ANL has a sub realm of `TEST.ANL.GOV` which will authenticate with `NERSC.GOV` but not `PNL.GOV`. The [capaths] section for `ANL.GOV` systems would look like this:

```
[capaths]
    ANL.GOV = {
        TEST.ANL.GOV = .
        PNL.GOV = ES.NET
        NERSC.GOV = ES.NET
        ES.NET = .
    }
    TEST.ANL.GOV = {
        ANL.GOV = .
    }
    PNL.GOV = {
        ANL.GOV = ES.NET
    }
    NERSC.GOV = {
        ANL.GOV = ES.NET
    }
    ES.NET = {
        ANL.GOV = .
    }
```

The [capaths] section of the configuration file used on `NERSC.GOV` systems would look like this:

```
[capaths]
    NERSC.GOV = {
        ANL.GOV = ES.NET
        TEST.ANL.GOV = ES.NET
        TEST.ANL.GOV = ANL.GOV
        PNL.GOV = ES.NET
```

(continues on next page)

(continued from previous page)

```
        ES.NET = .
    }
    ANL.GOV = {
        NERSC.GOV = ES.NET
    }
    PNL.GOV = {
        NERSC.GOV = ES.NET
    }
    ES.NET = {
        NERSC.GOV = .
    }
    TEST.ANL.GOV = {
        NERSC.GOV = ANL.GOV
        NERSC.GOV = ES.NET
    }
```

When a subtag is used more than once within a tag, clients will use the order of values to determine the path. The order of values is not important to servers.

[appdefaults]

Each tag in the [appdefaults] section names a Kerberos V5 application or an option that is used by some Kerberos V5 application[s]. The value of the tag defines the default behaviors for that application.

For example:

```
[appdefaults]
    telnet = {
        ATHENA.MIT.EDU = {
            option1 = false
        }
    }
    telnet = {
        option1 = true
        option2 = true
    }
    ATHENA.MIT.EDU = {
        option2 = false
    }
    option2 = true
```

The above four ways of specifying the value of an option are shown in order of decreasing precedence. In this example, if telnet is running in the realm EXAMPLE.COM, it should, by default, have option1 and option2 set to true. However, a telnet program in the realm `ATHENA.MIT.EDU` should have `option1` set to false and `option2` set to true. Any other programs in ATHENA.MIT.EDU should have `option2` set to false by default. Any programs running in other realms should have `option2` set to true.

The list of specifiable options for each application may be found in that application's man pages. The application defaults specified here are overridden by those specified in the *realms* section.

[plugins]

- *pwqual* interface
- *kadm5_hook* interface
- *clpreauth* and *kdcpreauth* interfaces

Tags in the [plugins] section can be used to register dynamic plugin modules and to turn modules on and off. Not every krb5 pluggable interface uses the [plugins] section; the ones that do are documented here.

New in release 1.9.

Each pluggable interface corresponds to a subsection of [plugins]. All subsections support the same tags:

disable
: This tag may have multiple values. If there are values for this tag, then the named modules will be disabled for the pluggable interface.

enable_only
: This tag may have multiple values. If there are values for this tag, then only the named modules will be enabled for the pluggable interface.

module
: This tag may have multiple values. Each value is a string of the form `modulename:pathname`, which causes the shared object located at *pathname* to be registered as a dynamic module named *modulename* for the pluggable interface. If *pathname* is not an absolute path, it will be treated as relative to the **plugin_base_dir** value from *[libdefaults]*.

For pluggable interfaces where module order matters, modules registered with a **module** tag normally come first, in the order they are registered, followed by built-in modules in the order they are documented below. If **enable_only** tags are used, then the order of those tags overrides the normal module order.

The following subsections are currently supported within the [plugins] section:

ccselect interface

The ccselect subsection controls modules for credential cache selection within a cache collection. In addition to any registered dynamic modules, the following built-in modules exist (and may be disabled with the disable tag):

k5identity
: Uses a .k5identity file in the user's home directory to select a client principal

realm
: Uses the service realm to guess an appropriate cache from the collection

hostname
: If the service principal is host-based, uses the service hostname to guess an appropriate cache from the collection

pwqual interface

The pwqual subsection controls modules for the password quality interface, which is used to reject weak passwords when passwords are changed. The following built-in modules exist for this interface:

dict
Checks against the realm dictionary file

empty
Rejects empty passwords

hesiod
Checks against user information stored in Hesiod (only if Kerberos was built with Hesiod support)

princ
Checks against components of the principal name

kadm5_hook interface

The kadm5_hook interface provides plugins with information on principal creation, modification, password changes and deletion. This interface can be used to write a plugin to synchronize MIT Kerberos with another database such as Active Directory. No plugins are built in for this interface.

kadm5_auth interface

The kadm5_auth section (introduced in release 1.16) controls modules for the kadmin authorization interface, which determines whether a client principal is allowed to perform a kadmin operation. The following built-in modules exist for this interface:

acl
This module reads the *kadm5.acl* file, and authorizes operations which are allowed according to the rules in the file.

self
This module authorizes self-service operations including password changes, creation of new random keys, fetching the client's principal record or string attributes, and fetching the policy record associated with the client principal.

clpreauth and kdcpreauth interfaces

The clpreauth and kdcpreauth interfaces allow plugin modules to provide client and KDC preauthentication mechanisms. The following built-in modules exist for these interfaces:

pkinit
This module implements the PKINIT preauthentication mechanism.

encrypted_challenge
This module implements the encrypted challenge FAST factor.

encrypted_timestamp
This module implements the encrypted timestamp mechanism.

hostrealm interface

The hostrealm section (introduced in release 1.12) controls modules for the host-to-realm interface, which affects the local mapping of hostnames to realm names and the choice of default realm. The following built-in modules exist for this interface:

profile
: This module consults the [domain_realm] section of the profile for authoritative host-to-realm mappings, and the **default_realm** variable for the default realm.

dns
: This module looks for DNS records for fallback host-to-realm mappings and the default realm. It only operates if the **dns_lookup_realm** variable is set to true.

domain
: This module applies heuristics for fallback host-to-realm mappings. It implements the **realm_try_domains** variable, and uses the uppercased parent domain of the hostname if that does not produce a result.

localauth interface

The localauth section (introduced in release 1.12) controls modules for the local authorization interface, which affects the relationship between Kerberos principals and local system accounts. The following built-in modules exist for this interface:

default
: This module implements the **DEFAULT** type for **auth_to_local** values.

rule
: This module implements the **RULE** type for **auth_to_local** values.

names
: This module looks for an **auth_to_local_names** mapping for the principal name.

auth_to_local
: This module processes **auth_to_local** values in the default realm's section, and applies the default method if no **auth_to_local** values exist.

k5login
: This module authorizes a principal to a local account according to the account's .k5login(5) file.

an2ln
: This module authorizes a principal to a local account if the principal name maps to the local account name.

certauth interface

The certauth section (introduced in release 1.16) controls modules for the certificate authorization interface, which determines whether a certificate is allowed to preauthenticate a user via PKINIT. The following built-in modules exist for this interface:

pkinit_san
: This module authorizes the certificate if it contains a PKINIT Subject Alternative Name for the requested client principal, or a Microsoft UPN SAN matching the principal if **pkinit_allow_upn** is set to true for the realm.

pkinit_eku
: This module rejects the certificate if it does not contain an Extended Key Usage attribute consistent with the **pkinit_eku_checking** value for the realm.

dbmatch
This module authorizes or rejects the certificate according to whether it matches the **pkinit_cert_match** string attribute on the client principal, if that attribute is present.

PKINIT options

Note: The following are PKINIT-specific options. These values may be specified in [libdefaults] as global defaults, or within a realm-specific subsection of [libdefaults], or may be specified as realm-specific values in the [realms] section. A realm-specific value overrides, not adds to, a generic [libdefaults] specification. The search order is:

1. realm-specific subsection of [libdefaults]:

```
[libdefaults]
    EXAMPLE.COM = {
        pkinit_anchors = FILE:/usr/local/example.com.crt
    }
```

2. realm-specific value in the [realms] section:

```
[realms]
    OTHERREALM.ORG = {
        pkinit_anchors = FILE:/usr/local/otherrealm.org.crt
    }
```

3. generic value in the [libdefaults] section:

```
[libdefaults]
    pkinit_anchors = DIR:/usr/local/generic_trusted_cas/
```

Specifying PKINIT identity information

The syntax for specifying Public Key identity, trust, and revocation information for PKINIT is as follows:

FILE:*filename*[,*keyfilename*]
This option has context-specific behavior.

In **pkinit_identity** or **pkinit_identities**, *filename* specifies the name of a PEM-format file containing the user's certificate. If *keyfilename* is not specified, the user's private key is expected to be in *filename* as well. Otherwise, *keyfilename* is the name of the file containing the private key.

In **pkinit_anchors** or **pkinit_pool**, *filename* is assumed to be the name of an OpenSSL-style ca-bundle file.

DIR:*dirname*
This option has context-specific behavior.

In **pkinit_identity** or **pkinit_identities**, *dirname* specifies a directory with files named `*.crt` and `*.key` where the first part of the file name is the same for matching pairs of certificate and private key files. When a file with a name ending with `.crt` is found, a matching file ending with `.key` is assumed to contain the private key. If no such file is found, then the certificate in the `.crt` is not used.

In **pkinit_anchors** or **pkinit_pool**, *dirname* is assumed to be an OpenSSL-style hashed CA directory where each CA cert is stored in a file named `hash-of-ca-cert.#`. This infrastructure is encouraged, but all files in the directory will be examined and if they contain certificates (in PEM format), they will be used.

In **pkinit_revoke**, *dirname* is assumed to be an OpenSSL-style hashed CA directory where each revocation list is stored in a file named `hash-of-ca-cert.r#`. This infrastructure is encouraged, but all files in the directory will be examined and if they contain a revocation list (in PEM format), they will be used.

PKCS12:*filename*
filename is the name of a PKCS #12 format file, containing the user's certificate and private key.

PKCS11:[module_name=]*modname*[:slotid=*slot-id*][:token=*token-label*][:certid=*cert-id*][:certlabel=*cert-label*]
All keyword/values are optional. *modname* specifies the location of a library implementing PKCS #11. If a value is encountered with no keyword, it is assumed to be the *modname*. If no module-name is specified, the default is *PKCS11_MODNAME*. `slotid=` and/or `token=` may be specified to force the use of a particular smard card reader or token if there is more than one available. `certid=` and/or `certlabel=` may be specified to force the selection of a particular certificate on the device. Specifier values must not contain colon characters, as colons are always treated as separators. See the **pkinit_cert_match** configuration option for more ways to select a particular certificate to use for PKINIT.

ENV:*envvar*
envvar specifies the name of an environment variable which has been set to a value conforming to one of the previous values. For example, `ENV:X509_PROXY`, where environment variable `X509_PROXY` has been set to `FILE:/tmp/my_proxy.pem`.

PKINIT krb5.conf options

pkinit_anchors
Specifies the location of trusted anchor (root) certificates which the client trusts to sign KDC certificates. This option may be specified multiple times. These values from the config file are not used if the user specifies X509_anchors on the command line.

pkinit_cert_match
Specifies matching rules that the client certificate must match before it is used to attempt PKINIT authentication. If a user has multiple certificates available (on a smart card, or via other media), there must be exactly one certificate chosen before attempting PKINIT authentication. This option may be specified multiple times. All the available certificates are checked against each rule in order until there is a match of exactly one certificate.

The Subject and Issuer comparison strings are the **RFC 2253** string representations from the certificate Subject DN and Issuer DN values.

The syntax of the matching rules is:

[*relation-operator*]*component-rule* ...

where:

relation-operator
can be either &&, meaning all component rules must match, or ||, meaning only one component rule must match. The default is &&.

component-rule
can be one of the following. Note that there is no punctuation or whitespace between component rules.

<SUBJECT>*regular-expression*
<ISSUER>*regular-expression*
<SAN>*regular-expression*
<EKU>*extended-key-usage-list*
<KU>*key-usage-list*

extended-key-usage-list is a comma-separated list of required Extended Key Usage values. All values in the list must be present in the certificate. Extended Key Usage values can be:

- pkinit
- msScLogin
- clientAuth
- emailProtection

key-usage-list is a comma-separated list of required Key Usage values. All values in the list must be present in the certificate. Key Usage values can be:

- digitalSignature
- keyEncipherment

Examples:

```
pkinit_cert_match = ||<SUBJECT>.*DoE.*<SAN>.*@EXAMPLE.COM
pkinit_cert_match = &&<EKU>msScLogin,clientAuth<ISSUER>.*DoE.*
pkinit_cert_match = <EKU>msScLogin,clientAuth<KU>digitalSignature
```

pkinit_eku_checking
This option specifies what Extended Key Usage value the KDC certificate presented to the client must contain. (Note that if the KDC certificate has the pkinit SubjectAlternativeName encoded as the Kerberos TGS name, EKU checking is not necessary since the issuing CA has certified this as a KDC certificate.) The values recognized in the krb5.conf file are:

kpKDC
This is the default value and specifies that the KDC must have the id-pkinit-KPKdc EKU as defined in **RFC 4556**.

kpServerAuth
If **kpServerAuth** is specified, a KDC certificate with the id-kp-serverAuth EKU will be accepted. This key usage value is used in most commercially issued server certificates.

none
If **none** is specified, then the KDC certificate will not be checked to verify it has an acceptable EKU. The use of this option is not recommended.

pkinit_dh_min_bits
Specifies the group of the Diffie-Hellman key the client will attempt to use. The acceptable values are 1024, 2048, P-256, 4096, P-384, and P-521. The default is 2048. (P-256, P-384, and P-521 are new in release 1.22.)

pkinit_identities
Specifies the location(s) to be used to find the user's X.509 identity information. If this option is specified multiple times, each value is attempted in order until certificates are found. Note that these values are not used if the user specifies **X509_user_identity** on the command line.

pkinit_kdc_hostname
The presence of this option indicates that the client is willing to accept a KDC certificate with a dNSName SAN (Subject Alternative Name) rather than requiring the id-pkinit-san as defined in **RFC 4556**. This option may be specified multiple times. Its value should contain the acceptable hostname for the KDC (as contained in its certificate).

pkinit_pool
Specifies the location of intermediate certificates which may be used by the client to complete the trust chain between a KDC certificate and a trusted anchor. This option may be specified multiple times.

pkinit_require_crl_checking
The default certificate verification process will always check the available revocation information to see if a certificate has been revoked. If a match is found for the certificate in a CRL, verification fails. If the certificate being

verified is not listed in a CRL, or there is no CRL present for its issuing CA, and **pkinit_require_crl_checking** is false, then verification succeeds.

However, if **pkinit_require_crl_checking** is true and there is no CRL information available for the issuing CA, then verification fails.

pkinit_require_crl_checking should be set to true if the policy is such that up-to-date CRLs must be present for every CA.

pkinit_revoke
Specifies the location of Certificate Revocation List (CRL) information to be used by the client when verifying the validity of the KDC certificate presented. This option may be specified multiple times.

Parameter expansion

Starting with release 1.11, several variables, such as **default_keytab_name**, allow parameters to be expanded. Valid parameters are:

%{TEMP}	Temporary directory
%{uid}	Unix real UID or Windows SID
%{euid}	Unix effective user ID or Windows SID
%{USERID}	Same as %{uid}
%{null}	Empty string
%{LIBDIR}	Installation library directory
%{BINDIR}	Installation binary directory
%{SBINDIR}	Installation admin binary directory
%{username}	(Unix) Username of effective user ID
%{APPDATA}	(Windows) Roaming application data for current user
%{COMMON_APPDATA}	(Windows) Application data for all users
%{LOCAL_APPDATA}	(Windows) Local application data for current user
%{SYSTEM}	(Windows) Windows system folder
%{WINDOWS}	(Windows) Windows folder
%{USERCONFIG}	(Windows) Per-user MIT krb5 config file directory
%{COMMONCONFIG}	(Windows) Common MIT krb5 config file directory

Sample krb5.conf file

Here is an example of a generic krb5.conf file:

```
[libdefaults]
    default_realm = ATHENA.MIT.EDU
    dns_lookup_kdc = true
    dns_lookup_realm = false

[realms]
    ATHENA.MIT.EDU = {
        kdc = kerberos.mit.edu
        kdc = kerberos-1.mit.edu
        kdc = kerberos-2.mit.edu
        admin_server = kerberos.mit.edu
        primary_kdc = kerberos.mit.edu
    }
```

(continues on next page)

(continued from previous page)

```
    EXAMPLE.COM = {
        kdc = kerberos.example.com
        kdc = kerberos-1.example.com
        admin_server = kerberos.example.com
    }

[domain_realm]
    mit.edu = ATHENA.MIT.EDU

[capaths]
    ATHENA.MIT.EDU = {
            EXAMPLE.COM = .
    }
    EXAMPLE.COM = {
            ATHENA.MIT.EDU = .
    }
```

FILES

`/etc/krb5.conf`

SEE ALSO

syslog(3)

2.1.2 kdc.conf

The kdc.conf file supplements *krb5.conf* for programs which are typically only used on a KDC, such as the *krb5kdc* and *kadmind* daemons and the *kdb5_util* program. Relations documented here may also be specified in krb5.conf; for the KDC programs mentioned, krb5.conf and kdc.conf will be merged into a single configuration profile.

Normally, the kdc.conf file is found in the KDC state directory, *LOCALSTATEDIR*`/krb5kdc`. You can override the default location by setting the environment variable **KRB5_KDC_PROFILE**.

Please note that you need to restart the KDC daemon for any configuration changes to take effect.

Structure

The kdc.conf file is set up in the same format as the *krb5.conf* file.

Sections

The kdc.conf file may contain the following sections:

[kdcdefaults]	Default values for KDC behavior
[realms]	Realm-specific database configuration and settings
[dbdefaults]	Default database settings
[dbmodules]	Per-database settings
[logging]	Controls how Kerberos daemons perform logging

[kdcdefaults]

Some relations in the [kdcdefaults] section specify default values for realm variables, to be used if the [realms] subsection does not contain a relation for the tag. See the *[realms]* section for the definitions of these relations.

- **host_based_services**
- **kdc_listen**
- **kdc_ports**
- **kdc_tcp_listen**
- **kdc_tcp_ports**
- **no_host_referral**
- **restrict_anonymous_to_tgt**

The following [kdcdefaults] variables have no per-realm equivalent:

kdc_max_dgram_reply_size
: Specifies the maximum packet size that can be sent over UDP. The default value is 4096 bytes.

kdc_tcp_listen_backlog
: (Integer.) Set the size of the listen queue length for the KDC daemon. The value may be limited by OS settings. The default value is 5.

spake_preauth_kdc_challenge
: (String.) Specifies the group for a SPAKE optimistic challenge. See the **spake_preauth_groups** variable in *[libdefaults]* for possible values. The default is not to issue an optimistic challenge. (New in release 1.17.)

[realms]

Each tag in the [realms] section is the name of a Kerberos realm. The value of the tag is a subsection where the relations define KDC parameters for that particular realm. The following example shows how to define one parameter for the ATHENA.MIT.EDU realm:

```
[realms]
    ATHENA.MIT.EDU = {
        max_renewable_life = 7d 0h 0m 0s
    }
```

The following tags may be specified in a [realms] subsection:

acl_file
(String.) Location of the access control list file that *kadmind* uses to determine which principals are allowed which permissions on the Kerberos database. To operate without an ACL file, set this relation to the empty string with `acl_file = ""`. The default value is *LOCALSTATEDIR*`/krb5kdc/kadm5.acl`. For more information on Kerberos ACL file see *kadm5.acl*.

database_module
(String.) This relation indicates the name of the configuration section under *[dbmodules]* for database-specific parameters used by the loadable database library. The default value is the realm name. If this configuration section does not exist, default values will be used for all database parameters.

database_name
(String, deprecated.) This relation specifies the location of the Kerberos database for this realm, if the DB2 module is being used and the *[dbmodules]* configuration section does not specify a database name. The default value is *LOCALSTATEDIR*`/krb5kdc/principal`.

default_principal_expiration
(abstime string.) Specifies the default expiration date of principals created in this realm. The default value is 0, which means no expiration date.

default_principal_flags
(Flag string.) Specifies the default attributes of principals created in this realm. The format for this string is a comma-separated list of flags, with '+' before each flag that should be enabled and '-' before each flag that should be disabled. The **postdateable**, **forwardable**, **tgt-based**, **renewable**, **proxiable**, **dup-skey**, **allow-tickets**, and **service** flags default to enabled.

There are a number of possible flags:

allow-tickets
Enabling this flag means that the KDC will issue tickets for this principal. Disabling this flag essentially deactivates the principal within this realm.

dup-skey
Enabling this flag allows the KDC to issue user-to-user service tickets for this principal.

forwardable
Enabling this flag allows the principal to obtain forwardable tickets.

hwauth
If this flag is enabled, then the principal is required to preauthenticate using a hardware device before receiving any tickets.

no-auth-data-required
Enabling this flag prevents PAC or AD-SIGNEDPATH data from being added to service tickets for the principal.

ok-as-delegate
If this flag is enabled, it hints the client that credentials can and should be delegated when authenticating to the service.

ok-to-auth-as-delegate
Enabling this flag allows the principal to use S4USelf tickets.

postdateable
Enabling this flag allows the principal to obtain postdateable tickets.

preauth
If this flag is enabled on a client principal, then that principal is required to preauthenticate to the KDC before receiving any tickets. On a service principal, enabling this flag means that service tickets for this principal will only be issued to clients with a TGT that has the preauthenticated bit set.

proxiable
Enabling this flag allows the principal to obtain proxy tickets.

pwchange
Enabling this flag forces a password change for this principal.

pwservice
If this flag is enabled, it marks this principal as a password change service. This should only be used in special cases, for example, if a user's password has expired, then the user has to get tickets for that principal without going through the normal password authentication in order to be able to change the password.

renewable
Enabling this flag allows the principal to obtain renewable tickets.

service
Enabling this flag allows the the KDC to issue service tickets for this principal. In release 1.17 and later, user-to-user service tickets are still allowed if the **dup-skey** flag is set.

tgt-based
Enabling this flag allows a principal to obtain tickets based on a ticket-granting-ticket, rather than repeating the authentication process that was used to obtain the TGT.

dict_file
(String.) Location of the dictionary file containing strings that are not allowed as passwords. The file should contain one string per line, with no additional whitespace. If none is specified or if there is no policy assigned to the principal, no dictionary checks of passwords will be performed.

disable_pac
(Boolean value.) If true, the KDC will not issue PACs for this realm, and S4U2Self and S4U2Proxy operations will be disabled. The default is false, which will permit the KDC to issue PACs. New in release 1.20.

encrypted_challenge_indicator
(String.) Specifies the authentication indicator value that the KDC asserts into tickets obtained using FAST encrypted challenge pre-authentication. New in 1.16.

host_based_services
(Whitespace- or comma-separated list.) Lists services which will get host-based referral processing even if the server principal is not marked as host-based by the client.

iprop_enable
(Boolean value.) Specifies whether incremental database propagation is enabled. The default value is false.

iprop_ulogsize
(Integer.) Specifies the maximum number of log entries to be retained for incremental propagation. The default value is 1000. Prior to release 1.11, the maximum value was 2500. New in release 1.19.

iprop_master_ulogsize
The name for **iprop_ulogsize** prior to release 1.19. Its value is used as a fallback if **iprop_ulogsize** is not specified.

iprop_replica_poll
(Delta time string.) Specifies how often the replica KDC polls for new updates from the primary. The default value is `2m` (that is, two minutes). New in release 1.17.

iprop_slave_poll
(Delta time string.) The name for **iprop_replica_poll** prior to release 1.17. Its value is used as a fallback if **iprop_replica_poll** is not specified.

iprop_listen
(Whitespace- or comma-separated list.) Specifies the iprop RPC listening addresses and/or ports for the *kadmind* daemon. Each entry may be an interface address, a port number, or an address and port number separated by

a colon. If the address contains colons, enclose it in square brackets. If no address is specified, the wildcard address is used. If kadmind fails to bind to any of the specified addresses, it will fail to start. The default (when **iprop_enable** is true) is to bind to the wildcard address at the port specified in **iprop_port**. New in release 1.15.

iprop_port
(Port number.) Specifies the port number to be used for incremental propagation. When **iprop_enable** is true, this relation is required in the replica KDC configuration file, and this relation or **iprop_listen** is required in the primary configuration file, as there is no default port number. Port numbers specified in **iprop_listen** entries will override this port number for the *kadmind* daemon.

iprop_resync_timeout
(Delta time string.) Specifies the amount of time to wait for a full propagation to complete. This is optional in configuration files, and is used by replica KDCs only. The default value is 5 minutes (`5m`). New in release 1.11.

iprop_logfile
(File name.) Specifies where the update log file for the realm database is to be stored. The default is to use the **database_name** entry from the realms section of the krb5 config file, with `.ulog` appended. (NOTE: If **database_name** isn't specified in the realms section, perhaps because the LDAP database back end is being used, or the file name is specified in the [dbmodules] section, then the hard-coded default for **database_name** is used. Determination of the **iprop_logfile** default value will not use values from the [dbmodules] section.)

kadmind_listen
(Whitespace- or comma-separated list.) Specifies the kadmin RPC listening addresses and/or ports for the *kadmind* daemon. Each entry may be an interface address, a port number, an address and port number separated by a colon, or a UNIX domain socket pathname. If the address contains colons, enclose it in square brackets. If no address is specified, the wildcard address is used. To disable listening for kadmin RPC connections, set this relation to the empty string with `kadmind_listen = ""`. If kadmind fails to bind to any of the specified addresses, it will fail to start. The default is to bind to the wildcard address at the port specified in **kadmind_port**, or the standard kadmin port (749). New in release 1.15.

kadmind_port
(Port number.) Specifies the port on which the *kadmind* daemon is to listen for this realm. Port numbers specified in **kadmind_listen** entries will override this port number. The assigned port for kadmind is 749, which is used by default.

key_stash_file
(String.) Specifies the location where the master key has been stored (via kdb5_util stash). The default is *LOCALSTATEDIR*`/krb5kdc/.k5.REALM`, where *REALM* is the Kerberos realm.

kdc_listen
(Whitespace- or comma-separated list.) Specifies the listening addresses and/or ports for the *krb5kdc* daemon. Each entry may be an interface address, a port number, an address and port number separated by a colon, or a UNIX domain socket pathname. If the address contains colons, enclose it in square brackets. If no address is specified, the wildcard address is used. If no port is specified, the standard port (88) is used. To disable listening on UDP, set this relation to the empty string with `kdc_listen = ""`. If the KDC daemon fails to bind to any of the specified addresses, it will fail to start. The default is to bind to the wildcard address on the standard port. New in release 1.15.

kdc_ports
(Whitespace- or comma-separated list, deprecated.) Prior to release 1.15, this relation lists the ports for the *krb5kdc* daemon to listen on for UDP requests. In release 1.15 and later, it has the same meaning as **kdc_listen** if that relation is not defined.

kdc_tcp_listen
(Whitespace- or comma-separated list.) Specifies the TCP listening addresses and/or ports for the *krb5kdc* daemon. The syntax is identical to that of **kdc_listen**. To disable listening on TCP, set this relation to the empty string with `kdc_tcp_listen = ""`. The default is to bind to the same addresses and ports as for UDP. New in release 1.15.

kdc_tcp_ports
(Whitespace- or comma-separated list, deprecated.) Prior to release 1.15, this relation lists the ports for the *krb5kdc* daemon to listen on for UDP requests. In release 1.15 and later, it has the same meaning as **kdc_tcp_listen** if that relation is not defined.

kpasswd_listen
(Comma-separated list.) Specifies the kpasswd listening addresses and/or ports for the *kadmind* daemon. Each entry may be an interface address, a port number, an address and port number separated by a colon, or a UNIX domain socket pathname. If the address contains colons, enclose it in square brackets. If no address is specified, the wildcard address is used. To disable listening for kpasswd requests, set this relation to the empty string with `kpasswd_listen = ""`. If kadmind fails to bind to any of the specified addresses, it will fail to start. The default is to bind to the wildcard address at the port specified in **kpasswd_port**, or the standard kpasswd port (464). New in release 1.15.

kpasswd_port
(Port number.) Specifies the port on which the *kadmind* daemon is to listen for password change requests for this realm. Port numbers specified in **kpasswd_listen** entries will override this port number. The assigned port for password change requests is 464, which is used by default.

master_key_name
(String.) Specifies the name of the principal associated with the master key. The default is `K/M`.

master_key_type
(Key type string.) Specifies the master key's key type. The default value for this is `aes256-cts-hmac-sha1-96`. For a list of all possible values, see *Encryption types*.

max_life
(duration string.) Specifies the maximum time period for which a ticket may be valid in this realm. The default value is 24 hours.

max_renewable_life
(duration string.) Specifies the maximum time period during which a valid ticket may be renewed in this realm. The default value is 0.

no_host_referral
(Whitespace- or comma-separated list.) Lists services to block from getting host-based referral processing, even if the client marks the server principal as host-based or the service is also listed in **host_based_services**. `no_host_referral = *` will disable referral processing altogether.

reject_bad_transit
(Boolean value.) If set to true, the KDC will check the list of transited realms for cross-realm tickets against the transit path computed from the realm names and the capaths section of its *krb5.conf* file; if the path in the ticket to be issued contains any realms not in the computed path, the ticket will not be issued, and an error will be returned to the client instead. If this value is set to false, such tickets will be issued anyways, and it will be left up to the application server to validate the realm transit path.

If the disable-transited-check flag is set in the incoming request, this check is not performed at all. Having the **reject_bad_transit** option will cause such ticket requests to be rejected always.

This transit path checking and config file option currently apply only to TGS requests.

The default value is true.

restrict_anonymous_to_tgt
(Boolean value.) If set to true, the KDC will reject ticket requests from anonymous principals to service principals other than the realm's ticket-granting service. This option allows anonymous PKINIT to be enabled for use as FAST armor tickets without allowing anonymous authentication to services. The default value is false. New in release 1.9.

spake_preauth_indicator
(String.) Specifies an authentication indicator value that the KDC asserts into tickets obtained using SPAKE pre-authentication. The default is not to add any indicators. This option may be specified multiple times. New in release 1.17.

supported_enctypes
(List of *key*:*salt* strings.) Specifies the default key/salt combinations of principals for this realm. Any principals created through *kadmin* will have keys of these types. The default value for this tag is `aes256-cts-hmac-sha1-96:normal aes128-cts-hmac-sha1-96:normal`. For lists of possible values, see *Keysalt lists*.

[dbdefaults]

The [dbdefaults] section specifies default values for some database parameters, to be used if the [dbmodules] subsection does not contain a relation for the tag. See the *[dbmodules]* section for the definitions of these relations.

- **ldap_kerberos_container_dn**
- **ldap_kdc_dn**
- **ldap_kdc_sasl_authcid**
- **ldap_kdc_sasl_authzid**
- **ldap_kdc_sasl_mech**
- **ldap_kdc_sasl_realm**
- **ldap_kadmind_dn**
- **ldap_kadmind_sasl_authcid**
- **ldap_kadmind_sasl_authzid**
- **ldap_kadmind_sasl_mech**
- **ldap_kadmind_sasl_realm**
- **ldap_service_password_file**
- **ldap_conns_per_server**

[dbmodules]

The [dbmodules] section contains parameters used by the KDC database library and database modules. Each tag in the [dbmodules] section is the name of a Kerberos realm or a section name specified by a realm's **database_module** parameter. The following example shows how to define one database parameter for the ATHENA.MIT.EDU realm:

```
[dbmodules]
    ATHENA.MIT.EDU = {
        disable_last_success = true
    }
```

The following tags may be specified in a [dbmodules] subsection:

database_name
This DB2-specific tag indicates the location of the database in the filesystem. The default is *LOCALSTATEDIR*`/krb5kdc/principal`.

db_library
This tag indicates the name of the loadable database module. The value should be `db2` for the DB2 module, `klmdb` for the LMDB module, or `kldap` for the LDAP module.

disable_last_success
If set to `true`, suppresses KDC updates to the "Last successful authentication" field of principal entries requiring preauthentication. Setting this flag may improve performance. (Principal entries which do not require preauthentication never update the "Last successful authentication" field.). First introduced in release 1.9.

disable_lockout
If set to `true`, suppresses KDC updates to the "Last failed authentication" and "Failed password attempts" fields of principal entries requiring preauthentication. Setting this flag may improve performance, but also disables account lockout. First introduced in release 1.9.

ldap_conns_per_server
This LDAP-specific tag indicates the number of connections to be maintained per LDAP server.

ldap_kdc_dn and ldap_kadmind_dn
These LDAP-specific tags indicate the default DN for binding to the LDAP server. The *krb5kdc* daemon uses **ldap_kdc_dn**, while the *kadmind* daemon and other administrative programs use **ldap_kadmind_dn**. The kadmind DN must have the rights to read and write the Kerberos data in the LDAP database. The KDC DN must have the same rights, unless **disable_lockout** and **disable_last_success** are true, in which case it only needs to have rights to read the Kerberos data. These tags are ignored if a SASL mechanism is set with **ldap_kdc_sasl_mech** or **ldap_kadmind_sasl_mech**.

ldap_kdc_sasl_mech and ldap_kadmind_sasl_mech
These LDAP-specific tags specify the SASL mechanism (such as `EXTERNAL`) to use when binding to the LDAP server. New in release 1.13.

ldap_kdc_sasl_authcid and ldap_kadmind_sasl_authcid
These LDAP-specific tags specify the SASL authentication identity to use when binding to the LDAP server. Not all SASL mechanisms require an authentication identity. If the SASL mechanism requires a secret (such as the password for `DIGEST-MD5`), these tags also determine the name within the **ldap_service_password_file** where the secret is stashed. New in release 1.13.

ldap_kdc_sasl_authzid and ldap_kadmind_sasl_authzid
These LDAP-specific tags specify the SASL authorization identity to use when binding to the LDAP server. In most circumstances they do not need to be specified. New in release 1.13.

ldap_kdc_sasl_realm and ldap_kadmind_sasl_realm
These LDAP-specific tags specify the SASL realm to use when binding to the LDAP server. In most circumstances they do not need to be set. New in release 1.13.

ldap_kerberos_container_dn
This LDAP-specific tag indicates the DN of the container object where the realm objects will be located.

ldap_servers
This LDAP-specific tag indicates the list of LDAP servers that the Kerberos servers can connect to. The list of LDAP servers is whitespace-separated. The LDAP server is specified by a LDAP URI. It is recommended to use `ldapi:` or `ldaps:` URLs to connect to the LDAP server.

ldap_service_password_file
This LDAP-specific tag indicates the file containing the stashed passwords (created by `kdb5_ldap_util stashsrvpw`) for the **ldap_kdc_dn** and **ldap_kadmind_dn** objects, or for the **ldap_kdc_sasl_authcid** or **ldap_kadmind_sasl_authcid** names for SASL authentication. This file must be kept secure.

mapsize
This LMDB-specific tag indicates the maximum size of the two database environments in megabytes. The default value is 128. Increase this value to address "Environment mapsize limit reached" errors. New in release 1.17.

max_readers
: This LMDB-specific tag indicates the maximum number of concurrent reading processes for the databases. The default value is 128. New in release 1.17.

nosync
: This LMDB-specific tag can be set to improve the throughput of kadmind and other administrative agents, at the expense of durability (recent database changes may not survive a power outage or other sudden reboot). It does not affect the throughput of the KDC. The default value is false. New in release 1.17.

unlockiter
: If set to `true`, this DB2-specific tag causes iteration operations to release the database lock while processing each principal. Setting this flag to `true` can prevent extended blocking of KDC or kadmin operations when dumps of large databases are in progress. First introduced in release 1.13.

The following tag may be specified directly in the [dbmodules] section to control where database modules are loaded from:

db_module_dir
: This tag controls where the plugin system looks for database modules. The value should be an absolute path.

[logging]

The [logging] section indicates how *krb5kdc* and *kadmind* perform logging. It may contain the following relations:

admin_server
: Specifies how *kadmind* performs logging.

kdc
: Specifies how *krb5kdc* performs logging.

default
: Specifies how either daemon performs logging in the absence of relations specific to the daemon.

debug
: (Boolean value.) Specifies whether debugging messages are included in log outputs other than SYSLOG. Debugging messages are always included in the system log output because syslog performs its own priority filtering. The default value is false. New in release 1.15.

Logging specifications may have the following forms:

FILE=*filename* or FILE:*filename*
: This value causes the daemon's logging messages to go to the *filename*. If the = form is used, the file is overwritten. If the `:` form is used, the file is appended to.

STDERR
: This value causes the daemon's logging messages to go to its standard error stream.

CONSOLE
: This value causes the daemon's logging messages to go to the console, if the system supports it.

DEVICE=<*devicename*>
: This causes the daemon's logging messages to go to the specified device.

SYSLOG[:*severity*[:*facility*]]
: This causes the daemon's logging messages to go to the system log.

  For backward compatibility, a severity argument may be specified, and must be specified in order to specify a facility. This argument will be ignored.

The facility argument specifies the facility under which the messages are logged. This may be any of the following facilities supported by the syslog(3) call minus the LOG_ prefix: **KERN**, **USER**, **MAIL**, **DAEMON**, **AUTH**, **LPR**, **NEWS**, **UUCP**, **CRON**, and **LOCAL0** through **LOCAL7**. If no facility is specified, the default is **AUTH**.

In the following example, the logging messages from the KDC will go to the console and to the system log under the facility LOG_DAEMON, and the logging messages from the administrative server will be appended to the file `/var/adm/kadmin.log` and sent to the device `/dev/tty04`.

```
[logging]
    kdc = CONSOLE
    kdc = SYSLOG:INFO:DAEMON
    admin_server = FILE:/var/adm/kadmin.log
    admin_server = DEVICE=/dev/tty04
```

If no logging specification is given, the default is to use syslog. To disable logging entirely, specify `default = DEVICE=/dev/null`.

[otp]

Each subsection of [otp] is the name of an OTP token type. The tags within the subsection define the configuration required to forward a One Time Password request to a RADIUS server.

For each token type, the following tags may be specified:

server
This is the server to send the RADIUS request to. It can be a hostname with optional port, an ip address with optional port, or a Unix domain socket address. The default is *LOCALSTATEDIR*`/krb5kdc/<name>.socket`.

secret
This tag indicates a filename (which may be relative to *LOCALSTATEDIR*`/krb5kdc`) containing the secret used to encrypt the RADIUS packets. The secret should appear in the first line of the file by itself; leading and trailing whitespace on the line will be removed. If the value of **server** is a Unix domain socket address, this tag is optional, and an empty secret will be used if it is not specified. Otherwise, this tag is required.

timeout
An integer which specifies the time in seconds during which the KDC should attempt to contact the RADIUS server. This tag is the total time across all retries and should be less than the time which an OTP value remains valid for. The default is 5 seconds.

retries
This tag specifies the number of retries to make to the RADIUS server. The default is 3 retries (4 tries).

strip_realm
If this tag is `true`, the principal without the realm will be passed to the RADIUS server. Otherwise, the realm will be included. The default value is `true`.

indicator
This tag specifies an authentication indicator to be included in the ticket if this token type is used to authenticate. This option may be specified multiple times. (New in release 1.14.)

In the following example, requests are sent to a remote server via UDP:

```
[otp]
    MyRemoteTokenType = {
        server = radius.mydomain.com:1812
        secret = SEmfiajf42$
        timeout = 15
```

(continues on next page)

(continued from previous page)

```
        retries = 5
        strip_realm = true
    }
```

An implicit default token type named `DEFAULT` is defined for when the per-principal configuration does not specify a token type. Its configuration is shown below. You may override this token type to something applicable for your situation:

```
[otp]
    DEFAULT = {
        strip_realm = false
    }
```

PKINIT options

Note: The following are pkinit-specific options. These values may be specified in [kdcdefaults] as global defaults, or within a realm-specific subsection of [realms]. Also note that a realm-specific value over-rides, does not add to, a generic [kdcdefaults] specification. The search order is:

1. realm-specific subsection of [realms]:

   ```
   [realms]
       EXAMPLE.COM = {
           pkinit_anchors = FILE:/usr/local/example.com.crt
       }
   ```

2. generic value in the [kdcdefaults] section:

   ```
   [kdcdefaults]
       pkinit_anchors = DIR:/usr/local/generic_trusted_cas/
   ```

For information about the syntax of some of these options, see *Specifying PKINIT identity information* in *krb5.conf*.

pkinit_anchors
: Specifies the location of trusted anchor (root) certificates which the KDC trusts to sign client certificates. This option is required if pkinit is to be supported by the KDC. This option may be specified multiple times.

pkinit_dh_min_bits
: Specifies the minimum strength of Diffie-Hellman group the KDC is willing to accept for key exchange. Valid values in order of increasing strength are 1024, 2048, P-256, 4096, P-384, and P-521. The default is 2048. (P-256, P-384, and P-521 are new in release 1.22.)

pkinit_allow_upn
: Specifies that the KDC is willing to accept client certificates with the Microsoft UserPrincipalName (UPN) Subject Alternative Name (SAN). This means the KDC accepts the binding of the UPN in the certificate to the Kerberos principal name. The default value is false.

  Without this option, the KDC will only accept certificates with the id-pkinit-san as defined in **RFC 4556**. There is currently no option to disable SAN checking in the KDC.

pkinit_eku_checking
: This option specifies what Extended Key Usage (EKU) values the KDC is willing to accept in client certificates. The values recognized in the kdc.conf file are:

kpClientAuth
This is the default value and specifies that client certificates must have the id-pkinit-KPClientAuth EKU as defined in RFC 4556.

scLogin
If scLogin is specified, client certificates with the Microsoft Smart Card Login EKU (id-ms-kp-sc-logon) will be accepted.

none
If none is specified, then client certificates will not be checked to verify they have an acceptable EKU. The use of this option is not recommended.

pkinit_identity
Specifies the location of the KDC's X.509 identity information. This option is required if pkinit is to be supported by the KDC.

pkinit_indicator
Specifies an authentication indicator to include in the ticket if pkinit is used to authenticate. This option may be specified multiple times. (New in release 1.14.)

pkinit_pool
Specifies the location of intermediate certificates which may be used by the KDC to complete the trust chain between a client's certificate and a trusted anchor. This option may be specified multiple times.

pkinit_revoke
Specifies the location of Certificate Revocation List (CRL) information to be used by the KDC when verifying the validity of client certificates. This option may be specified multiple times.

pkinit_require_crl_checking
The default certificate verification process will always check the available revocation information to see if a certificate has been revoked. If a match is found for the certificate in a CRL, verification fails. If the certificate being verified is not listed in a CRL, or there is no CRL present for its issuing CA, and **pkinit_require_crl_checking** is false, then verification succeeds.

However, if **pkinit_require_crl_checking** is true and there is no CRL information available for the issuing CA, then verification fails.

pkinit_require_crl_checking should be set to true if the policy is such that up-to-date CRLs must be present for every CA.

pkinit_require_freshness
Specifies whether to require clients to include a freshness token in PKINIT requests. The default value is false. (New in release 1.17.)

Encryption types

Any tag in the configuration files which requires a list of encryption types can be set to some combination of the following strings. Encryption types marked as "weak" and "deprecated" are available for compatibility but not recommended for use.

des3-cbc-raw	Triple DES cbc mode raw (weak)
des3-cbc-sha1 des3-hmac-sha1 des3-cbc-sha1-kd	Triple DES cbc mode with HMAC/sha1 (deprecated)
aes256-cts-hmac-sha1-96 aes256-cts aes256-sha1	AES-256 CTS mode with 96-bit SHA-1 HMAC
aes128-cts-hmac-sha1-96 aes128-cts aes128-sha1	AES-128 CTS mode with 96-bit SHA-1 HMAC
aes256-cts-hmac-sha384-192 aes256-sha2	AES-256 CTS mode with 192-bit SHA-384 HMAC
aes128-cts-hmac-sha256-128 aes128-sha2	AES-128 CTS mode with 128-bit SHA-256 HMAC
arcfour-hmac rc4-hmac arcfour-hmac-md5	RC4 with HMAC/MD5 (deprecated)
arcfour-hmac-exp rc4-hmac-exp arcfour-hmac-md5-exp	Exportable RC4 with HMAC/MD5 (weak)
camellia256-cts-cmac camellia256-cts	Camellia-256 CTS mode with CMAC
camellia128-cts-cmac camellia128-cts	Camellia-128 CTS mode with CMAC
des3	The triple DES family: des3-cbc-sha1
aes	The AES family: aes256-cts-hmac-sha1-96, aes128-cts-hmac-sha1-96, aes256-cts-hmac-sha384-192, and aes128-cts-hmac-sha256-128
rc4	The RC4 family: arcfour-hmac
camellia	The Camellia family: camellia256-cts-cmac and camellia128-cts-cmac

The string **DEFAULT** can be used to refer to the default set of types for the variable in question. Types or families can be removed from the current list by prefixing them with a minus sign ("-"). Types or families can be prefixed with a plus sign ("+") for symmetry; it has the same meaning as just listing the type or family. For example, "`DEFAULT -rc4`" would be the default set of encryption types with RC4 types removed, and "`des3 DEFAULT`" would be the default set of encryption types with triple DES types moved to the front.

While **aes128-cts** and **aes256-cts** are supported for all Kerberos operations, they are not supported by very old versions of our GSSAPI implementation (krb5-1.3.1 and earlier). Services running versions of krb5 without AES support must not be given keys of these encryption types in the KDC database.

The **aes128-sha2** and **aes256-sha2** encryption types are new in release 1.15. Services running versions of krb5 without support for these newer encryption types must not be given keys of these encryption types in the KDC database.

Keysalt lists

Kerberos keys for users are usually derived from passwords. Kerberos commands and configuration parameters that affect generation of keys take lists of enctype-salttype ("keysalt") pairs, known as *keysalt lists*. Each keysalt pair is an enctype name followed by a salttype name, in the format *enc*:*salt*. Individual keysalt list members are separated by comma (",") characters or space characters. For example:

```
kadmin -e aes256-cts:normal,aes128-cts:normal
```

would start up kadmin so that by default it would generate password-derived keys for the **aes256-cts** and **aes128-cts** encryption types, using a **normal** salt.

To ensure that people who happen to pick the same password do not have the same key, Kerberos 5 incorporates more information into the key using something called a salt. The supported salt types are as follows:

normal	default for Kerberos Version 5
norealm	same as the default, without using realm information
onlyrealm	uses only realm information as the salt
special	generate a random salt

Sample kdc.conf File

Here's an example of a kdc.conf file:

```
[kdcdefaults]
    kdc_listen = 88
    kdc_tcp_listen = 88
[realms]
    ATHENA.MIT.EDU = {
        kadmind_port = 749
        max_life = 12h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        master_key_type = aes256-cts-hmac-sha1-96
        supported_enctypes = aes256-cts-hmac-sha1-96:normal aes128-cts-hmac-sha1-
↪96:normal
        database_module = openldap_ldapconf
    }

[logging]
    kdc = FILE:/usr/local/var/krb5kdc/kdc.log
    admin_server = FILE:/usr/local/var/krb5kdc/kadmin.log

[dbdefaults]
    ldap_kerberos_container_dn = cn=krbcontainer,dc=mit,dc=edu

[dbmodules]
    openldap_ldapconf = {
        db_library = kldap
        disable_last_success = true
        ldap_kdc_dn = "cn=krbadmin,dc=mit,dc=edu"
            # this object needs to have read rights on
            # the realm container and principal subtrees
        ldap_kadmind_dn = "cn=krbadmin,dc=mit,dc=edu"
            # this object needs to have read and write rights on
            # the realm container and principal subtrees
        ldap_service_password_file = /etc/kerberos/service.keyfile
        ldap_servers = ldaps://kerberos.mit.edu
        ldap_conns_per_server = 5
    }
```

FILES

LOCALSTATEDIR/krb5kdc/kdc.conf

SEE ALSO

krb5.conf, *krb5kdc*, *kadm5.acl*

2.1.3 kadm5.acl

DESCRIPTION

The Kerberos *kadmind* daemon uses an Access Control List (ACL) file to manage access rights to the Kerberos database. For operations that affect principals, the ACL file also controls which principals can operate on which other principals.

The default location of the Kerberos ACL file is *LOCALSTATEDIR*/krb5kdc/kadm5.acl unless this is overridden by the *acl_file* variable in *kdc.conf*.

SYNTAX

Empty lines and lines starting with the sharp sign (#) are ignored. Lines containing ACL entries have the format:

```
principal  permissions  [target_principal  [restrictions] ]
```

Note: Line order in the ACL file is important. The first matching entry will control access for an actor principal on a target principal.

principal
(Partially or fully qualified Kerberos principal name.) Specifies the principal whose permissions are to be set.

Each component of the name may be wildcarded using the * character.

permissions
Specifies what operations may or may not be performed by a *principal* matching a particular entry. This is a string of one or more of the following list of characters or their upper-case counterparts. If the character is *upper-case*, then the operation is disallowed. If the character is *lower-case*, then the operation is permitted.

a	[Dis]allows the addition of principals or policies
c	[Dis]allows the changing of passwords for principals
d	[Dis]allows the deletion of principals or policies
e	[Dis]allows the extraction of principal keys
i	[Dis]allows inquiries about principals or policies
l	[Dis]allows the listing of all principals or policies
m	[Dis]allows the modification of principals or policies
p	[Dis]allows the propagation of the principal database (used in *Incremental database propagation*)
s	[Dis]allows the explicit setting of the key for a principal
x	Short for admcilsp. All privileges (except e)
*	Same as x.

Note: The `extract` privilege is not included in the wildcard privilege; it must be explicitly assigned. This privilege allows the user to extract keys from the database, and must be handled with great care to avoid disclosure of important keys like those of the kadmin/* or krbtgt/* principals. The **lockdown_keys** principal attribute can be used to prevent key extraction from specific principals regardless of the granted privilege.

target_principal
(Optional. Partially or fully qualified Kerberos principal name.) Specifies the principal on which *permissions* may be applied. Each component of the name may be wildcarded using the * character.

target_principal can also include back-references to *principal*, in which `*number` matches the corresponding wildcard in *principal*.

restrictions
(Optional) A string of flags. Allowed restrictions are:

{+|-}*flagname*
flag is forced to the indicated value. The permissible flags are the same as those for the **default_principal_flags** variable in *kdc.conf*.

-clearpolicy
policy is forced to be empty.

-policy pol
policy is forced to be *pol*.

-{expire, pwexpire, maxlife, maxrenewlife} time
(getdate string) associated value will be forced to MIN(*time*, requested value).

The above flags act as restrictions on any add or modify operation which is allowed due to that ACL line.

Warning: If the kadmind ACL file is modified, the kadmind daemon needs to be restarted for changes to take effect.

EXAMPLE

Here is an example of a kadm5.acl file:

```
*/admin@ATHENA.MIT.EDU    *                              # line 1
joeadmin@ATHENA.MIT.EDU   ADMCIL                         # line 2
joeadmin/*@ATHENA.MIT.EDU i   */root@ATHENA.MIT.EDU      # line 3
*/root@ATHENA.MIT.EDU     ci  *1@ATHENA.MIT.EDU          # line 4
*/root@ATHENA.MIT.EDU     l   *                          # line 5
sms@ATHENA.MIT.EDU        x   * -maxlife 9h -postdateable # line 6
```

(line 1) Any principal in the `ATHENA.MIT.EDU` realm with an `admin` instance has all administrative privileges except extracting keys.

(lines 1-3) The user `joeadmin` has all permissions except extracting keys with his `admin` instance, `joeadmin/admin@ATHENA.MIT.EDU` (matches line 1). He has no permissions at all with his null instance, `joeadmin@ATHENA.MIT.EDU` (matches line 2). His `root` and other non-`admin`, non-null instances (e.g., `extra` or `dbadmin`) have inquire permissions with any principal that has the instance `root` (matches line 3).

(line 4) Any `root` principal in `ATHENA.MIT.EDU` can inquire or change the password of their null instance, but not any other null instance. (Here, `*1` denotes a back-reference to the component matching the first wildcard in the actor principal.)

(line 5) Any `root` principal in `ATHENA.MIT.EDU` can generate the list of principals in the database, and the list of policies in the database. This line is separate from line 4, because list permission can only be granted globally, not to specific target principals.

(line 6) Finally, the Service Management System principal `sms@ATHENA.MIT.EDU` has all permissions except extracting keys, but any principal that it creates or modifies will not be able to get postdateable tickets or tickets with a life of longer than 9 hours.

MODULE BEHAVIOR

The ACL file can coexist with other authorization modules in release 1.16 and later, as configured in the *kadm5_auth interface* section of *krb5.conf*. The ACL file will positively authorize operations according to the rules above, but will never authoritatively deny an operation, so other modules can authorize operations in addition to those authorized by the ACL file.

To operate without an ACL file, set the *acl_file* variable in *kdc.conf* to the empty string with `acl_file = ""`.

SEE ALSO

kdc.conf, *kadmind*

CHAPTER

THREE

REALM CONFIGURATION DECISIONS

Before installing Kerberos V5, it is necessary to consider the following issues:

- The name of your Kerberos realm (or the name of each realm, if you need more than one).
- How you will assign your hostnames to Kerberos realms.
- Which ports your KDC and and kadmind services will use, if they will not be using the default ports.
- How many replica KDCs you need and where they should be located.
- The hostnames of your primary and replica KDCs.
- How frequently you will propagate the database from the primary KDC to the replica KDCs.

3.1 Realm name

Although your Kerberos realm can be any ASCII string, convention is to make it the same as your domain name, in upper-case letters.

For example, hosts in the domain `example.com` would be in the Kerberos realm:

```
EXAMPLE.COM
```

If you need multiple Kerberos realms, MIT recommends that you use descriptive names which end with your domain name, such as:

```
BOSTON.EXAMPLE.COM
HOUSTON.EXAMPLE.COM
```

3.2 Mapping hostnames onto Kerberos realms

Mapping hostnames onto Kerberos realms is done in one of three ways.

The first mechanism works through a set of rules in the *[domain_realm]* section of *krb5.conf*. You can specify mappings for an entire domain or on a per-hostname basis. Typically you would do this by specifying the mappings for a given domain or subdomain and listing the exceptions.

The second mechanism is to use KDC host-based service referrals. With this method, the KDC's krb5.conf has a full [domain_realm] mapping for hosts, but the clients do not, or have mappings for only a subset of the hosts they might contact. When a client needs to contact a server host for which it has no mapping, it will ask the client realm's KDC for the service ticket, and will receive a referral to the appropriate service realm.

To use referrals, clients must be running MIT krb5 1.6 or later, and the KDC must be running MIT krb5 1.7 or later. The **host_based_services** and **no_host_referral** variables in the *[realms]* section of *kdc.conf* can be used to fine-tune referral behavior on the KDC.

It is also possible for clients to use DNS TXT records, if **dns_lookup_realm** is enabled in *krb5.conf*. Such lookups are disabled by default because DNS is an insecure protocol and security holes could result if DNS records are spoofed. If enabled, the client will try to look up a TXT record formed by prepending the prefix `_kerberos` to the hostname in question. If that record is not found, the client will attempt a lookup by prepending `_kerberos` to the host's domain name, then its parent domain, up to the top-level domain. For the hostname `boston.engineering.example.com`, the names looked up would be:

```
_kerberos.boston.engineering.example.com
_kerberos.engineering.example.com
_kerberos.example.com
_kerberos.com
```

The value of the first TXT record found is taken as the realm name.

Even if you do not choose to use this mechanism within your site, you may wish to set it up anyway, for use when interacting with other sites.

3.3 Ports for the KDC and admin services

The default ports used by Kerberos are port 88 for the KDC and port 749 for the admin server. You can, however, choose to run on other ports, as long as they are specified in each host's *krb5.conf* files or in DNS SRV records, and the *kdc.conf* file on each KDC. For a more thorough treatment of port numbers used by the Kerberos V5 programs, refer to the *Configuring your firewall to work with Kerberos V5*.

3.4 Replica KDCs

Replica KDCs provide an additional source of Kerberos ticket-granting services in the event of inaccessibility of the primary KDC. The number of replica KDCs you need and the decision of where to place them, both physically and logically, depends on the specifics of your network.

Kerberos authentication requires that each client be able to contact a KDC. Therefore, you need to anticipate any likely reason a KDC might be unavailable and have a replica KDC to take up the slack.

Some considerations include:

- Have at least one replica KDC as a backup, for when the primary KDC is down, is being upgraded, or is otherwise unavailable.
- If your network is split such that a network outage is likely to cause a network partition (some segment or segments of the network to become cut off or isolated from other segments), have a replica KDC accessible to each segment.
- If possible, have at least one replica KDC in a different building from the primary, in case of power outages, fires, or other localized disasters.

3.5 Hostnames for KDCs

MIT recommends that your KDCs have a predefined set of CNAME records (DNS hostname aliases), such as `kerberos` for the primary KDC and `kerberos-1`, `kerberos-2`, ... for the replica KDCs. This way, if you need to swap a machine, you only need to change a DNS entry, rather than having to change hostnames.

As of MIT krb5 1.4, clients can locate a realm's KDCs through DNS using SRV records (**RFC 2782**), assuming the Kerberos realm name is also a DNS domain name. These records indicate the hostname and port number to contact for that service, optionally with weighting and prioritization. The domain name used in the SRV record name is the realm name. Several different Kerberos-related service names are used:

_kerberos._udp
: This is for contacting any KDC by UDP. This entry will be used the most often. Normally you should list port 88 on each of your KDCs.

_kerberos._tcp
: This is for contacting any KDC by TCP. Normally you should use port 88. This entry should be omitted if the KDC does not listen on TCP ports, as was the default prior to release 1.13.

_kerberos-master._udp
: This entry should refer to those KDCs, if any, that will immediately see password changes to the Kerberos database. If a user is logging in and the password appears to be incorrect, the client will retry with the primary KDC before failing with an "incorrect password" error given.

  If you have only one KDC, or for whatever reason there is no accessible KDC that would get database changes faster than the others, you do not need to define this entry.

_kerberos-adm._tcp
: This should list port 749 on your primary KDC. Support for it is not complete at this time, but it will eventually be used by the *kadmin* program and related utilities. For now, you will also need the **admin_server** variable in *krb5.conf*.

_kerberos-master._tcp
: The corresponding TCP port for _kerberos-master._udp, assuming the primary KDC listens on a TCP port.

_kpasswd._udp
: This entry should list port 464 on your primary KDC. It is used when a user changes her password. If this entry is not defined but a _kerberos-adm._tcp entry is defined, the client will use the _kerberos-adm._tcp entry with the port number changed to 464.

_kpasswd._tcp
: The corresponding TCP port for _kpasswd._udp.

The DNS SRV specification requires that the hostnames listed be the canonical names, not aliases. So, for example, you might include the following records in your (BIND-style) zone file:

```
$ORIGIN foobar.com.
_kerberos               TXT     "FOOBAR.COM"
kerberos                CNAME   daisy
kerberos-1              CNAME   use-the-force-luke
kerberos-2              CNAME   bunny-rabbit
_kerberos._udp          SRV     0 0 88 daisy
                        SRV     0 0 88 use-the-force-luke
                        SRV     0 0 88 bunny-rabbit
_kerberos-master._udp   SRV     0 0 88 daisy
_kerberos-adm._tcp      SRV     0 0 749 daisy
_kpasswd._udp           SRV     0 0 464 daisy
```

Clients can also be configured with the explicit location of services using the **kdc**, **master_kdc**, **admin_server**, and **kpasswd_server** variables in the *[realms]* section of *krb5.conf*. Even if some clients will be configured with explicit server locations, providing SRV records will still benefit unconfigured clients, and be useful for other sites.

Clients can be configured with the **sitename** realm variable (new in release 1.22). If a site name is set, the client first attempts SRV record lookups with ".*sitename*._sites" inserted after the service and protocol name and before the Kerberos realm. Site-specific records may indicate servers more proximal to the client, allowing for faster access.

3.6 KDC Discovery

As of MIT krb5 1.15, clients can also locate KDCs in DNS through URI records (**RFC 7553**). Limitations with the SRV record format may result in extra DNS queries in situations where a client must failover to other transport types, or find a primary server. The URI record can convey more information about a realm's KDCs with a single query.

The client performs a query for the following URI records:

- `_kerberos.REALM` for finding KDCs.
- `_kerberos-adm.REALM` for finding kadmin services.
- `_kpasswd.REALM` for finding password services.

The URI record includes a priority, weight, and a URI string that consists of case-insensitive colon separated fields, in the form `scheme:[flags]:transport:residual`.

- *scheme* defines the registered URI type. It should always be `krb5srv`.
- *flags* contains zero or more flag characters. Currently the only valid flag is `m`, which indicates that the record is for a primary server.
- *transport* defines the transport type of the residual URL or address. Accepted values are `tcp`, `udp`, or `kkdcp` for the MS-KKDCP type.
- *residual* contains the hostname, IP address, or URL to be contacted using the specified transport, with an optional port extension. The MS-KKDCP transport type uses a HTTPS URL, and can include a port and/or path extension.

An example of URI records in a zone file:

```
_kerberos.EXAMPLE.COM  URI  10 1 krb5srv:m:tcp:kdc1.example.com
                       URI  20 1 krb5srv:m:udp:kdc2.example.com:89
                       URI  40 1 krb5srv::udp:10.10.0.23
                       URI  30 1 krb5srv::kkdcp:https://proxy:89/auth
```

URI lookups are enabled by default, and can be disabled by setting **dns_uri_lookup** in the *[libdefaults]* section of *krb5.conf* to False. When enabled, URI lookups take precedence over SRV lookups, falling back to SRV lookups if no URI records are found.

The **sitename** variable in the *[realms]* section of *krb5.conf* applies to URI lookups as well as SRV lookups.

3.7 Database propagation

The Kerberos database resides on the primary KDC, and must be propagated regularly (usually by a cron job) to the replica KDCs. In deciding how frequently the propagation should happen, you will need to balance the amount of time the propagation takes against the maximum reasonable amount of time a user should have to wait for a password change to take effect.

If the propagation time is longer than this maximum reasonable time (e.g., you have a particularly large database, you have a lot of replicas, or you experience frequent network delays), you may wish to cut down on your propagation delay by performing the propagation in parallel. To do this, have the primary KDC propagate the database to one set of replicas, and then have each of these replicas propagate the database to additional replicas.

See also *Incremental database propagation*

CHAPTER

FOUR

DATABASE ADMINISTRATION

A Kerberos database contains all of a realm's Kerberos principals, their passwords, and other administrative information about each principal. For the most part, you will use the *kdb5_util* program to manipulate the Kerberos database as a whole, and the *kadmin* program to make changes to the entries in the database. (One notable exception is that users will use the kpasswd(1) program to change their own passwords.) The kadmin program has its own command-line interface, to which you type the database administrating commands.

kdb5_util provides a means to create, delete, load, or dump a Kerberos database. It also contains commands to roll over the database master key, and to stash a copy of the key so that the *kadmind* and *krb5kdc* daemons can use the database without manual input.

kadmin provides for the maintenance of Kerberos principals, password policies, and service key tables (keytabs). Normally it operates as a network client using Kerberos authentication to communicate with *kadmind*, but there is also a variant, named kadmin.local, which directly accesses the Kerberos database on the local filesystem (or through LDAP). kadmin.local is necessary to set up enough of the database to be able to use the remote version.

kadmin can authenticate to the admin server using the service principal `kadmin/admin` or `kadmin/HOST` (where *HOST* is the hostname of the admin server). If the credentials cache contains a ticket for either service principal and the **-c** ccache option is specified, that ticket is used to authenticate to KADM5. Otherwise, the **-p** and **-k** options are used to specify the client Kerberos principal name used to authenticate. Once kadmin has determined the principal name, it requests a `kadmin/admin` Kerberos service ticket from the KDC, and uses that service ticket to authenticate to KADM5.

See *kadmin* for the available kadmin and kadmin.local commands and options.

4.1 Principals

Each entry in the Kerberos database contains a Kerberos principal and the attributes and policies associated with that principal.

To add a principal to the database, use the *kadmin* **add_principal** command. User principals should usually be created with the `+requires_preauth -allow_svr` options to help mitigate dictionary attacks (see *Addressing dictionary attack risks*):

```
kadmin: addprinc +requires_preauth -allow_svr alice
Enter password for principal "alice@KRBTEST.COM":
Re-enter password for principal "alice@KRBTEST.COM":
```

User principals which will authenticate with *PKINIT configuration* should instead by created with the `-nokey` option:

```
kadmin: addprinc -nokey alice
```

Service principals can be created with the `-nokey` option; long-term keys will be added when a keytab is generated:

```
kadmin: addprinc -nokey host/foo.mit.edu
kadmin: ktadd -k foo.keytab host/foo.mit.edu
Entry for principal host/foo.mit.edu with kvno 1, encryption type aes256-cts-hmac-sha1-
↪96 added to keytab WRFILE:foo.keytab.
Entry for principal host/foo.mit.edu with kvno 1, encryption type aes128-cts-hmac-sha1-
↪96 added to keytab WRFILE:foo.keytab.
```

To modify attributes of an existing principal, use the kadmin **modify_principal** command:

```
kadmin: modprinc -expire tomorrow alice
Principal "alice@KRBTEST.COM" modified.
```

To delete a principal, use the kadmin **delete_principal** command:

```
kadmin: delprinc alice
Are you sure you want to delete the principal "alice@KRBTEST.COM"? (yes/no): yes
Principal "alice@KRBTEST.COM" deleted.
Make sure that you have removed this principal from all ACLs before reusing.
```

To change a principal's password, use the kadmin **change_password** command. Password changes made through kadmin are subject to the same password policies as would apply to password changes made through kpasswd(1).

To view the attributes of a principal, use the kadmin` **get_principal** command.

To generate a listing of principals, use the kadmin **list_principals** command.

To give a principal additional names, use the kadmin **add_alias** command to create aliases to the principal (new in release 1.22). Aliases can be removed with the **delete_principal** command.

4.2 Policies

A policy is a set of rules governing passwords. Policies can dictate minimum and maximum password lifetimes, minimum number of characters and character classes a password must contain, and the number of old passwords kept in the database.

To add a new policy, use the *kadmin* **add_policy** command:

```
kadmin: addpol -maxlife "1 year" -history 3 stduser
```

To modify attributes of a principal, use the kadmin **modify_policy** command. To delete a policy, use the kadmin **delete_policy** command.

To associate a policy with a principal, use the kadmin **modify_principal** command with the **-policy** option:

kadmin: modprinc -policy stduser alice Principal "alice@KRBTEST.COM" modified.

A principal entry may be associated with a nonexistent policy, either because the policy did not exist at the time of associated or was deleted afterwards. kadmin will warn when associated a principal with a nonexistent policy, and will annotate the policy name with "[does not exist]" in the **get_principal** output.

4.2.1 Updating the history key

If a policy specifies a number of old keys kept of two or more, the stored old keys are encrypted in a history key, which is found in the key data of the `kadmin/history` principal.

Currently there is no support for proper rollover of the history key, but you can change the history key (for example, to use a better encryption type) at the cost of invalidating currently stored old keys. To change the history key, run:

```
kadmin: change_password -randkey kadmin/history
```

This command will fail if you specify the **-keepold** flag. Only one new history key will be created, even if you specify multiple key/salt combinations.

In the future, we plan to migrate towards encrypting old keys in the master key instead of the history key, and implementing proper rollover support for stored old keys.

4.3 Privileges

Administrative privileges for the Kerberos database are stored in the file *kadm5.acl*.

Note: A common use of an admin instance is so you can grant separate permissions (such as administrator access to the Kerberos database) to a separate Kerberos principal. For example, the user `joeadmin` might have a principal for his administrative use, called `joeadmin/admin`. This way, `joeadmin` would obtain `joeadmin/admin` tickets only when he actually needs to use those permissions.

4.4 Operations on the Kerberos database

The *kdb5_util* command is the primary tool for administrating the Kerberos database when using the DB2 or LMDB modules (see *Database types*). Creating a database is described in *Create the KDC database*.

To create a stash file using the master password (because the database was not created with one using the `create -s` flag, or after restoring from a backup which did not contain the stash file), use the kdb5_util **stash** command:

```
$ kdb5_util stash
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key
Enter KDC database master key:  <= Type the KDC database master password.
```

To destroy a database, use the kdb5_util destroy command:

```
$ kdb5_util destroy
Deleting KDC database stored in '/var/krb5kdc/principal', are you sure?
(type 'yes' to confirm)? yes
OK, deleting database '/var/krb5kdc/principal'...
** Database '/var/krb5kdc/principal' destroyed.
```

4.4.1 Dumping and loading a Kerberos database

To dump a Kerberos database into a text file for backup or transfer purposes, use the *kdb5_util* **dump** command on one of the KDCs:

```
$ kdb5_util dump dumpfile

$ kbd5_util dump -verbose dumpfile
kadmin/admin@ATHENA.MIT.EDU
krbtgt/ATHENA.MIT.EDU@ATHENA.MIT.EDU
kadmin/history@ATHENA.MIT.EDU
K/M@ATHENA.MIT.EDU
kadmin/changepw@ATHENA.MIT.EDU
```

You may specify which principals to dump, using full principal names including realm:

```
$ kdb5_util dump -verbose someprincs K/M@ATHENA.MIT.EDU kadmin/admin@ATHENA.MIT.EDU
kadmin/admin@ATHENA.MIT.EDU
K/M@ATHENA.MIT.EDU
```

To restore a Kerberos database dump from a file, use the *kdb5_util* **load** command:

```
$ kdb5_util load dumpfile
```

To update an existing database with a partial dump file containing only some principals, use the `-update` flag:

```
$ kdb5_util load -update someprincs
```

Note: If the database file exists, and the *-update* flag was not given, *kdb5_util* will overwrite the existing database.

4.4.2 Updating the master key

Starting with release 1.7, *kdb5_util* allows the master key to be changed using a rollover process, with minimal loss of availability. To roll over the master key, follow these steps:

1. On the primary KDC, run `kdb5_util list_mkeys` to view the current master key version number (KVNO). If you have never rolled over the master key before, this will likely be version 1:

   ```
   $ kdb5_util list_mkeys
   Master keys for Principal: K/M@KRBTEST.COM
   KVNO: 1, Enctype: aes256-cts-hmac-sha384-192, Active on: Thu Jan 01 00:00:00 UTC␣
   ↪1970 *
   ```

2. On the primary KDC, run `kdb5_util use_mkey 1` to ensure that a master key activation list is present in the database. This step is unnecessary in release 1.11.4 or later, or if the database was initially created with release 1.7 or later.

3. On the primary KDC, run `kdb5_util add_mkey -s` to create a new master key and write it to the stash file. Enter a secure password when prompted. If this is the first time you are changing the master key, the new key will have version 2. The new master key will not be used until you make it active.

4. Propagate the database to all replica KDCs, either manually or by waiting until the next scheduled propagation. If you do not have any replica KDCs, you can skip this and the next step.

5. On each replica KDC, run `kdb5_util list_mkeys` to verify that the new master key is present, and then `kdb5_util stash` to write the new master key to the replica KDC's stash file.
6. On the primary KDC, run `kdb5_util use_mkey 2` to begin using the new master key. Replace `2` with the version of the new master key, as appropriate. You can optionally specify a date for the new master key to become active; by default, it will become active immediately. Prior to release 1.12, *kadmind* must be restarted for this change to take full effect.
7. On the primary KDC, run `kdb5_util update_princ_encryption`. This command will iterate over the database and re-encrypt all keys in the new master key. If the database is large and uses DB2, the primary KDC will become unavailable while this command runs, but clients should fail over to replica KDCs (if any are present) during this time period. In release 1.13 and later, you can instead run `kdb5_util -x unlockiter update_princ_encryption` to use unlocked iteration; this variant will take longer, but will keep the database available to the KDC and kadmind while it runs.
8. Wait until the above changes have propagated to all replica KDCs and until all running KDC and kadmind processes have serviced requests using updated principal entries.
9. On the primary KDC, run `kdb5_util purge_mkeys` to clean up the old master key.

4.5 Operations on the LDAP database

The *kdb5_ldap_util* command is the primary tool for administrating the Kerberos database when using the LDAP module. Creating an LDAP Kerberos database is describe in *Configuring Kerberos with OpenLDAP back-end*.

To view a list of realms in the LDAP database, use the kdb5_ldap_util **list** command:

```
$ kdb5_ldap_util list
KRBTEST.COM
```

To modify the attributes of a realm, use the kdb5_ldap_util **modify** command. For example, to change the default realm's maximum ticket life:

```
$ kdb5_ldap_util modify -maxtktlife "10 hours"
```

To display the attributes of a realm, use the kdb5_ldap_util **view** command:

```
$ kdb5_ldap_util view
                Realm Name: KRBTEST.COM
     Maximum Ticket Life: 0 days 00:10:00
```

To remove a realm from the LDAP database, destroying its contents, use the kdb5_ldap_util **destroy** command:

```
$ kdb5_ldap_util destroy
Deleting KDC database of 'KRBTEST.COM', are you sure?
(type 'yes' to confirm)? yes
OK, deleting database of 'KRBTEST.COM'...
** Database of 'KRBTEST.COM' destroyed.
```

4.5.1 Ticket Policy operations

Unlike the DB2 and LMDB modules, the LDAP module supports ticket policy objects, which can be associated with principals to restrict maximum ticket lifetimes and set mandatory principal flags. Ticket policy objects are distinct from the password policies described earlier on this page, and are chiefly managed through kdb5_ldap_util rather than kadmin. To create a new ticket policy, use the kdb5_ldap_util **create_policy** command:

```
$ kdb5_ldap_util create_policy -maxrenewlife "2 days" users
```

To associate a ticket policy with a principal, use the *kadmin* **modify_principal** (or **add_principal**) command with the **-x tktpolicy=***policy* option:

```
$ kadmin.local modprinc -x tktpolicy=users alice
```

To remove a ticket policy reference from a principal, use the same command with an empty *policy*:

```
$ kadmin.local modprinc -x tktpolicy= alice
```

To list the existing ticket policy objects, use the kdb5_ldap_util **list_policy** command:

```
$ kdb5_ldap_util list_policy
users
```

To modify the attributes of a ticket policy object, use the kdb5_ldap_util **modify_policy** command:

```
$ kdb5_ldap_util modify_policy -allow_svr +requires_preauth users
```

To view the attributes of a ticket policy object, use the kdb5_ldap_util **view_policy** command:

```
$ kdb5_ldap_util view_policy users
            Ticket policy: users
   Maximum renewable life: 2 days 00:00:00
             Ticket flags: REQUIRES_PRE_AUTH DISALLOW_SVR
```

To destroy an ticket policy object, use the kdb5_ldap_util **destroy_policy** command:

```
$ kdb5_ldap_util destroy_policy users
This will delete the policy object 'users', are you sure?
(type 'yes' to confirm)? yes
** policy object 'users' deleted.
```

4.6 Cross-realm authentication

In order for a KDC in one realm to authenticate Kerberos users in a different realm, it must share a key with the KDC in the other realm. In both databases, there must be krbtgt service principals for both realms. For example, if you need to do cross-realm authentication between the realms `ATHENA.MIT.EDU` and `EXAMPLE.COM`, you would need to add the principals `krbtgt/EXAMPLE.COM@ATHENA.MIT.EDU` and `krbtgt/ATHENA.MIT.EDU@EXAMPLE.COM` to both databases. These principals must all have the same passwords, key version numbers, and encryption types; this may require explicitly setting the key version number with the **-kvno** option.

In the ATHENA.MIT.EDU and EXAMPLE.COM cross-realm case, the administrators would run the following commands on the KDCs in both realms:

```
shell%: kadmin.local -e "aes256-cts:normal"
kadmin: addprinc -requires_preauth krbtgt/ATHENA.MIT.EDU@EXAMPLE.COM
Enter password for principal krbtgt/ATHENA.MIT.EDU@EXAMPLE.COM:
Re-enter password for principal krbtgt/ATHENA.MIT.EDU@EXAMPLE.COM:
kadmin: addprinc -requires_preauth krbtgt/EXAMPLE.COM@ATHENA.MIT.EDU
Enter password for principal krbtgt/EXAMPLE.COM@ATHENA.MIT.EDU:
Enter password for principal krbtgt/EXAMPLE.COM@ATHENA.MIT.EDU:
kadmin:
```

Note: Even if most principals in a realm are generally created with the **requires_preauth** flag enabled, this flag is not desirable on cross-realm authentication keys because doing so makes it impossible to disable preauthentication on a service-by-service basis. Disabling it as in the example above is recommended.

Note: It is very important that these principals have good passwords. MIT recommends that TGT principal passwords be at least 26 characters of random ASCII text.

4.7 Changing the krbtgt key

A Kerberos Ticket Granting Ticket (TGT) is a service ticket for the principal `krbtgt/REALM`. The key for this principal is created when the Kerberos database is initialized and need not be changed. However, it will only have the encryption types supported by the KDC at the time of the initial database creation. To allow use of newer encryption types for the TGT, this key has to be changed.

Changing this key using the normal *kadmin* **change_password** command would invalidate any previously issued TGTs. Therefore, when changing this key, normally one should use the **-keepold** flag to change_password to retain the previous key in the database as well as the new key. For example:

```
kadmin: change_password -randkey -keepold krbtgt/ATHENA.MIT.EDU@ATHENA.MIT.EDU
```

Warning: After issuing this command, the old key is still valid and is still vulnerable to (for instance) brute force attacks. To completely retire an old key or encryption type, run the kadmin **purgekeys** command to delete keys with older kvnos, ideally first making sure that all tickets issued with the old keys have expired.

Only the first krbtgt key of the newest key version is used to encrypt ticket-granting tickets. However, the set of encryption types present in the krbtgt keys is used by default to determine the session key types supported by the krbtgt service (see *Session key selection*). Because non-MIT Kerberos clients sometimes send a limited set of encryption types when making AS requests, it can be important for the krbtgt service to support multiple encryption types. This can be accomplished by giving the krbtgt principal multiple keys, which is usually as simple as not specifying any **-e** option when changing the krbtgt key, or by setting the **session_enctypes** string attribute on the krbtgt principal (see *set_string*).

Due to a bug in releases 1.8 through 1.13, renewed and forwarded tickets may not work if the original ticket was obtained prior to a krbtgt key change and the modified ticket is obtained afterwards. Upgrading the KDC to release 1.14 or later will correct this bug.

4.8 Incremental database propagation

4.8.1 Overview

At some very large sites, dumping and transmitting the database can take more time than is desirable for changes to propagate from the primary KDC to the replica KDCs. The incremental propagation support added in the 1.7 release is intended to address this.

With incremental propagation enabled, all programs on the primary KDC that change the database also write information about the changes to an "update log" file, maintained as a circular buffer of a certain size. A process on each replica KDC connects to a service on the primary KDC (currently implemented in the *kadmind* server) and periodically requests the changes that have been made since the last check. By default, this check is done every two minutes.

Incremental propagation uses the following entries in the per-realm data in the KDC config file (See *kdc.conf*):

iprop_	*boole*	If *true*, then incremental propagation is enabled, and (as noted below) normal kprop propagation is disabled. The default is *false*.
iprop_	*integer*	Indicates the number of entries that should be retained in the update log. The default is 1000; the maximum number is 2500.
iprop_	*time interval*	Indicates how often the replica should poll the primary KDC for changes to the database. The default is two minutes.
iprop_	*integer*	Specifies the port number to be used for incremental propagation. This is required in both primary and replica configuration files.
iprop_	*integer*	Specifies the number of seconds to wait for a full propagation to complete. This is optional on replica configurations. Defaults to 300 seconds (5 minutes).
iprop_	*filename*	Specifies where the update log file for the realm database is to be stored. The default is to use the *database_name* entry from the realms section of the config file *kdc.conf*, with *.ulog* appended. (NOTE: If database_name isn't specified in the realms section, perhaps because the LDAP database back end is being used, or the file name is specified in the *dbmodules* section, then the hard-coded default for *database_name* is used. Determination of the *iprop_logfile* default value will not use values from the *dbmodules* section.)

Both primary and replica sides must have a principal named `kiprop/hostname` (where *hostname* is the lowercase, fully-qualified, canonical name for the host) registered in the Kerberos database, and have keys for that principal stored in the default keytab file (*DEFKTNAME*). The `kiprop/hostname` principal may have been created automatically for the primary KDC, but it must always be created for replica KDCs.

On the primary KDC side, the `kiprop/hostname` principal must be listed in the kadmind ACL file *kadm5.acl*, and given the **p** privilege (see *Privileges*).

On the replica KDC side, *kpropd* should be run. When incremental propagation is enabled, it will connect to the kadmind on the primary KDC and start requesting updates.

The normal kprop mechanism is disabled by the incremental propagation support. However, if the replica has been unable to fetch changes from the primary KDC for too long (network problems, perhaps), the log on the primary may wrap around and overwrite some of the updates that the replica has not yet retrieved. In this case, the replica will instruct the primary KDC to dump the current database out to a file and invoke a one-time kprop propagation, with special options to also convey the point in the update log at which the replica should resume fetching incremental updates. Thus, all the keytab and ACL setup previously described for kprop propagation is still needed.

If an environment has a large number of replicas, it may be desirable to arrange them in a hierarchy instead of having the primary serve updates to every replica. To do this, run `kadmind -proponly` on each intermediate replica, and `kpropd -A upstreamhostname` on downstream replicas to direct each one to the appropriate upstream replica.

There are several known restrictions in the current implementation:

- The incremental update protocol does not transport changes to policy objects. Any policy changes on the primary will result in full resyncs to all replicas.
- The replica's KDB module must support locking; it cannot be using the LDAP KDB module.
- The primary and replica must be able to initiate TCP connections in both directions, without an intervening NAT.

4.8.2 Sun/MIT incremental propagation differences

Sun donated the original code for supporting incremental database propagation to MIT. Some changes have been made in the MIT source tree that will be visible to administrators. (These notes are based on Sun's patches. Changes to Sun's implementation since then may not be reflected here.)

The Sun config file support looks for `sunw_dbprop_enable`, `sunw_dbprop_master_ulogsize`, and `sunw_dbprop_slave_poll`.

The incremental propagation service is implemented as an ONC RPC service. In the Sun implementation, the service is registered with rpcbind (also known as portmapper) and the client looks up the port number to contact. In the MIT implementation, where interaction with some modern versions of rpcbind doesn't always work well, the port number must be specified in the config file on both the primary and replica sides.

The Sun implementation hard-codes pathnames in `/var/krb5` for the update log and the per-replica kprop dump files. In the MIT implementation, the pathname for the update log is specified in the config file, and the per-replica dump files are stored in *LOCALSTATEDIR*`/krb5kdc/replica_datatrans_hostname`.

CHAPTER

FIVE

DATABASE TYPES

A Kerberos database can be implemented with one of three built-in database providers, called KDB modules. Software which incorporates the MIT krb5 KDC may also provide its own KDB module. The following subsections describe the three built-in KDB modules and the configuration specific to them.

The database type can be configured with the **db_library** variable in the *[dbmodules]* subsection for the realm. For example:

```
[dbmodules]
    ATHENA.MIT.EDU = {
        db_library = db2
    }
```

If the `ATHENA.MIT.EDU` realm subsection contains a **database_module** setting, then the subsection within `[dbmodules]` should use that name instead of `ATHENA.MIT.EDU`.

To transition from one database type to another, stop the *kadmind* service, use `kdb5_util dump` to create a dump file, change the **db_library** value and set any appropriate configuration for the new database type, and use `kdb5_util load` to create and populate the new database. If the new database type is LDAP, create the new database using `kdb5_ldap_util` and populate it from the dump file using `kdb5_util load -update`. Then restart the *krb5kdc* and *kadmind* services.

5.1 Berkeley database module (db2)

The default KDB module is `db2`, which uses a version of the Berkeley DB library. It creates four files based on the database pathname. If the pathname ends with `principal` then the four files are:

- `principal`, containing principal entry data
- `principal.ok`, a lock file for the principal database
- `principal.kadm5`, containing policy object data
- `principal.kadm5.lock`, a lock file for the policy database

For large databases, the *kdb5_util* **dump** command (perhaps invoked by *kprop* or by *kadmind* for incremental propagation) may cause *krb5kdc* to stop for a noticeable period of time while it iterates over the database. This delay can be avoided by disabling account lockout features so that the KDC does not perform database writes (see *KDC performance and account lockout*). Alternatively, a slower form of iteration can be enabled by setting the **unlockiter** variable to `true`. For example:

```
[dbmodules]
    ATHENA.MIT.EDU = {
```

(continues on next page)

(continued from previous page)

```
        db_library = db2
        unlockiter = true
    }
```

In rare cases, a power failure or other unclean system shutdown may cause inconsistencies in the internal pointers within a database file, such that `kdb5_util dump` cannot retrieve all principal entries in the database. In this situation, it may be possible to retrieve all of the principal data by running `kdb5_util dump -recurse` to iterate over the database using the tree pointers instead of the iteration pointers. Running `kdb5_util dump -rev` to iterate over the database backwards may also retrieve some of the data which is not retrieved by a normal dump operation.

5.2 Lightning Memory-Mapped Database module (klmdb)

The klmdb module was added in release 1.17. It uses the LMDB library, and may offer better performance and reliability than the db2 module. It creates four files based on the database pathname. If the pathname ends with `principal`, then the four files are:

- `principal.mdb`, containing policy object data and most principal entry data
- `principal.mdb-lock`, a lock file for the primary database
- `principal.lockout.mdb`, containing the account lockout attributes (last successful authentication time, last failed authentication time, and number of failed attempts) for each principal entry
- `principal.lockout.mdb-lock`, a lock file for the lockout database

Separating out the lockout attributes ensures that the KDC will never block on an administrative operation such as a database dump or load. It also allows the KDC to operate without write access to the primary database. If both account lockout features are disabled (see *KDC performance and account lockout*), the lockout database files will be created but will not subsequently be opened, and the account lockout attributes will always have zero values.

Because LMDB creates a memory map to the database files, it requires a configured memory map size which also determines the maximum size of the database. This size is applied equally to the two databases, so twice the configured size will be consumed in the process address space; this is primarily a limitation on 32-bit platforms. The default value of 128 megabytes should be sufficient for several hundred thousand principal entries. If the limit is reached, kadmin operations will fail and the error message "Environment mapsize limit reached" will appear in the kadmind log file. In this case, the **mapsize** variable can be used to increase the map size. The following example sets the map size to 512 megabytes:

```
[dbmodules]
    ATHENA.MIT.EDU = {
        db_library = klmdb
        mapsize = 512
    }
```

LMDB has a configurable maximum number of readers. The default value of 128 should be sufficient for most deployments. If you are going to use a large number of KDC worker processes, it may be necessary to set the **max_readers** variable to a larger number.

By default, LMDB synchronizes database files to disk after each write transaction to ensure durability in the case of an unclean system shutdown. The klmdb module always turns synchronization off for the lockout database to ensure reasonable KDC performance, but leaves it on for the primary database. If high throughput for administrative operations (including password changes) is required, the **nosync** variable can be set to "true" to disable synchronization for the primary database.

The klmdb module does not support explicit locking with the *kadmin* **lock** command.

5.3 LDAP module (kldap)

The kldap module stores principal and policy data using an LDAP server. To use it you must configure an LDAP server to use the Kerberos schema. See *Configuring Kerberos with OpenLDAP back-end* for details.

Because *krb5kdc* is single-threaded, latency in LDAP database accesses may limit KDC operation throughput. If the LDAP server is located on the same server host as the KDC and accessed through an `ldapi://` URL, latency should be minimal. If this is not possible, consider starting multiple KDC worker processes with the *krb5kdc* **-w** option to enable concurrent processing of KDC requests.

The kldap module does not support explicit locking with the *kadmin* **lock** command.

CHAPTER

SIX

ACCOUNT LOCKOUT

As of release 1.8, the KDC can be configured to lock out principals after a number of failed authentication attempts within a period of time. Account lockout can make it more difficult to attack a principal's password by brute force, but also makes it easy for an attacker to deny access to a principal.

6.1 Configuring account lockout

Account lockout only works for principals with the **+requires_preauth** flag set. Without this flag, the KDC cannot know whether or not a client successfully decrypted the ticket it issued. It is also important to set the **-allow_svr** flag on a principal to protect its password from an off-line dictionary attack through a TGS request. You can set these flags on a principal with *kadmin* as follows:

```
kadmin: modprinc +requires_preauth -allow_svr PRINCNAME
```

Account lockout parameters are configured via *policy objects*. There may be an existing policy associated with user principals (such as the "default" policy), or you may need to create a new one and associate it with each user principal.

The policy parameters related to account lockout are:

- *maxfailure*: the number of failed attempts before the principal is locked out
- *failurecountinterval*: the allowable interval between failed attempts
- *lockoutduration*: the amount of time a principal is locked out for

Here is an example of setting these parameters on a new policy and associating it with a principal:

```
kadmin: addpol -maxfailure 10 -failurecountinterval 180
    -lockoutduration 60 lockout_policy
kadmin: modprinc -policy lockout_policy PRINCNAME
```

6.2 Testing account lockout

To test that account lockout is working, try authenticating as the principal (hopefully not one that might be in use) multiple times with the wrong password. For instance, if **maxfailure** is set to 2, you might see:

```
$ kinit user
Password for user@KRBTEST.COM:
kinit: Password incorrect while getting initial credentials
$ kinit user
```

(continues on next page)

(continued from previous page)

```
Password for user@KRBTEST.COM:
kinit: Password incorrect while getting initial credentials
$ kinit user
kinit: Client's credentials have been revoked while getting initial credentials
```

6.3 Account lockout principal state

A principal entry keeps three pieces of state related to account lockout:

- The time of last successful authentication
- The time of last failed authentication
- A counter of failed attempts

The time of last successful authentication is not actually needed for the account lockout system to function, but may be of administrative interest. These fields can be observed with the **getprinc** kadmin command. For example:

```
kadmin: getprinc user
Principal: user@KRBTEST.COM
...
Last successful authentication: [never]
Last failed authentication: Mon Dec 03 12:30:33 EST 2012
Failed password attempts: 2
...
```

A principal which has been locked out can be administratively unlocked with the **-unlock** option to the **modprinc** kadmin command:

```
kadmin: modprinc -unlock PRINCNAME
```

This command will reset the number of failed attempts to 0.

6.4 KDC replication and account lockout

The account lockout state of a principal is not replicated by either traditional *kprop* or incremental propagation. Because of this, the number of attempts an attacker can make within a time period is multiplied by the number of KDCs. For instance, if the **maxfailure** parameter on a policy is 10 and there are four KDCs in the environment (a primary and three replicas), an attacker could make as many as 40 attempts before the principal is locked out on all four KDCs.

An administrative unlock is propagated from the primary to the replica KDCs during the next propagation. Propagation of an administrative unlock will cause the counter of failed attempts on each replica to reset to 1 on the next failure.

If a KDC environment uses a replication strategy other than kprop or incremental propagation, such as the LDAP KDB module with multi-master LDAP replication, then account lockout state may be replicated between KDCs and the concerns of this section may not apply.

6.5 KDC performance and account lockout

In order to fully track account lockout state, the KDC must write to the the database on each successful and failed authentication. Writing to the database is generally more expensive than reading from it, so these writes may have a significant impact on KDC performance. As of release 1.9, it is possible to turn off account lockout state tracking in order to improve performance, by setting the **disable_last_success** and **disable_lockout** variables in the database module subsection of *kdc.conf*. For example:

```
[dbmodules]
    DB = {
        disable_last_success = true
        disable_lockout = true
    }
```

Of the two variables, setting **disable_last_success** will usually have the largest positive impact on performance, and will still allow account lockout policies to operate. However, it will make it impossible to observe the last successful authentication time with kadmin.

6.6 KDC setup and account lockout

To update the account lockout state on principals, the KDC must be able to write to the principal database. For the DB2 module, no special setup is required. For the LDAP module, the KDC DN must be granted write access to the principal objects. If the KDC DN has only read access, account lockout will not function.

CHAPTER

SEVEN

CONFIGURING KERBEROS WITH OPENLDAP BACK-END

1. Make sure the LDAP server is using local authentication (`ldapi://`) or TLS (`ldaps`). See https://www.openldap.org/doc/admin/tls.html for instructions on configuring TLS support in OpenLDAP.

2. Add the Kerberos schema file to the LDAP Server using the OpenLDAP LDIF file from the krb5 source directory (`src/plugins/kdb/ldap/libkdb_ldap/kerberos.openldap.ldif`). The following example uses local authentication:

   ```
   ldapadd -Y EXTERNAL -H ldapi:/// -f /path/to/kerberos.openldap.ldif
   ```

3. Choose DNs for the *krb5kdc* and *kadmind* servers to bind to the LDAP server, and create them if necessary. Specify these DNs with the **ldap_kdc_dn** and **ldap_kadmind_dn** directives in *kdc.conf*. The kadmind DN will also be used for administrative commands such as *kdb5_util*.

   Alternatively, you may configure krb5kdc and kadmind to use SASL authentication to access the LDAP server; see the *[dbmodules]* relations **ldap_kdc_sasl_mech** and similar.

4. Specify a location for the LDAP service password file by setting **ldap_service_password_file**. Use `kdb5_ldap_util stashsrvpw` to stash passwords for the KDC and kadmind DNs chosen above. For example:

   ```
   kdb5_ldap_util stashsrvpw -f /path/to/service.keyfile cn=krbadmin,dc=example,dc=com
   ```

   Skip this step if you are using SASL authentication and the mechanism does not require a password.

5. Choose a DN for the global Kerberos container entry (but do not create the entry at this time). Specify this DN with the **ldap_kerberos_container_dn** directive in *kdc.conf*. Realm container entries will be created underneath this DN. Principal entries may exist either underneath the realm container (the default) or in separate trees referenced from the realm container.

6. Configure the LDAP server ACLs to enable the KDC and kadmin server DNs to read and write the Kerberos data. If **disable_last_success** and **disable_lockout** are both set to true in the *[dbmodules]* subsection for the realm, then the KDC DN only requires read access to the Kerberos data.

   Sample access control information:

   ```
   access to dn.base=""
       by * read

   access to dn.base="cn=Subschema"
       by * read

   # Provide access to the realm container.
   access to dn.subtree= "cn=EXAMPLE.COM,cn=krbcontainer,dc=example,dc=com"
       by dn.exact="cn=kdc-service,dc=example,dc=com" write
       by dn.exact="cn=adm-service,dc=example,dc=com" write
   ```

(continues on next page)

(continued from previous page)

```
    by * none

# Provide access to principals, if not underneath the realm container.
access to dn.subtree= "ou=users,dc=example,dc=com"
    by dn.exact="cn=kdc-service,dc=example,dc=com" write
    by dn.exact="cn=adm-service,dc=example,dc=com" write
    by * none

access to *
    by * read
```

If the locations of the container and principals or the DNs of the service objects for a realm are changed then this information should be updated.

7. In *kdc.conf*, make sure the following relations are set in the *[dbmodules]* subsection for the realm:

```
db_library (set to ``kldap``)
ldap_kerberos_container_dn
ldap_kdc_dn
ldap_kadmind_dn
ldap_service_password_file
ldap_servers
```

8. Create the realm using *kdb5_ldap_util*:

   kdb5_ldap_util create -subtrees ou=users,dc=example,dc=com -s

   Use the **-subtrees** option if the principals are to exist in a separate subtree from the realm container. Before executing the command, make sure that the subtree mentioned above (`ou=users,dc=example,dc=com`) exists. If the principals will exist underneath the realm container, omit the **-subtrees** option and do not worry about creating the principal subtree.

   For more information, refer to the section *Operations on the LDAP database*.

   The realm object is created under the **ldap_kerberos_container_dn** specified in the configuration file. This operation will also create the Kerberos container, if not present already. This container can be used to store information related to multiple realms.

9. Add an `eq` index for `krbPrincipalName` to speed up principal lookup operations. See https://www.openldap.org/doc/admin/tuning.html#Indexes for details.

With the LDAP back end it is possible to provide aliases for principal entries. Beginning in release 1.22, aliases can be added with the kadmin **add_alias** command, but it is also possible (in release 1.7 or later) to provide aliases through direct manipulation of the LDAP entries.

An entry with aliases contains multiple values of the *krbPrincipalName* attribute. Since LDAP attribute values are not ordered, it is necessary to specify which principal name is canonical, by using the *krbCanonicalName* attribute. Therefore, to create aliases for an entry, first set the *krbCanonicalName* attribute of the entry to the canonical principal name (which should be identical to the pre-existing *krbPrincipalName* value), and then add additional *krbPrincipalName* attributes for the aliases.

Principal aliases are only returned by the KDC when the client requests canonicalization. Canonicalization is normally requested for service principals; for client principals, an explicit flag is often required (e.g., `kinit -C`) and canonicalization is only performed for initial ticket requests.

CHAPTER

EIGHT

APPLICATION SERVERS

If you need to install the Kerberos V5 programs on an application server, please refer to the Kerberos V5 Installation Guide. Once you have installed the software, you need to add that host to the Kerberos database (see *Principals*), and generate a keytab for that host, that contains the host's key. You also need to make sure the host's clock is within your maximum clock skew of the KDCs.

8.1 Keytabs

A keytab is a host's copy of its own keylist, which is analogous to a user's password. An application server that needs to authenticate itself to the KDC has to have a keytab that contains its own principal and key. Just as it is important for users to protect their passwords, it is equally important for hosts to protect their keytabs. You should always store keytab files on local disk, and make them readable only by root, and you should never send a keytab file over a network in the clear. Ideally, you should run the *kadmin* command to extract a keytab on the host on which the keytab is to reside.

8.1.1 Adding principals to keytabs

To generate a keytab, or to add a principal to an existing keytab, use the **ktadd** command from kadmin. Here is a sample session, using configuration files that enable only AES encryption:

```
kadmin: ktadd host/daffodil.mit.edu@ATHENA.MIT.EDU
Entry for principal host/daffodil.mit.edu with kvno 2, encryption type aes256-cts-hmac-
↪sha1-96 added to keytab FILE:/etc/krb5.keytab
Entry for principal host/daffodil.mit.edu with kvno 2, encryption type aes128-cts-hmac-
↪sha1-96 added to keytab FILE:/etc/krb5.keytab
```

8.1.2 Removing principals from keytabs

To remove a principal from an existing keytab, use the kadmin **ktremove** command:

```
kadmin:  ktremove host/daffodil.mit.edu@ATHENA.MIT.EDU
Entry for principal host/daffodil.mit.edu with kvno 2 removed from keytab FILE:/etc/krb5.
↪keytab.
Entry for principal host/daffodil.mit.edu with kvno 2 removed from keytab FILE:/etc/krb5.
↪keytab.
```

8.1.3 Using a keytab to acquire client credentials

While keytabs are ordinarily used to accept credentials from clients, they can also be used to acquire initial credentials, allowing one service to authenticate to another.

To manually obtain credentials using a keytab, use the kinit(1) **-k** option, together with the **-t** option if the keytab is not in the default location.

Beginning with release 1.11, GSSAPI applications can be configured to automatically obtain initial credentials from a keytab as needed. The recommended configuration is as follows:

1. Create a keytab containing a single entry for the desired client identity.
2. Place the keytab in a location readable by the service, and set the **KRB5_CLIENT_KTNAME** environment variable to its filename. Alternatively, use the **default_client_keytab_name** profile variable in *[libdefaults]*, or use the default location of *DEFCKTNAME*.
3. Set **KRB5CCNAME** to a filename writable by the service, which will not be used for any other purpose. Do not manually obtain credentials at this location. (Another credential cache type besides **FILE** can be used if desired, as long the cache will not conflict with another use. A **MEMORY** cache can be used if the service runs as a long-lived process. See ccache_definition for details.)
4. Start the service. When it authenticates using GSSAPI, it will automatically obtain credentials from the client keytab into the specified credential cache, and refresh them before they expire.

8.2 Clock Skew

A Kerberos application server host must keep its clock synchronized or it will reject authentication requests from clients. Modern operating systems typically provide a facility to maintain the correct time; make sure it is enabled. This is especially important on virtual machines, where clocks tend to drift more rapidly than normal machine clocks.

The default allowable clock skew is controlled by the **clockskew** variable in *[libdefaults]*.

8.3 Getting DNS information correct

Several aspects of Kerberos rely on name service. When a hostname is used to name a service, clients may canonicalize the hostname using forward and possibly reverse name resolution. The result of this canonicalization must match the principal entry in the host's keytab, or authentication will fail. To work with all client canonicalization configurations, each host's canonical name must be the fully-qualified host name (including the domain), and each host's IP address must reverse-resolve to the canonical name.

Configuration of hostnames varies by operating system. On the application server itself, canonicalization will typically use the `/etc/hosts` file rather than the DNS. Ensure that the line for the server's hostname is in the following form:

```
IP address      fully-qualified hostname        aliases
```

Here is a sample `/etc/hosts` file:

```
# this is a comment
127.0.0.1       localhost localhost.mit.edu
10.0.0.6        daffodil.mit.edu daffodil trillium wake-robin
```

The output of `klist -k` for this example host should look like:

```
viola# klist -k
Keytab name: /etc/krb5.keytab
KVNO Principal
---- --------------------------------------------------------------------
   2 host/daffodil.mit.edu@ATHENA.MIT.EDU
```

If you were to ssh to this host with a fresh credentials cache (ticket file), and then klist(1), the output should list a service principal of `host/daffodil.mit.edu@ATHENA.MIT.EDU`.

8.4 Configuring your firewall to work with Kerberos V5

If you need off-site users to be able to get Kerberos tickets in your realm, they must be able to get to your KDC. This requires either that you have a replica KDC outside your firewall, or that you configure your firewall to allow UDP requests into at least one of your KDCs, on whichever port the KDC is running. (The default is port 88; other ports may be specified in the KDC's *kdc.conf* file.) Similarly, if you need off-site users to be able to change their passwords in your realm, they must be able to get to your Kerberos admin server on the kpasswd port (which defaults to 464). If you need off-site users to be able to administer your Kerberos realm, they must be able to get to your Kerberos admin server on the administrative port (which defaults to 749).

If your on-site users inside your firewall will need to get to KDCs in other realms, you will also need to configure your firewall to allow outgoing TCP and UDP requests to port 88, and to port 464 to allow password changes. If your on-site users inside your firewall will need to get to Kerberos admin servers in other realms, you will also need to allow outgoing TCP and UDP requests to port 749.

If any of your KDCs are outside your firewall, you will need to allow kprop requests to get through to the remote KDC. *kprop* uses the `krb5_prop` service on port 754 (tcp).

The book *UNIX System Security*, by David Curry, is a good starting point for learning to configure firewalls.

CHAPTER

NINE

HOST CONFIGURATION

All hosts running Kerberos software, whether they are clients, application servers, or KDCs, can be configured using *krb5.conf*. Here we describe some of the behavior changes you might want to make.

9.1 Default realm

In the *[libdefaults]* section, the **default_realm** realm relation sets the default Kerberos realm. For example:

```
[libdefaults]
    default_realm = ATHENA.MIT.EDU
```

The default realm affects Kerberos behavior in the following ways:

- When a principal name is parsed from text, the default realm is used if no `@REALM` component is specified.
- The default realm affects login authorization as described below.
- For programs which operate on a Kerberos database, the default realm is used to determine which database to operate on, unless the **-r** parameter is given to specify a realm.
- A server program may use the default realm when looking up its key in a *keytab file*, if its realm is not determined by *[domain_realm]* configuration or by the server program itself.
- If kinit(1) is passed the **-n** flag, it requests anonymous tickets from the default realm.

In some situations, these uses of the default realm might conflict. For example, it might be desirable for principal name parsing to use one realm by default, but for login authorization to use a second realm. In this situation, the first realm can be configured as the default realm, and **auth_to_local** relations can be used as described below to use the second realm for login authorization.

9.2 Login authorization

If a host runs a Kerberos-enabled login service such as OpenSSH with GSSAPIAuthentication enabled, login authorization rules determine whether a Kerberos principal is allowed to access a local account.

By default, a Kerberos principal is allowed access to an account if its realm matches the default realm and its name matches the account name. (For historical reasons, access is also granted by default if the name has two components and the second component matches the default realm; for instance, `alice/ATHENA.MIT.EDU@ATHENA.MIT.EDU` is granted access to the `alice` account if `ATHENA.MIT.EDU` is the default realm.)

The simplest way to control local access is using .k5login(5) files. To use these, place a `.k5login` file in the home directory of each account listing the principal names which should have login access to that account. If it is not desirable

to use `.k5login` files located in account home directories, the **k5login_directory** relation in the *[libdefaults]* section can specify a directory containing one file per account uname.

By default, if a `.k5login` file is present, it controls authorization both positively and negatively–any principal name contained in the file is granted access and any other principal name is denied access, even if it would have had access if the `.k5login` file didn't exist. The **k5login_authoritative** relation in the *[libdefaults]* section can be set to false to make `.k5login` files provide positive authorization only.

The **auth_to_local** relation in the *[realms]* section for the default realm can specify pattern-matching rules to control login authorization. For example, the following configuration allows access to principals from a different realm than the default realm:

```
[realms]
    DEFAULT.REALM = {
        # Allow access to principals from OTHER.REALM.
        #
        # [1:$1@$0] matches single-component principal names and creates
        # a selection string containing the principal name and realm.
        #
        # (.*@OTHER\.REALM) matches against the selection string, so that
        # only principals in OTHER.REALM are matched.
        #
        # s/@OTHER\.REALM$// removes the realm name, leaving behind the
        # principal name as the account name.
        auth_to_local = RULE:[1:$1@$0](.*@OTHER\.REALM)s/@OTHER\.REALM$//

        # Also allow principals from the default realm.  Omit this line
        # to only allow access to principals in OTHER.REALM.
        auth_to_local = DEFAULT
    }
```

The **auth_to_local_names** subsection of the *[realms]* section for the default realm can specify explicit mappings from principal names to local accounts. The key used in this subsection is the principal name without realm, so it is only safe to use in a Kerberos environment with a single realm or a tightly controlled set of realms. An example use of **auth_to_local_names** might be:

```
[realms]
    ATHENA.MIT.EDU = {
        auth_to_local_names = {
            # Careful, these match principals in any realm!
            host/example.com = hostaccount
            fred = localfred
        }
    }
```

Local authorization behavior can also be modified using plugin modules; see hostrealm_plugin for details.

9.3 Plugin module configuration

Many aspects of Kerberos behavior, such as client preauthentication and KDC service location, can be modified through the use of plugin modules. For most of these behaviors, you can use the *[plugins]* section of krb5.conf to register third-party modules, and to switch off registered or built-in modules.

A plugin module takes the form of a Unix shared object (`modname.so`) or Windows DLL (`modname.dll`). If you have installed a third-party plugin module and want to register it, you do so using the **module** relation in the appropriate subsection of the [plugins] section. The value for **module** must give the module name and the path to the module, separated by a colon. The module name will often be the same as the shared object's name, but in unusual cases (such as a shared object which implements multiple modules for the same interface) it might not be. For example, to register a client preauthentication module named `mypreauth` installed at `/path/to/mypreauth.so`, you could write:

```
[plugins]
    clpreauth = {
        module = mypreauth:/path/to/mypreauth.so
    }
```

Many of the pluggable behaviors in MIT krb5 contain built-in modules which can be switched off. You can disable a built-in module (or one you have registered) using the **disable** directive in the appropriate subsection of the [plugins] section. For example, to disable the use of .k5identity files to select credential caches, you could write:

```
[plugins]
    ccselect = {
        disable = k5identity
    }
```

If you want to disable multiple modules, specify the **disable** directive multiple times, giving one module to disable each time.

Alternatively, you can explicitly specify which modules you want to be enabled for that behavior using the **enable_only** directive. For example, to make *kadmind* check password quality using only a module you have registered, and no other mechanism, you could write:

```
[plugins]
    pwqual = {
        module = mymodule:/path/to/mymodule.so
        enable_only = mymodule
    }
```

Again, if you want to specify multiple modules, specify the **enable_only** directive multiple times, giving one module to enable each time.

Some Kerberos interfaces use different mechanisms to register plugin modules.

9.3.1 KDC location modules

For historical reasons, modules to control how KDC servers are located are registered simply by placing the shared object or DLL into the "libkrb5" subdirectory of the krb5 plugin directory, which defaults to *LIBDIR*`/krb5/plugins`. For example, Samba's winbind krb5 locator plugin would be registered by placing its shared object in *LIBDIR*`/krb5/plugins/libkrb5/winbind_krb5_locator.so`.

9.3.2 GSSAPI mechanism modules

GSSAPI mechanism modules are registered using the file *SYSCONFDIR*`/gss/mech` or configuration files in the *SYSCONFDIR*`/gss/mech.d` directory with a `.conf` suffix. Each line in these files has the form:

```
name  oid  pathname  [options]  <type>
```

Only the name, oid, and pathname are required. *name* is the mechanism name, which may be used for debugging or logging purposes. *oid* is the object identifier of the GSSAPI mechanism to be registered. *pathname* is a path to the module shared object or DLL. *options* (if present) are options provided to the plugin module, surrounded in square brackets. *type* (if present) can be used to indicate a special type of module. Currently the only special module type is "interposer", for a module designed to intercept calls to other mechanisms.

If the environment variable **GSS_MECH_CONFIG** is set, its value is used as the sole mechanism configuration filename.

9.3.3 Configuration profile modules

A configuration profile module replaces the information source for *krb5.conf* itself. To use a profile module, begin krb5.conf with the line:

```
module PATHNAME:STRING
```

where *PATHNAME* is a path to the module shared object or DLL, and *STRING* is a string to provide to the module. The module will then take over, and the rest of krb5.conf will be ignored.

CHAPTER

TEN

BACKUPS OF SECURE HOSTS

When you back up a secure host, you should exclude the host's keytab file from the backup. If someone obtained a copy of the keytab from a backup, that person could make any host masquerade as the host whose keytab was compromised. In many configurations, knowledge of the host's keytab also allows root access to the host. This could be particularly dangerous if the compromised keytab was from one of your KDCs. If the machine has a disk crash and the keytab file is lost, it is easy to generate another keytab file. (See *Adding principals to keytabs*.) If you are unable to exclude particular files from backups, you should ensure that the backups are kept as secure as the host's root password.

10.1 Backing up the Kerberos database

As with any file, it is possible that your Kerberos database could become corrupted. If this happens on one of the replica KDCs, you might never notice, since the next automatic propagation of the database would install a fresh copy. However, if it happens to the primary KDC, the corrupted database would be propagated to all of the replicas during the next propagation. For this reason, MIT recommends that you back up your Kerberos database regularly. Because the primary KDC is continuously dumping the database to a file in order to propagate it to the replica KDCs, it is a simple matter to have a cron job periodically copy the dump file to a secure machine elsewhere on your network. (Of course, it is important to make the host where these backups are stored as secure as your KDCs, and to encrypt its transmission across your network.) Then if your database becomes corrupted, you can load the most recent dump onto the primary KDC. (See *Dumping and loading a Kerberos database*.)

CHAPTER

ELEVEN

PKINIT CONFIGURATION

PKINIT is a preauthentication mechanism for Kerberos 5 which uses X.509 certificates to authenticate the KDC to clients and vice versa. PKINIT can also be used to enable anonymity support, allowing clients to communicate securely with the KDC or with application servers without authenticating as a particular client principal.

11.1 Creating certificates

PKINIT requires an X.509 certificate for the KDC and one for each client principal which will authenticate using PKINIT. For anonymous PKINIT, a KDC certificate is required, but client certificates are not. A commercially issued server certificate can be used for the KDC certificate, but generally cannot be used for client certificates.

The instruction in this section describe how to establish a certificate authority and create standard PKINIT certificates. Skip this section if you are using a commercially issued server certificate as the KDC certificate for anonymous PKINIT, or if you are configuring a client to use an Active Directory KDC.

11.1.1 Generating a certificate authority certificate

You can establish a new certificate authority (CA) for use with a PKINIT deployment with the commands:

```
openssl genrsa -out cakey.pem 2048
openssl req -key cakey.pem -new -x509 -out cacert.pem -days 3650
```

The second command will ask for the values of several certificate fields. These fields can be set to any values. You can adjust the expiration time of the CA certificate by changing the number after `-days`. Since the CA certificate must be deployed to client machines each time it changes, it should normally have an expiration time far in the future; however, expiration times after 2037 may cause interoperability issues in rare circumstances.

The result of these commands will be two files, cakey.pem and cacert.pem. cakey.pem will contain a 2048-bit RSA private key, which must be carefully protected. cacert.pem will contain the CA certificate, which must be placed in the filesystems of the KDC and each client host. cakey.pem will be required to create KDC and client certificates.

11.1.2 Generating a KDC certificate

A KDC certificate for use with PKINIT is required to have some unusual fields, which makes generating them with OpenSSL somewhat complicated. First, you will need a file containing the following:

```
[kdc_cert]
basicConstraints=CA:FALSE
keyUsage=nonRepudiation,digitalSignature,keyEncipherment,keyAgreement
extendedKeyUsage=1.3.6.1.5.2.3.5
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
issuerAltName=issuer:copy
subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:kdc_princ_name

[kdc_princ_name]
realm=EXP:0,GeneralString:${ENV::REALM}
principal_name=EXP:1,SEQUENCE:kdc_principal_seq

[kdc_principal_seq]
name_type=EXP:0,INTEGER:2
name_string=EXP:1,SEQUENCE:kdc_principals

[kdc_principals]
princ1=GeneralString:krbtgt
princ2=GeneralString:${ENV::REALM}
```

If the above contents are placed in extensions.kdc, you can generate and sign a KDC certificate with the following commands:

```
openssl genrsa -out kdckey.pem 2048
openssl req -new -out kdc.req -key kdckey.pem
env REALM=YOUR_REALMNAME openssl x509 -req -in kdc.req \
    -CAkey cakey.pem -CA cacert.pem -out kdc.pem -days 365 \
    -extfile extensions.kdc -extensions kdc_cert -CAcreateserial
rm kdc.req
```

The second command will ask for the values of certificate fields, which can be set to any values. In the third command, substitute your KDC's realm name for YOUR_REALMNAME. You can adjust the certificate's expiration date by changing the number after `-days`. Remember to create a new KDC certificate before the old one expires.

The result of this operation will be in two files, kdckey.pem and kdc.pem. Both files must be placed in the KDC's filesystem. kdckey.pem, which contains the KDC's private key, must be carefully protected.

If you examine the KDC certificate with `openssl x509 -in kdc.pem -text -noout`, OpenSSL will not know how to display the KDC principal name in the Subject Alternative Name extension, so it will appear as `othername:<unsupported>`. This is normal and does not mean anything is wrong with the KDC certificate.

11.1.3 Generating client certificates

PKINIT client certificates also must have some unusual certificate fields. To generate a client certificate with OpenSSL for a single-component principal name, you will need an extensions file (different from the KDC extensions file above) containing:

```
[client_cert]
basicConstraints=CA:FALSE
keyUsage=digitalSignature,keyEncipherment,keyAgreement
extendedKeyUsage=1.3.6.1.5.2.3.4
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
issuerAltName=issuer:copy
subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:princ_name

[princ_name]
realm=EXP:0,GeneralString:${ENV::REALM}
principal_name=EXP:1,SEQUENCE:principal_seq

[principal_seq]
name_type=EXP:0,INTEGER:1
name_string=EXP:1,SEQUENCE:principals

[principals]
princ1=GeneralString:${ENV::CLIENT}
```

If the above contents are placed in extensions.client, you can generate and sign a client certificate with the following commands:

```
openssl genrsa -out clientkey.pem 2048
openssl req -new -key clientkey.pem -out client.req
env REALM=YOUR_REALMNAME CLIENT=YOUR_PRINCNAME openssl x509 \
    -CAkey cakey.pem -CA cacert.pem -req -in client.req \
    -extensions client_cert -extfile extensions.client \
    -days 365 -out client.pem
rm client.req
```

Normally, the first two commands should be run on the client host, and the resulting client.req file transferred to the certificate authority host for the third command. As in the previous steps, the second command will ask for the values of certificate fields, which can be set to any values. In the third command, substitute your realm's name for YOUR_REALMNAME and the client's principal name (without realm) for YOUR_PRINCNAME. You can adjust the certificate's expiration date by changing the number after `-days`.

The result of this operation will be two files, clientkey.pem and client.pem. Both files must be present on the client's host; clientkey.pem, which contains the client's private key, must be protected from access by others.

As in the KDC certificate, OpenSSL will display the client principal name as `othername:<unsupported>` in the Subject Alternative Name extension of a PKINIT client certificate.

If the client principal name contains more than one component (e.g. `host/example.com@REALM`), the `[principals]` section of `extensions.client` must be altered to contain multiple entries. (Simply setting `CLIENT` to `host/example.com` would generate a certificate for `host\/example.com@REALM` which would not match the multi-component principal name.) For a two-component principal, the section should read:

```
[principals]
princ1=GeneralString:${ENV::CLIENT1}
princ2=GeneralString:${ENV::CLIENT2}
```

The environment variables `CLIENT1` and `CLIENT2` must then be set to the first and second components when running `openssl x509`.

11.2 Configuring the KDC

The KDC must have filesystem access to the KDC certificate (kdc.pem) and the KDC private key (kdckey.pem). Configure the following relation in the KDC's *kdc.conf* file, either in the *[kdcdefaults]* section or in a *[realms]* subsection (with appropriate pathnames):

```
pkinit_identity = FILE:/var/lib/krb5kdc/kdc.pem,/var/lib/krb5kdc/kdckey.pem
```

If any clients will authenticate using regular (as opposed to anonymous) PKINIT, the KDC must also have filesystem access to the CA certificate (cacert.pem), and the following configuration (with the appropriate pathname):

```
pkinit_anchors = FILE:/var/lib/krb5kdc/cacert.pem
```

Because of the larger size of requests and responses using PKINIT, you may also need to allow TCP access to the KDC:

```
kdc_tcp_listen = 88
```

Restart the *krb5kdc* daemon to pick up the configuration changes.

The principal entry for each PKINIT-using client must be configured to require preauthentication. Ensure this with the command:

```
kadmin -q 'modprinc +requires_preauth YOUR_PRINCNAME'
```

Starting with release 1.12, it is possible to remove the long-term keys of a principal entry, which can save some space in the database and help to clarify some PKINIT-related error conditions by not asking for a password:

```
kadmin -q 'purgekeys -all YOUR_PRINCNAME'
```

These principal options can also be specified at principal creation time as follows:

```
kadmin -q 'add_principal +requires_preauth -nokey YOUR_PRINCNAME'
```

By default, the KDC requires PKINIT client certificates to have the standard Extended Key Usage and Subject Alternative Name attributes for PKINIT. Starting in release 1.16, it is possible to authorize client certificates based on the subject or other criteria instead of the standard PKINIT Subject Alternative Name, by setting the **pkinit_cert_match** string attribute on each client principal entry. For example:

```
kadmin set_string user@REALM pkinit_cert_match "<SUBJECT>CN=user@REALM$"
```

The **pkinit_cert_match** string attribute follows the syntax used by the *krb5.conf* **pkinit_cert_match** relation. To allow the use of non-PKINIT client certificates, it will also be necessary to disable key usage checking using the **pkinit_eku_checking** relation; for example:

```
[kdcdefaults]
    pkinit_eku_checking = none
```

11.3 Configuring the clients

Client hosts must be configured to trust the issuing authority for the KDC certificate. For a newly established certificate authority, the client host must have filesystem access to the CA certificate (cacert.pem) and the following relation in *krb5.conf* in the appropriate *[realms]* subsection (with appropriate pathnames):

```
pkinit_anchors = FILE:/etc/krb5/cacert.pem
```

If the KDC certificate is a commercially issued server certificate, the issuing certificate is most likely included in a system directory. You can specify it by filename as above, or specify the whole directory like so:

```
pkinit_anchors = DIR:/etc/ssl/certs
```

A commercially issued server certificate will usually not have the standard PKINIT principal name or Extended Key Usage extensions, so the following additional configuration is required:

```
pkinit_eku_checking = kpServerAuth
pkinit_kdc_hostname = hostname.of.kdc.certificate
```

Multiple **pkinit_kdc_hostname** relations can be configured to recognize multiple KDC certificates. If the KDC is an Active Directory domain controller, setting **pkinit_kdc_hostname** is necessary, but it should not be necessary to set **pkinit_eku_checking**.

To perform regular (as opposed to anonymous) PKINIT authentication, a client host must have filesystem access to a client certificate (client.pem), and the corresponding private key (clientkey.pem). Configure the following relations in the client host's *krb5.conf* file in the appropriate *[realms]* subsection (with appropriate pathnames):

```
pkinit_identities = FILE:/etc/krb5/client.pem,/etc/krb5/clientkey.pem
```

If the KDC and client are properly configured, it should now be possible to run `kinit username` without entering a password.

11.4 Anonymous PKINIT

Anonymity support in Kerberos allows a client to obtain a ticket without authenticating as any particular principal. Such a ticket can be used as a FAST armor ticket, or to securely communicate with an application server anonymously.

To configure anonymity support, you must generate or otherwise procure a KDC certificate and configure the KDC host, but you do not need to generate any client certificates. On the KDC, you must set the **pkinit_identity** variable to provide the KDC certificate, but do not need to set the **pkinit_anchors** variable or store the issuing certificate if you won't have any client certificates to verify. On client hosts, you must set the **pkinit_anchors** variable (and possibly **pkinit_kdc_hostname** and **pkinit_eku_checking**) in order to trust the issuing authority for the KDC certificate, but do not need to set the **pkinit_identities** variable.

Anonymity support is not enabled by default. To enable it, you must create the principal `WELLKNOWN/ANONYMOUS` using the command:

```
kadmin -q 'addprinc -randkey WELLKNOWN/ANONYMOUS'
```

Some Kerberos deployments include application servers which lack proper access control, and grant some level of access to any user who can authenticate. In such an environment, enabling anonymity support on the KDC would present a security issue. If you need to enable anonymity support for TGTs (for use as FAST armor tickets) without enabling anonymous authentication to application servers, you can set the variable **restrict_anonymous_to_tgt** to `true` in the appropriate *[realms]* subsection of the KDC's *kdc.conf* file.

To obtain anonymous credentials on a client, run `kinit -n`, or `kinit -n @REALMNAME` to specify a realm. The resulting tickets will have the client name `WELLKNOWN/ANONYMOUS@WELLKNOWN:ANONYMOUS`.

11.5 Freshness tokens

Freshness tokens can ensure that the client has recently had access to its certificate private key. If freshness tokens are not required by the KDC, a client program with temporary possession of the private key can compose requests for future timestamps and use them later.

In release 1.17 and later, freshness tokens are supported by the client and are sent by the KDC when the client indicates support for them. Because not all clients support freshness tokens yet, they are not required by default. To check if freshness tokens are supported by a realm's clients, look in the KDC logs for the lines:

```
PKINIT: freshness token received from <client principal>
PKINIT: no freshness token received from <client principal>
```

To require freshness tokens for all clients in a realm (except for clients authenticating anonymously), set the **pkinit_require_freshness** variable to `true` in the appropriate *[realms]* subsection of the KDC's *kdc.conf* file. To test that this option is in effect, run `kinit -X disable_freshness` and verify that authentication is unsuccessful.

CHAPTER

TWELVE

OTP PREAUTHENTICATION

OTP is a preauthentication mechanism for Kerberos 5 which uses One Time Passwords (OTP) to authenticate the client to the KDC. The OTP is passed to the KDC over an encrypted FAST channel in clear-text. The KDC uses the password along with per-user configuration to proxy the request to a third-party RADIUS system. This enables out-of-the-box compatibility with a large number of already widely deployed proprietary systems.

Additionally, our implementation of the OTP system allows for the passing of RADIUS requests over a UNIX domain stream socket. This permits the use of a local companion daemon which can handle the details of authentication.

12.1 Defining token types

Token types are defined in either *krb5.conf* or *kdc.conf* according to the following format:

```
[otp]
    <name> = {
        server = <host:port or filename> (default: see below)
        secret = <filename>
        timeout = <integer> (default: 5 [seconds])
        retries = <integer> (default: 3)
        strip_realm = <boolean> (default: true)
        indicator = <string> (default: none)
    }
```

If the server field begins with '/', it will be interpreted as a UNIX socket. Otherwise, it is assumed to be in the format host:port. When a UNIX domain socket is specified, the secret field is optional and an empty secret is used by default. If the server field is not specified, it defaults to *RUNSTATEDIR*`/krb5kdc/<name>.socket`.

When forwarding the request over RADIUS, by default the principal is used in the User-Name attribute of the RADIUS packet. The strip_realm parameter controls whether the principal is forwarded with or without the realm portion.

If an indicator field is present, tickets issued using this token type will be annotated with the specified authentication indicator (see *Authentication indicators*). This key may be specified multiple times to add multiple indicators.

12.2 The default token type

A default token type is used internally when no token type is specified for a given user. It is defined as follows:

```
[otp]
    DEFAULT = {
        strip_realm = false
    }
```

The administrator may override the internal `DEFAULT` token type simply by defining a configuration with the same name.

12.3 Token instance configuration

To enable OTP for a client principal, the administrator must define the **otp** string attribute for that principal. (See *set_string*.) The **otp** user string is a JSON string of the format:

```
[{
    "type": <string>,
    "username": <string>,
    "indicators": [<string>, ...]
 }, ...]
```

This is an array of token objects. Both fields of token objects are optional. The **type** field names the token type of this token; if not specified, it defaults to `DEFAULT`. The **username** field specifies the value to be sent in the User-Name RADIUS attribute. If not specified, the principal name is sent, with or without realm as defined in the token type. The **indicators** field specifies a list of authentication indicators to annotate tickets with, overriding any indicators specified in the token type.

For ease of configuration, an empty array (`[]`) is treated as equivalent to one DEFAULT token (`[{}]`).

12.4 Other considerations

1. FAST is required for OTP to work.

CHAPTER

THIRTEEN

SPAKE PREAUTHENTICATION

SPAKE preauthentication (added in release 1.17) uses public key cryptography techniques to protect against *password dictionary attacks*. Unlike *PKINIT*, it does not require any additional infrastructure such as certificates; it simply needs to be turned on. Using SPAKE preauthentication may modestly increase the CPU and network load on the KDC.

SPAKE preauthentication can use one of four elliptic curve groups for its password-authenticated key exchange. The recommended group is `edwards25519`; three NIST curves (`P-256`, `P-384`, and `P-521`) are also supported.

By default, SPAKE with the `edwards25519` group is enabled on clients, but the KDC does not offer SPAKE by default. To turn it on, set the **spake_preauth_groups** variable in *[libdefaults]* to a list of allowed groups. This variable affects both the client and the KDC. Simply setting it to `edwards25519` is recommended:

```
[libdefaults]
    spake_preauth_groups = edwards25519
```

Set the **+requires_preauth** and **-allow_svr** flags on client principal entries, as you would for any preauthentication mechanism:

```
kadmin: modprinc +requires_preauth -allow_svr PRINCNAME
```

Clients which do not implement SPAKE preauthentication will fall back to encrypted timestamp.

An active attacker can force a fallback to encrypted timestamp by modifying the initial KDC response, defeating the protection against dictionary attacks. To prevent this fallback on clients which do implement SPAKE preauthentication, set the **disable_encrypted_timestamp** variable to `true` in the *[realms]* subsection for realms whose KDCs offer SPAKE preauthentication.

By default, SPAKE preauthentication requires an extra network round trip to the KDC during initial authentication. If most of the clients in a realm support SPAKE, this extra round trip can be eliminated using an optimistic challenge, by setting the **spake_preauth_kdc_challenge** variable in *[kdcdefaults]* to a single group name:

```
[kdcdefaults]
    spake_preauth_kdc_challenge = edwards25519
```

Using optimistic challenge will cause the KDC to do extra work for initial authentication requests that do not result in SPAKE preauthentication, but will save work when SPAKE preauthentication is used.

CHAPTER

FOURTEEN

ADDRESSING DICTIONARY ATTACK RISKS

Kerberos initial authentication is normally secured using the client principal's long-term key, which for users is generally derived from a password. Using a pasword-derived long-term key carries the risk of a dictionary attack, where an attacker tries a sequence of possible passwords, possibly requiring much less effort than would be required to try all possible values of the key. Even if *password policy objects* are used to force users not to pick trivial passwords, dictionary attacks can sometimes be successful against a significant fraction of the users in a realm. Dictionary attacks are not a concern for principals using random keys.

A dictionary attack may be online or offline. An online dictionary attack is performed by trying each password in a separate request to the KDC, and is therefore visible to the KDC and also limited in speed by the KDC's processing power and the network capacity between the client and the KDC. Online dictionary attacks can be mitigated using *account lockout*. This measure is not totally satisfactory, as it makes it easy for an attacker to deny access to a client principal.

An offline dictionary attack is performed by obtaining a ciphertext generated using the password-derived key, and trying each password against the ciphertext. This category of attack is invisible to the KDC and can be performed much faster than an online attack. The attack will generally take much longer with more recent encryption types (particularly the ones based on AES), because those encryption types use a much more expensive string-to-key function. However, the best defense is to deny the attacker access to a useful ciphertext. The required defensive measures depend on the attacker's level of network access.

An off-path attacker has no access to packets sent between legitimate users and the KDC. An off-path attacker could gain access to an attackable ciphertext either by making an AS request for a client principal which does not have the **+requires_preauth** flag, or by making a TGS request (after authenticating as a different user) for a server principal which does not have the **-allow_svr** flag. To address off-path attackers, a KDC administrator should set those flags on principals with password-derived keys:

```
kadmin: add_principal +requires_preauth -allow_svr princname
```

An attacker with passive network access (one who can monitor packets sent between legitimate users and the KDC, but cannot change them or insert their own packets) can gain access to an attackable ciphertext by observing an authentication by a user using the most common form of preauthentication, encrypted timestamp. Any of the following methods can prevent dictionary attacks by attackers with passive network access:

- Enabling *SPAKE preauthentication* (added in release 1.17) on the KDC, and ensuring that all clients are able to support it.
- Using an *HTTPS proxy* for communication with the KDC, if the attacker cannot monitor communication between the proxy server and the KDC.
- Using FAST, protecting the initial authentication with either a random key (such as a host key) or with *anonymous PKINIT*.

An attacker with active network access (one who can inject or modify packets sent between legitimate users and the KDC) can try to fool the client software into sending an attackable ciphertext using an encryption type and salt string of the attacker's choosing. Any of the following methods can prevent dictionary attacks by active attackers:

- Enabling SPAKE preauthentication and setting the **disable_encrypted_timestamp** variable to `true` in the *[realms]* subsection of the client configuration.
- Using an HTTPS proxy as described above, configured in the client's krb5.conf realm configuration. If *KDC discovery* is used to locate a proxy server, an active attacker may be able to use DNS spoofing to cause the client to use a different HTTPS server or to not use HTTPS.
- Using FAST as described above.

If *PKINIT* or *OTP* are used for initial authentication, the principal's long-term keys are not used and dictionary attacks are usually not a concern.

CHAPTER

FIFTEEN

PRINCIPAL NAMES AND DNS

Kerberos clients can do DNS lookups to canonicalize service principal names. This can cause difficulties when setting up Kerberos application servers, especially when the client's name for the service is different from what the service thinks its name is.

15.1 Service principal names

A frequently used kind of principal name is the host-based service principal name. This kind of principal name has two components: a service name and a hostname. For example, `imap/imap.example.com` is the principal name of the "imap" service on the host "imap.example.com". Other possible service names for the first component include "host" (remote login services such as ssh), "HTTP", and "nfs" (Network File System).

Service administrators often publish well-known hostname aliases that they would prefer users to use instead of the canonical name of the service host. This gives service administrators more flexibility in deploying services. For example, a shell login server might be named "long-vanity-hostname.example.com", but users will naturally prefer to type something like "login.example.com". Hostname aliases also allow for administrators to set up load balancing for some sorts of services based on rotating `CNAME` records in DNS.

15.2 Service principal canonicalization

In the MIT krb5 client library, canonicalization of host-based service principals is controlled by the **dns_canonicalize_hostname**, **rnds**, and **qualify_shortname** variables in *[libdefaults]*.

If **dns_canonicalize_hostname** is set to `true` (the default value), the client performs forward resolution by looking up the IPv4 and/or IPv6 addresses of the hostname using `getaddrinfo()`. This process will typically add a domain suffix to the hostname if needed, and follow CNAME records in the DNS. If **rdns** is also set to `true` (the default), the client will then perform a reverse lookup of the first returned Internet address using `getnameinfo()`, finding the name associated with the PTR record.

If **dns_canonicalize_hostname** is set to `false`, the hostname is not canonicalized using DNS. If the hostname has only one component (i.e. it contains no "." characters), the host's primary DNS search domain will be appended, if there is one. The **qualify_shortname** variable can be used to override or disable this suffix.

If **dns_canonicalize_hostname** is set to `fallback` (added in release 1.18), the hostname is initially treated according to the rules for `dns_canonicalize_hostname=false`. If a ticket request fails because the service principal is unknown, the hostname will be canonicalized according to the rules for `dns_canonicalize_hostname=true` and the request will be retried.

In all cases, the hostname is converted to lowercase, and any trailing dot is removed.

15.3 Reverse DNS mismatches

Sometimes, an enterprise will have control over its forward DNS but not its reverse DNS. The reverse DNS is sometimes under the control of the Internet service provider of the enterprise, and the enterprise may not have much influence in setting up reverse DNS records for its address space. If there are difficulties with getting forward and reverse DNS to match, it is best to set `rdns = false` on client machines.

15.4 Overriding application behavior

Applications can choose to use a default hostname component in their service principal name when accepting authentication, which avoids some sorts of hostname mismatches. Because not all relevant applications do this yet, using the *krb5.conf* setting:

```
[libdefaults]
    ignore_acceptor_hostname = true
```

will allow the Kerberos library to override the application's choice of service principal hostname and will allow a server program to accept incoming authentications using any key in its keytab that matches the service name and realm name (if given). This setting defaults to "false" and is available in releases krb5-1.10 and later.

15.5 Provisioning keytabs

One service principal entry that should be in the keytab is a principal whose hostname component is the canonical hostname that `getaddrinfo()` reports for all known aliases for the host. If the reverse DNS information does not match this canonical hostname, an additional service principal entry should be in the keytab for this different hostname.

15.6 Specific application advice

15.6.1 Secure shell (ssh)

Setting `GSSAPIStrictAcceptorCheck = no` in the configuration file of modern versions of the openssh daemon will allow the daemon to try any key in its keytab when accepting a connection, rather than looking for the keytab entry that matches the host's own idea of its name (typically the name that `gethostname()` returns). This requires krb5-1.10 or later.

15.6.2 OpenLDAP (ldapsearch, etc.)

OpenLDAP's SASL implementation performs reverse DNS lookup in order to canonicalize service principal names, even if **rdns** is set to `false` in the Kerberos configuration. To disable this behavior, add `SASL_NOCANON on` to `ldap.conf`, or set the `LDAPSASL_NOCANON` environment variable.

CHAPTER

SIXTEEN

ENCRYPTION TYPES

Kerberos can use a variety of cipher algorithms to protect data. A Kerberos **encryption type** (also known as an **enctype**) is a specific combination of a cipher algorithm with an integrity algorithm to provide both confidentiality and integrity to data.

16.1 Enctypes in requests

Clients make two types of requests (KDC-REQ) to the KDC: AS-REQs and TGS-REQs. The client uses the AS-REQ to obtain initial tickets (typically a Ticket-Granting Ticket (TGT)), and uses the TGS-REQ to obtain service tickets.

The KDC uses three different keys when issuing a ticket to a client:

- The long-term key of the service: the KDC uses this to encrypt the actual service ticket. The KDC only uses the first long-term key in the most recent kvno for this purpose.
- The session key: the KDC randomly chooses this key and places one copy inside the ticket and the other copy inside the encrypted part of the reply.
- The reply-encrypting key: the KDC uses this to encrypt the reply it sends to the client. For AS replies, this is a long-term key of the client principal. For TGS replies, this is either the session key of the authenticating ticket, or a subsession key.

Each of these keys is of a specific enctype.

Each request type allows the client to submit a list of enctypes that it is willing to accept. For the AS-REQ, this list affects both the session key selection and the reply-encrypting key selection. For the TGS-REQ, this list only affects the session key selection.

16.2 Session key selection

The KDC chooses the session key enctype by taking the intersection of its **permitted_enctypes** list, the list of long-term keys for the most recent kvno of the service, and the client's requested list of enctypes. Starting in krb5-1.21, all services are assumed to support aes256-cts-hmac-sha1-96; also, des3-cbc-sha1 and arcfour-hmac session keys will not be issued by default.

Starting in krb5-1.11, it is possible to set a string attribute on a service principal to control what session key enctypes the KDC may issue for service tickets for that principal, overriding the service's long-term keys and the assumption of aes256-cts-hmac-sha1-96 support. See *set_string* in *kadmin* for details.

16.3 Choosing enctypes for a service

Generally, a service should have a key of the strongest enctype that both it and the KDC support. If the KDC is running a release earlier than krb5-1.11, it is also useful to generate an additional key for each enctype that the service can support. The KDC will only use the first key in the list of long-term keys for encrypting the service ticket, but the additional long-term keys indicate the other enctypes that the service supports.

As noted above, starting with release krb5-1.11, there are additional configuration settings that control session key enctype selection independently of the set of long-term keys that the KDC has stored for a service principal.

16.4 Configuration variables

The following `[libdefaults]` settings in *krb5.conf* will affect how enctypes are chosen.

allow_weak_crypto
: defaults to *false* starting with krb5-1.8. When *false*, removes weak enctypes from **permitted_enctypes**, **default_tkt_enctypes**, and **default_tgs_enctypes**. Do not set this to *true* unless the use of weak enctypes is an acceptable risk for your environment and the weak enctypes are required for backward compatibility.

allow_des3
: was added in release 1.21 and defaults to *false*. Unless this flag is set to *true*, the KDC will not issue tickets with des3-cbc-sha1 session keys. In a future release, this flag will control whether des3-cbc-sha1 is permitted in similar fashion to weak enctypes.

allow_rc4
: was added in release 1.21 and defaults to *false*. Unless this flag is set to *true*, the KDC will not issue tickets with arcfour-hmac session keys. In a future release, this flag will control whether arcfour-hmac is permitted in similar fashion to weak enctypes.

permitted_enctypes
: controls the set of enctypes that a service will permit for session keys and for ticket and authenticator encryption. The KDC and other programs that access the Kerberos database will ignore keys of non-permitted enctypes. Starting in release 1.18, this setting also acts as the default for **default_tkt_enctypes** and **default_tgs_enctypes**.

default_tkt_enctypes
: controls the default set of enctypes that the Kerberos client library requests when making an AS-REQ. Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

default_tgs_enctypes
: controls the default set of enctypes that the Kerberos client library requests when making a TGS-REQ. Do not set this unless required for specific backward compatibility purposes; stale values of this setting can prevent clients from taking advantage of new stronger enctypes when the libraries are upgraded.

The following per-realm setting in *kdc.conf* affects the generation of long-term keys.

supported_enctypes
: controls the default set of enctype-salttype pairs that *kadmind* will use for generating long-term keys, either randomly or from passwords

16.5 Enctype compatibility

See *Encryption types* for additional information about enctypes.

enctype	weak?	krb5	Windows
des-cbc-crc	weak	<1.18	>=2000
des-cbc-md4	weak	<1.18	?
des-cbc-md5	weak	<1.18	>=2000
des3-cbc-sha1	deprecated	>=1.1	none
arcfour-hmac	deprecated	>=1.3	>=2000
arcfour-hmac-exp	weak	>=1.3	>=2000
aes128-cts-hmac-sha1-96		>=1.3	>=Vista
aes256-cts-hmac-sha1-96		>=1.3	>=Vista
aes128-cts-hmac-sha256-128		>=1.15	none
aes256-cts-hmac-sha384-192		>=1.15	none
camellia128-cts-cmac		>=1.9	none
camellia256-cts-cmac		>=1.9	none

krb5 releases 1.18 and later do not support single-DES. krb5 releases 1.8 and later disable the single-DES enctypes by default. Microsoft Windows releases Windows 7 and later disable single-DES enctypes by default.

krb5 releases 1.17 and later flag deprecated encryption types (including `des3-cbc-sha1` and `arcfour-hmac`) in KDC logs and kadmin output. krb5 release 1.19 issues a warning during initial authentication if `des3-cbc-sha1` is used. Future releases will disable `des3-cbc-sha1` by default and eventually remove support for it.

16.6 Migrating away from older encryption types

Administrator intervention may be required to migrate a realm away from legacy encryption types, especially if the realm was created using krb5 release 1.2 or earlier. This migration should be performed before upgrading to krb5 versions which disable or remove support for legacy encryption types.

If there is a **supported_enctypes** setting in *kdc.conf* on the KDC, make sure that it does not include weak or deprecated encryption types. This will ensure that newly created keys do not use those encryption types by default.

Check the `krbtgt/REALM` principal using the *kadmin* **getprinc** command. If it lists a weak or deprecated encryption type as the first key, it must be migrated using the procedure in *Changing the krbtgt key*.

Check the `kadmin/history` principal, which should have only one key entry. If it uses a weak or deprecated encryption type, it should be upgraded following the notes in *Updating the history key*.

Check the other kadmin principals: kadmin/changepw, kadmin/admin, and any kadmin/hostname principals that may exist. These principals can be upgraded with **change_password -randkey** in kadmin.

Check the `K/M` entry. If it uses a weak or deprecated encryption type, it should be upgraded following the procedure in *Updating the master key*.

User and service principals using legacy encryption types can be enumerated with the *kdb5_util* **tabdump keyinfo** command.

Service principals can be migrated with a keytab rotation on the service host, which can be accomplished using the *k5srvutil* **change** and **delold** commands. Allow enough time for existing tickets to expire between the change and delold operations.

User principals with password-based keys can be migrated with a password change. The realm administrator can set a password expiration date using the *kadmin* **modify_principal -pwexpire** command to force a password change.

If a legacy encryption type has not yet been disabled by default in the version of krb5 running on the KDC, it can be disabled administratively with the **permitted_enctypes** variable. For example, setting **permitted_enctypes** to `DEFAULT -des3 -rc4` will cause any database keys of the triple-DES and RC4 encryption types to be ignored.

CHAPTER

SEVENTEEN

HTTPS PROXY CONFIGURATION

In addition to being able to use UDP or TCP to communicate directly with a KDC as is outlined in RFC4120, and with kpasswd services in a similar fashion, the client libraries can attempt to use an HTTPS proxy server to communicate with a KDC or kpasswd service, using the protocol outlined in [MS-KKDCP].

Communicating with a KDC through an HTTPS proxy allows clients to contact servers when network firewalls might otherwise prevent them from doing so. The use of TLS also encrypts all traffic between the clients and the KDC, preventing observers from conducting password dictionary attacks or from observing the client and server principals being authenticated, at additional computational cost to both clients and servers.

An HTTPS proxy server is provided as a feature in some versions of Microsoft Windows Server, and a WSGI implementation named *kdcproxy* is available in the python package index.

17.1 Configuring the clients

To use an HTTPS proxy, a client host must trust the CA which issued that proxy's SSL certificate. If that CA's certificate is not in the system-wide default set of trusted certificates, configure the following relation in the client host's *krb5.conf* file in the appropriate *[realms]* subsection:

```
http_anchors = FILE:/etc/krb5/cacert.pem
```

Adjust the pathname to match the path of the file which contains a copy of the CA's certificate. The *http_anchors* option is documented more fully in *krb5.conf*.

Configure the client to access the KDC and kpasswd service by specifying their locations in its *krb5.conf* file in the form of HTTPS URLs for the proxy server:

```
kdc = https://server.fqdn/KdcProxy
kpasswd_server = https://server.fqdn/KdcProxy
```

If the proxy and client are properly configured, client commands such as `kinit`, `kvno`, and `kpasswd` should all function normally.

CHAPTER

EIGHTEEN

AUTHENTICATION INDICATORS

As of release 1.14, the KDC can be configured to annotate tickets if the client authenticated using a stronger preauthentication mechanism such as *PKINIT* or *OTP*. These annotations are called "authentication indicators." Service principals can be configured to require particular authentication indicators in order to authenticate to that service. An authentication indicator value can be any string chosen by the KDC administrator; there are no pre-set values.

To use authentication indicators with PKINIT or OTP, first configure the KDC to include an indicator when that preauthentication mechanism is used. For PKINIT, use the **pkinit_indicator** variable in *kdc.conf* . For OTP, use the **indicator** variable in the token type definition, or specify the indicators in the **otp** user string as described in *OTP Preauthentication*.

To require an indicator to be present in order to authenticate to a service principal, set the **require_auth** string attribute on the principal to the indicator value to be required. If you wish to allow one of several indicators to be accepted, you can specify multiple indicator values separated by spaces.

For example, a realm could be configured to set the authentication indicator value "strong" when PKINIT is used to authenticate, using a setting in the *[realms]* subsection:

```
pkinit_indicator = strong
```

A service principal could be configured to require the "strong" authentication indicator value:

```
$ kadmin setstr host/high.value.server require_auth strong
Password for user/admin@KRBTEST.COM:
```

A user who authenticates with PKINIT would be able to obtain a ticket for the service principal:

```
$ kinit -X X509_user_identity=FILE:/my/cert.pem,/my/key.pem user
$ kvno host/high.value.server
host/high.value.server@KRBTEST.COM: kvno = 1
```

but a user who authenticates with a password would not:

```
$ kinit user
Password for user@KRBTEST.COM:
$ kvno host/high.value.server
kvno: KDC policy rejects request while getting credentials for
  host/high.value.server@KRBTEST.COM
```

GSSAPI server applications can inspect authentication indicators through the auth-indicators name attribute.

CHAPTER

NINETEEN

ADMINISTRATION PROGRAMS

19.1 kadmin

19.1.1 SYNOPSIS

kadmin [**-O**|**-N**] [**-r** *realm*] [**-p** *principal*] [**-q** *query*] [[**-c** *cache_name*]|[**-k** [**-t** *keytab*]]|**-n**] [**-w** *password*] [**-s** *admin_server*[:*port*]] [command args...]

kadmin.local [**-r** *realm*] [**-p** *principal*] [**-q** *query*] [**-d** *dbname*] [**-e** *enc*:*salt* ...] [**-m**] [**-x** *db_args*] [command args...]

19.1.2 DESCRIPTION

kadmin and kadmin.local are command-line interfaces to the Kerberos V5 administration system. They provide nearly identical functionalities; the difference is that kadmin.local directly accesses the KDC database, while kadmin performs operations using *kadmind*. Except as explicitly noted otherwise, this man page will use "kadmin" to refer to both versions. kadmin provides for the maintenance of Kerberos principals, password policies, and service key tables (keytabs).

The remote kadmin client uses Kerberos to authenticate to kadmind using the service principal `kadmin/admin` or `kadmin/ADMINHOST` (where *ADMINHOST* is the fully-qualified hostname of the admin server). If the credentials cache contains a ticket for one of these principals, and the **-c** credentials_cache option is specified, that ticket is used to authenticate to kadmind. Otherwise, the **-p** and **-k** options are used to specify the client Kerberos principal name used to authenticate. Once kadmin has determined the principal name, it requests a service ticket from the KDC, and uses that service ticket to authenticate to kadmind.

Since kadmin.local directly accesses the KDC database, it usually must be run directly on the primary KDC with sufficient permissions to read the KDC database. If the KDC database uses the LDAP database module, kadmin.local can be run on any host which can access the LDAP server.

19.1.3 OPTIONS

-r *realm*
: Use *realm* as the default database realm.

-p *principal*
: Use *principal* to authenticate. Otherwise, kadmin will append `/admin` to the primary principal name of the default ccache, the value of the **USER** environment variable, or the username as obtained with getpwuid, in order of preference.

-k
: Use a keytab to decrypt the KDC response instead of prompting for a password. In this case, the default principal will be `host/hostname`. If there is no keytab specified with the **-t** option, then the default keytab will be used.

-t ***keytab***
Use *keytab* to decrypt the KDC response. This can only be used with the **-k** option.

-n
Requests anonymous processing. Two types of anonymous principals are supported. For fully anonymous Kerberos, configure PKINIT on the KDC and configure **pkinit_anchors** in the client's *krb5.conf*. Then use the **-n** option with a principal of the form `@REALM` (an empty principal name followed by the at-sign and a realm name). If permitted by the KDC, an anonymous ticket will be returned. A second form of anonymous tickets is supported; these realm-exposed tickets hide the identity of the client but not the client's realm. For this mode, use `kinit -n` with a normal principal name. If supported by the KDC, the principal (but not realm) will be replaced by the anonymous principal. As of release 1.8, the MIT Kerberos KDC only supports fully anonymous operation.

-c ***credentials_cache***
Use *credentials_cache* as the credentials cache. The cache should contain a service ticket for the `kadmin/admin` or `kadmin/ADMINHOST` (where *ADMINHOST* is the fully-qualified hostname of the admin server) service; it can be acquired with the kinit(1) program. If this option is not specified, kadmin requests a new service ticket from the KDC, and stores it in its own temporary ccache.

-w ***password***
Use *password* instead of prompting for one. Use this option with care, as it may expose the password to other users on the system via the process list.

-q ***query***
Perform the specified query and then exit.

-d ***dbname***
Specifies the name of the KDC database. This option does not apply to the LDAP database module.

-s ***admin_server*****[:*****port*****]**
Specifies the admin server which kadmin should contact.

-m
If using kadmin.local, prompt for the database master password instead of reading it from a stash file.

-e "*enc*:*salt* ..."
Sets the keysalt list to be used for any new keys created. See *Keysalt lists* in *kdc.conf* for a list of possible values.

-O
Force use of old AUTH_GSSAPI authentication flavor.

-N
Prevent fallback to AUTH_GSSAPI authentication flavor.

-x ***db_args***
Specifies the database specific arguments. See the next section for supported options.

Starting with release 1.14, if any command-line arguments remain after the options, they will be treated as a single query to be executed. This mode of operation is intended for scripts and behaves differently from the interactive mode in several respects:

- Query arguments are split by the shell, not by kadmin.
- Informational and warning messages are suppressed. Error messages and query output (e.g. for **get_principal**) will still be displayed.
- Confirmation prompts are disabled (as if **-force** was given). Password prompts will still be issued as required.
- The exit status will be non-zero if the query fails.

The **-q** option does not carry these behavior differences; the query will be processed as if it was entered interactively. The **-q** option cannot be used in combination with a query in the remaining arguments.

19.1.4 DATABASE OPTIONS

Database options can be used to override database-specific defaults. Supported options for the DB2 module are:

-x dbname=*filename*
Specifies the base filename of the DB2 database.

-x lockiter
Make iteration operations hold the lock for the duration of the entire operation, rather than temporarily releasing the lock while handling each principal. This is the default behavior, but this option exists to allow command line override of a [dbmodules] setting. First introduced in release 1.13.

-x unlockiter
Make iteration operations unlock the database for each principal, instead of holding the lock for the duration of the entire operation. First introduced in release 1.13.

Supported options for the LDAP module are:

-x host=*ldapuri*
Specifies the LDAP server to connect to by a LDAP URI.

-x binddn=*bind_dn*
Specifies the DN used to bind to the LDAP server.

-x bindpwd=*password*
Specifies the password or SASL secret used to bind to the LDAP server. Using this option may expose the password to other users on the system via the process list; to avoid this, instead stash the password using the **stashsrvpw** command of *kdb5_ldap_util*.

-x sasl_mech=*mechanism*
Specifies the SASL mechanism used to bind to the LDAP server. The bind DN is ignored if a SASL mechanism is used. New in release 1.13.

-x sasl_authcid=*name*
Specifies the authentication name used when binding to the LDAP server with a SASL mechanism, if the mechanism requires one. New in release 1.13.

-x sasl_authzid=*name*
Specifies the authorization name used when binding to the LDAP server with a SASL mechanism. New in release 1.13.

-x sasl_realm=*realm*
Specifies the realm used when binding to the LDAP server with a SASL mechanism, if the mechanism uses one. New in release 1.13.

-x debug=*level*
sets the OpenLDAP client library debug level. *level* is an integer to be interpreted by the library. Debugging messages are printed to standard error. New in release 1.12.

19.1.5 COMMANDS

When using the remote client, available commands may be restricted according to the privileges specified in the *kadm5.acl* file on the admin server.

add_principal

add_principal [*options*] *newprinc*

Creates the principal *newprinc*, prompting twice for a password. If no password policy is specified with the **-policy** option, and the policy named `default` is assigned to the principal if it exists. However, creating a policy named `default` will not automatically assign this policy to previously existing principals. This policy assignment can be suppressed with the **-clearpolicy** option.

This command requires the **add** privilege.

Aliases: **addprinc**, **ank**

Options:

-expire *expdate*
(getdate string) The expiration date of the principal.

-pwexpire *pwexpdate*
(getdate string) The password expiration date.

-maxlife *maxlife*
(duration or getdate string) The maximum ticket life for the principal.

-maxrenewlife *maxrenewlife*
(duration or getdate string) The maximum renewable life of tickets for the principal.

-kvno *kvno*
The initial key version number.

-policy *policy*
The password policy used by this principal. If not specified, the policy `default` is used if it exists (unless **-clearpolicy** is specified).

-clearpolicy
Prevents any policy from being assigned when **-policy** is not specified.

{-|+}allow_postdated
-allow_postdated prohibits this principal from obtaining postdated tickets. **+allow_postdated** clears this flag.

{-|+}allow_forwardable
-allow_forwardable prohibits this principal from obtaining forwardable tickets. **+allow_forwardable** clears this flag.

{-|+}allow_renewable
-allow_renewable prohibits this principal from obtaining renewable tickets. **+allow_renewable** clears this flag.

{-|+}allow_proxiable
-allow_proxiable prohibits this principal from obtaining proxiable tickets. **+allow_proxiable** clears this flag.

{-|+}allow_dup_skey
-allow_dup_skey disables user-to-user authentication for this principal by prohibiting others from obtaining a service ticket encrypted in this principal's TGT session key. **+allow_dup_skey** clears this flag.

{-|+}requires_preauth
+requires_preauth requires this principal to preauthenticate before being allowed to kinit. **-requires_preauth** clears this flag. When **+requires_preauth** is set on a service principal, the KDC will only issue service tickets for that service principal if the client's initial authentication was performed using preauthentication.

{-|+}requires_hwauth
+requires_hwauth requires this principal to preauthenticate using a hardware device before being allowed to kinit. **-requires_hwauth** clears this flag. When **+requires_hwauth** is set on a service principal, the KDC will

only issue service tickets for that service principal if the client's initial authentication was performed using a hardware device to preauthenticate.

{-|+}ok_as_delegate
+ok_as_delegate sets the **okay as delegate** flag on tickets issued with this principal as the service. Clients may use this flag as a hint that credentials should be delegated when authenticating to the service. **-ok_as_delegate** clears this flag.

{-|+}allow_svr
-allow_svr prohibits the issuance of service tickets for this principal. In release 1.17 and later, user-to-user service tickets are still allowed unless the **-allow_dup_skey** flag is also set. **+allow_svr** clears this flag.

{-|+}allow_tgs_req
-allow_tgs_req specifies that a Ticket-Granting Service (TGS) request for a service ticket for this principal is not permitted. **+allow_tgs_req** clears this flag.

{-|+}allow_tix
-allow_tix forbids the issuance of any tickets for this principal. **+allow_tix** clears this flag.

{-|+}needchange
+needchange forces a password change on the next initial authentication to this principal. **-needchange** clears this flag.

{-|+}password_changing_service
+password_changing_service marks this principal as a password change service principal.

{-|+}ok_to_auth_as_delegate
+ok_to_auth_as_delegate allows this principal to acquire forwardable tickets to itself from arbitrary users, for use with constrained delegation.

{-|+}no_auth_data_required
+no_auth_data_required prevents PAC or AD-SIGNEDPATH data from being added to service tickets for the principal.

{-|+}lockdown_keys
+lockdown_keys prevents keys for this principal from leaving the KDC via kadmind. The chpass and extract operations are denied for a principal with this attribute. The chrand operation is allowed, but will not return the new keys. The delete and rename operations are also denied if this attribute is set, in order to prevent a malicious administrator from replacing principals like krbtgt/* or kadmin/* with new principals without the attribute. This attribute can be set via the network protocol, but can only be removed using kadmin.local.

-randkey
Sets the key of the principal to a random value.

-nokey
Causes the principal to be created with no key. New in release 1.12.

-pw ***password***
Sets the password of the principal to the specified string and does not prompt for a password. Note: using this option in a shell script may expose the password to other users on the system via the process list.

-e ***enc*:*salt*,...**
Uses the specified keysalt list for setting the keys of the principal. See *Keysalt lists* in *kdc.conf* for a list of possible values.

-x ***db_princ_args***
Indicates database-specific options. The options for the LDAP database module are:

-x dn=*dn*
Specifies the LDAP object that will contain the Kerberos principal being created.

-x linkdn=*dn*
Specifies the LDAP object to which the newly created Kerberos principal object will point.

-x containerdn=*container_dn*
Specifies the container object under which the Kerberos principal is to be created.

-x tktpolicy=*policy*
Associates a ticket policy to the Kerberos principal.

Note:

- The **containerdn** and **linkdn** options cannot be specified with the **dn** option.
- If the *dn* or *containerdn* options are not specified while adding the principal, the principals are created under the principal container configured in the realm or the realm container.
- *dn* and *containerdn* should be within the subtrees or principal container configured in the realm.

Example:

```
kadmin: addprinc jennifer
No policy specified for "jennifer@ATHENA.MIT.EDU";
defaulting to no policy.
Enter password for principal jennifer@ATHENA.MIT.EDU:
Re-enter password for principal jennifer@ATHENA.MIT.EDU:
Principal "jennifer@ATHENA.MIT.EDU" created.
kadmin:
```

modify_principal

modify_principal [*options*] *principal*

Modifies the specified principal, changing the fields as specified. The options to **add_principal** also apply to this command, except for the **-randkey**, **-pw**, and **-e** options. In addition, the option **-clearpolicy** will clear the current policy of a principal.

This command requires the *modify* privilege.

Alias: **modprinc**

Options (in addition to the **addprinc** options):

-unlock
Unlocks a locked principal (one which has received too many failed authentication attempts without enough time between them according to its password policy) so that it can successfully authenticate.

rename_principal

rename_principal [**-force**] *old_principal new_principal*

Renames the specified *old_principal* to *new_principal*. This command prompts for confirmation, unless the **-force** option is given.

This command requires the **add** and **delete** privileges.

Alias: **renprinc**

add_alias

add_alias *alias_princ target_princ*

Create an alias *alias_princ* pointing to *target_princ*. Aliases may be chained (that is, *target_princ* may itself be an alias) up to a depth of 10.

This command requires the **add** privilege for *alias_princ* and the **modify** privilege for *target_princ*.

(New in release 1.22.)

Aliases: **alias**

delete_principal

delete_principal [**-force**] *principal*

Deletes the specified *principal* or alias from the database. This command prompts for deletion, unless the **-force** option is given.

This command requires the **delete** privilege.

Alias: **delprinc**

change_password

change_password [*options*] *principal*

Changes the password of *principal*. Prompts for a new password if neither **-randkey** or **-pw** is specified.

This command requires the **changepw** privilege, or that the principal running the program is the same as the principal being changed.

Alias: **cpw**

The following options are available:

-randkey
Sets the key of the principal to a random value.

-pw ***password***
Set the password to the specified string. Using this option in a script may expose the password to other users on the system via the process list.

-e ***enc*:*salt*,...**
Uses the specified keysalt list for setting the keys of the principal. See *Keysalt lists* in *kdc.conf* for a list of possible values.

-keepold
Keeps the existing keys in the database. This flag is usually not necessary except perhaps for `krbtgt` principals.

Example:

```
kadmin: cpw systest
Enter password for principal systest@BLEEP.COM:
Re-enter password for principal systest@BLEEP.COM:
Password for systest@BLEEP.COM changed.
kadmin:
```

purgekeys

purgekeys [**-all**|**-keepkvno** *oldest_kvno_to_keep*] *principal*

Purges previously retained old keys (e.g., from **change_password -keepold**) from *principal*. If **-keepkvno** is specified, then only purges keys with kvnos lower than *oldest_kvno_to_keep*. If **-all** is specified, then all keys are purged. The **-all** option is new in release 1.12.

This command requires the **modify** privilege.

get_principal

get_principal [**-terse**] *principal*

Gets the attributes of principal. With the **-terse** option, outputs fields as quoted tab-separated strings.

This command requires the **inquire** privilege, or that the principal running the the program to be the same as the one being listed.

Alias: **getprinc**

Examples:

```
kadmin: getprinc tlyu/admin
Principal: tlyu/admin@BLEEP.COM
Expiration date: [never]
Last password change: Mon Aug 12 14:16:47 EDT 1996
Password expiration date: [never]
Maximum ticket life: 0 days 10:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Mon Aug 12 14:16:47 EDT 1996 (bjaspan/admin@BLEEP.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 1
Key: vno 1, aes256-cts-hmac-sha384-192
MKey: vno 1
Attributes:
Policy: [none]

kadmin: getprinc -terse systest
systest@BLEEP.COM   3    86400     604800    1
785926535 753241234 785900000
tlyu/admin@BLEEP.COM     786100034 0    0
kadmin:
```

list_principals

list_principals [*expression*]

Retrieves all or some principal names. *expression* is a shell-style glob expression that can contain the wild-card characters ?, *, and []. All principal names matching the expression are printed. If no expression is provided, all principal names are printed. If the expression does not contain an @ character, an @ character followed by the local realm is appended to the expression.

This command requires the **list** privilege.

Alias: **listprincs**, **get_principals**, **getprincs**

Example:

```
kadmin:  listprincs test*
test3@SECURE-TEST.OV.COM
test2@SECURE-TEST.OV.COM
test1@SECURE-TEST.OV.COM
testuser@SECURE-TEST.OV.COM
kadmin:
```

get_strings

get_strings *principal*

Displays string attributes on *principal*.

This command requires the **inquire** privilege.

Alias: **getstrs**

set_string

set_string *principal name value*

Sets a string attribute on *principal*. String attributes are used to supply per-principal configuration to the KDC and some KDC plugin modules. The following string attribute names are recognized by the KDC:

require_auth
: Specifies an authentication indicator which is required to authenticate to the principal as a service. Multiple indicators can be specified, separated by spaces; in this case any of the specified indicators will be accepted. (New in release 1.14.)

session_enctypes
: Specifies the encryption types supported for session keys when the principal is authenticated to as a server. See *Encryption types* in *kdc.conf* for a list of the accepted values.

otp
: Enables One Time Passwords (OTP) preauthentication for a client *principal*. The *value* is a JSON string representing an array of objects, each having optional `type` and `username` fields.

pkinit_cert_match
: Specifies a matching expression that defines the certificate attributes required for the client certificate used by the principal during PKINIT authentication. The matching expression is in the same format as those used by the **pkinit_cert_match** option in *krb5.conf*. (New in release 1.16.)

pac_privsvr_enctype
: Forces the encryption type of the PAC KDC checksum buffers to the specified encryption type for tickets issued to this server, by deriving a key from the local krbtgt key if it is of a different encryption type. It may be necessary to set this value to "aes256-sha1" on the cross-realm krbtgt entry for an Active Directory realm when using aes-sha2 keys on the local krbtgt entry.

This command requires the **modify** privilege.

Alias: **setstr**

Example:

```
set_string host/foo.mit.edu session_enctypes aes128-cts
set_string user@FOO.COM otp "[{""type"":""hotp"",""username"":""al""}]"
```

del_string

del_string *principal key*

Deletes a string attribute from *principal*.

This command requires the **delete** privilege.

Alias: **delstr**

add_policy

add_policy [*options*] *policy*

Adds a password policy named *policy* to the database.

This command requires the **add** privilege.

Alias: **addpol**

The following options are available:

-maxlife ***time***
: (duration or getdate string) Sets the maximum lifetime of a password.

-minlife ***time***
: (duration or getdate string) Sets the minimum lifetime of a password.

-minlength ***length***
: Sets the minimum length of a password.

-minclasses ***number***
: Sets the minimum number of character classes required in a password. The five character classes are lower case, upper case, numbers, punctuation, and whitespace/unprintable characters.

-history ***number***
: Sets the number of past keys kept for a principal. This option is not supported with the LDAP KDC database module.

-maxfailure ***maxnumber***
: Sets the number of authentication failures before the principal is locked. Authentication failures are only tracked for principals which require preauthentication. The counter of failed attempts resets to 0 after a successful attempt to authenticate. A *maxnumber* value of 0 (the default) disables lockout.

-failurecountinterval ***failuretime***
(duration or getdate string) Sets the allowable time between authentication failures. If an authentication failure happens after *failuretime* has elapsed since the previous failure, the number of authentication failures is reset to 1. A *failuretime* value of 0 (the default) means forever.

-lockoutduration ***lockouttime***
(duration or getdate string) Sets the duration for which the principal is locked from authenticating if too many authentication failures occur without the specified failure count interval elapsing. A duration of 0 (the default) means the principal remains locked out until it is administratively unlocked with `modprinc -unlock`.

-allowedkeysalts
Specifies the key/salt tuples supported for long-term keys when setting or changing a principal's password/keys. See *Keysalt lists* in *kdc.conf* for a list of the accepted values, but note that key/salt tuples must be separated with commas (','). To clear the allowed key/salt policy use a value of '-'.

Example:

```
kadmin: add_policy -maxlife "2 days" -minlength 5 guests
kadmin:
```

modify_policy

modify_policy [*options*] *policy*

Modifies the password policy named *policy*. Options are as described for **add_policy**.

This command requires the **modify** privilege.

Alias: **modpol**

delete_policy

delete_policy [**-force**] *policy*

Deletes the password policy named *policy*. Prompts for confirmation before deletion. The command will fail if the policy is in use by any principals.

This command requires the **delete** privilege.

Alias: **delpol**

Example:

```
kadmin: del_policy guests
Are you sure you want to delete the policy "guests"?
(yes/no): yes
kadmin:
```

get_policy

get_policy [**-terse**] *policy*

Displays the values of the password policy named *policy*. With the **-terse** flag, outputs the fields as quoted strings separated by tabs.

This command requires the **inquire** privilege.

Alias: **getpol**

Examples:

```
kadmin: get_policy admin
Policy: admin
Maximum password life: 180 days 00:00:00
Minimum password life: 00:00:00
Minimum password length: 6
Minimum number of password character classes: 2
Number of old keys kept: 5
Reference count: 17

kadmin: get_policy -terse admin
admin     15552000  0    6    2    5    17
kadmin:
```

The "Reference count" is the number of principals using that policy. With the LDAP KDC database module, the reference count field is not meaningful.

list_policies

list_policies [*expression*]

Retrieves all or some policy names. *expression* is a shell-style glob expression that can contain the wild-card characters `?`, `*`, and `[]`. All policy names matching the expression are printed. If no expression is provided, all existing policy names are printed.

This command requires the **list** privilege.

Aliases: **listpols**, **get_policies**, **getpols**.

Examples:

```
kadmin:  listpols
test-pol
dict-only
once-a-min
test-pol-nopw

kadmin:  listpols t*
test-pol
test-pol-nopw
kadmin:
```

ktadd

ktadd [options] *principal*
ktadd [options] **-glob** *princ-exp*

Adds a *principal*, or all principals matching *princ-exp*, to a keytab file. Each principal's keys are randomized in the process. The rules for *princ-exp* are described in the **list_principals** command.

This command requires the **inquire** and **changepw** privileges. With the **-glob** form, it also requires the **list** privilege.

The options are:

-k[eytab] ***keytab***
Use *keytab* as the keytab file. Otherwise, the default keytab is used.

-e ***enc*:*salt*,...**
Uses the specified keysalt list for setting the new keys of the principal. See *Keysalt lists* in *kdc.conf* for a list of possible values.

-q
Display less verbose information.

-norandkey
Do not randomize the keys. The keys and their version numbers stay unchanged. This option cannot be specified in combination with the **-e** option.

An entry for each of the principal's unique encryption types is added, ignoring multiple keys with the same encryption type but different salt types.

Alias: **xst**

Example:

```
kadmin: ktadd -k /tmp/foo-new-keytab host/foo.mit.edu
Entry for principal host/foo.mit.edu@ATHENA.MIT.EDU with kvno 3,
     encryption type aes256-cts-hmac-sha1-96 added to keytab
     FILE:/tmp/foo-new-keytab
kadmin:
```

ktremove

ktremove [options] *principal* [*kvno* | *all* | *old*]

Removes entries for the specified *principal* from a keytab. Requires no permissions, since this does not require database access.

If the string "all" is specified, all entries for that principal are removed; if the string "old" is specified, all entries for that principal except those with the highest kvno are removed. Otherwise, the value specified is parsed as an integer, and all entries whose kvno match that integer are removed.

The options are:

-k[eytab] ***keytab***
Use *keytab* as the keytab file. Otherwise, the default keytab is used.

-q
Display less verbose information.

Alias: **ktrem**

Example:

```
kadmin: ktremove kadmin/admin all
Entry for principal kadmin/admin with kvno 3 removed from keytab
     FILE:/etc/krb5.keytab
kadmin:
```

lock

Lock database exclusively. Use with extreme caution! This command only works with the DB2 KDC database module.

unlock

Release the exclusive database lock.

list_requests

Lists available for kadmin requests.

Aliases: **lr**, **?**

quit

Exit program. If the database was locked, the lock is released.

Aliases: **exit**, **q**

19.1.6 HISTORY

The kadmin program was originally written by Tom Yu at MIT, as an interface to the OpenVision Kerberos administration program.

19.1.7 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.1.8 SEE ALSO

kpasswd(1), *kadmind*, kerberos(7)

19.2 kadmind

19.2.1 SYNOPSIS

kadmind [**-x** *db_args*] [**-r** *realm*] [**-m**] [**-nofork**] [**-proponly**] [**-port** *port-number*] [**-P** *pid_file*] [**-p** *kdb5_util_path*] [**-K** *kprop_path*] [**-k** *kprop_port*] [**-F** *dump_file*]

19.2.2 DESCRIPTION

kadmind starts the Kerberos administration server. kadmind typically runs on the primary Kerberos server, which stores the KDC database. If the KDC database uses the LDAP module, the administration server and the KDC server need not run on the same machine. kadmind accepts remote requests from programs such as *kadmin* and kpasswd(1) to administer the information in these database.

kadmind requires a number of configuration files to be set up in order for it to work:

kdc.conf
: The KDC configuration file contains configuration information for the KDC and admin servers. kadmind uses settings in this file to locate the Kerberos database, and is also affected by the **acl_file**, **dict_file**, **kadmind_port**, and iprop-related settings.

kadm5.acl
: kadmind's ACL (access control list) tells it which principals are allowed to perform administration actions. The pathname to the ACL file can be specified with the **acl_file** *kdc.conf* variable; by default, it is *LOCALSTATEDIR*`/krb5kdc/kadm5.acl`.

After the server begins running, it puts itself in the background and disassociates itself from its controlling terminal.

kadmind can be configured for incremental database propagation. Incremental propagation allows replica KDC servers to receive principal and policy updates incrementally instead of receiving full dumps of the database. This facility can be enabled in the *kdc.conf* file with the **iprop_enable** option. Incremental propagation requires the principal `kiprop/PRIMARY\@REALM` (where PRIMARY is the primary KDC's canonical host name, and REALM the realm name). In release 1.13, this principal is automatically created and registered into the datebase.

19.2.3 OPTIONS

-r ***realm***
: specifies the realm that kadmind will serve; if it is not specified, the default realm of the host is used.

-m
: causes the master database password to be fetched from the keyboard (before the server puts itself in the background, if not invoked with the **-nofork** option) rather than from a file on disk.

-nofork
: causes the server to remain in the foreground and remain associated to the terminal.

-proponly
: causes the server to only listen and respond to Kerberos replica incremental propagation polling requests. This option can be used to set up a hierarchical propagation topology where a replica KDC provides incremental updates to other Kerberos replicas.

-port ***port-number***
: specifies the port on which the administration server listens for connections. The default port is determined by the **kadmind_port** configuration variable in *kdc.conf*.

-P ***pid_file***
: specifies the file to which the PID of kadmind process should be written after it starts up. This file can be used to identify whether kadmind is still running and to allow init scripts to stop the correct process.

-p ***kdb5_util_path***
: specifies the path to the kdb5_util command to use when dumping the KDB in response to full resync requests when iprop is enabled.

-K ***kprop_path***
: specifies the path to the kprop command to use to send full dumps to replicas in response to full resync requests.

-k ***kprop_port***
: specifies the port by which the kprop process that is spawned by kadmind connects to the replica kpropd, in order to transfer the dump file during an iprop full resync request.

-F ***dump_file***
: specifies the file path to be used for dumping the KDB in response to full resync requests when iprop is enabled.

-x ***db_args***
: specifies database-specific arguments. See *Database Options* in *kadmin* for supported arguments.

19.2.4 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

As of release 1.22, kadmind supports systemd socket activation via the LISTEN_PID and LISTEN_FDS environment variables. Sockets provided by the caller must correspond to configured listener addresses (via the **kadmind_listen** or **kpasswd_listen** variables or equivalents) or they will be ignored. Any configured listener addresses that do not correspond to caller-provided sockets will be ignored if socket activation is used.

19.2.5 SEE ALSO

kpasswd(1), *kadmin*, *kdb5_util*, *kdb5_ldap_util*, *kadm5.acl*, kerberos(7)

19.3 kdb5_util

19.3.1 SYNOPSIS

kdb5_util [**-r** *realm*] [**-d** *dbname*] [**-k** *mkeytype*] [**-kv** *mkeyVNO*] [**-M** *mkeyname*] [**-m**] [**-sf** *stashfilename*] [**-P** *password*] [**-x** *db_args*] *command* [*command_options*]

19.3.2 DESCRIPTION

kdb5_util allows an administrator to perform maintenance procedures on the KDC database. Databases can be created, destroyed, and dumped to or loaded from ASCII files. kdb5_util can create a Kerberos master key stash file or perform live rollover of the master key.

When kdb5_util is run, it attempts to acquire the master key and open the database. However, execution continues regardless of whether or not kdb5_util successfully opens the database, because the database may not exist yet or the stash file may be corrupt.

Note that some KDC database modules may not support all kdb5_util commands.

19.3.3 COMMAND-LINE OPTIONS

-r ***realm***
specifies the Kerberos realm of the database.

-d ***dbname***
specifies the name under which the principal database is stored; by default the database is that listed in *kdc.conf*. The password policy database and lock files are also derived from this value.

-k ***mkeytype***
specifies the key type of the master key in the database. The default is given by the **master_key_type** variable in *kdc.conf*.

-kv ***mkeyVNO***
Specifies the version number of the master key in the database; the default is 1. Note that 0 is not allowed.

-M ***mkeyname***
principal name for the master key in the database. If not specified, the name is determined by the **master_key_name** variable in *kdc.conf*.

-m
specifies that the master database password should be read from the keyboard rather than fetched from a file on disk.

-sf ***stash_file***
specifies the stash filename of the master database password. If not specified, the filename is determined by the **key_stash_file** variable in *kdc.conf*.

-P ***password***
specifies the master database password. Using this option may expose the password to other users on the system via the process list.

-x ***db_args***
specifies database-specific options. See *kadmin* for supported options.

19.3.4 COMMANDS

create

create [**-s**]

Creates a new database. If the **-s** option is specified, the stash file is also created. This command fails if the database already exists. If the command is successful, the database is opened just as if it had already existed when the program was first run.

destroy

destroy [**-f**]

Destroys the database, first overwriting the disk sectors and then unlinking the files, after prompting the user for confirmation. With the **-f** argument, does not prompt the user.

stash

stash [**-f** *keyfile*]

Stores the master principal's keys in a stash file. The **-f** argument can be used to override the *keyfile* specified in *kdc.conf*.

dump

dump [**-b7**|**-r13**|**-r18**] [**-verbose**] [**-mkey_convert**] [**-new_mkey_file** *mkey_file*] [**-rev**] [**-recurse**] [*filename* [*principals*...]]

Dumps the current Kerberos and KADM5 database into an ASCII file. By default, the database is dumped in current format, "kdb5_util load_dump version 7". If filename is not specified, or is the string "-", the dump is sent to standard output. Options:

-b7
: causes the dump to be in the Kerberos 5 Beta 7 format ("kdb5_util load_dump version 4"). This was the dump format produced on releases prior to 1.2.2.

-r13
: causes the dump to be in the Kerberos 5 1.3 format ("kdb5_util load_dump version 5"). This was the dump format produced on releases prior to 1.8.

-r18
: causes the dump to be in the Kerberos 5 1.8 format ("kdb5_util load_dump version 6"). This was the dump format produced on releases prior to 1.11.

-verbose
: causes the name of each principal and policy to be printed as it is dumped.

-mkey_convert
: prompts for a new master key. This new master key will be used to re-encrypt principal key data in the dumpfile. The principal keys themselves will not be changed.

-new_mkey_file ***mkey_file***
: the filename of a stash file. The master key in this stash file will be used to re-encrypt the key data in the dumpfile. The key data in the database will not be changed.

-rev
: dumps in reverse order. This may recover principals that do not dump normally, in cases where database corruption has occurred.

-recurse
: causes the dump to walk the database recursively (btree only). This may recover principals that do not dump normally, in cases where database corruption has occurred. In cases of such corruption, this option will probably retrieve more principals than the **-rev** option will.

  Changed in version 1.15: Release 1.15 restored the functionality of the **-recurse** option.

  Changed in version 1.5: The **-recurse** option ceased working until release 1.15, doing a normal dump instead of a recursive traversal.

load

load [**-b7**|**-r13**|**-r18**] [**-hash**] [**-verbose**] [**-update**] *filename*

Loads a database dump from the named file into the named database. If no option is given to determine the format of the dump file, the format is detected automatically and handled as appropriate. Unless the **-update** option is given, **load** creates a new database containing only the data in the dump file, overwriting the contents of any previously existing database. Note that when using the LDAP KDC database module, the **-update** flag is required.

Options:

-b7
: requires the database to be in the Kerberos 5 Beta 7 format ("kdb5_util load_dump version 4"). This was the dump format produced on releases prior to 1.2.2.

-r13
: requires the database to be in Kerberos 5 1.3 format ("kdb5_util load_dump version 5"). This was the dump format produced on releases prior to 1.8.

-r18
: requires the database to be in Kerberos 5 1.8 format ("kdb5_util load_dump version 6"). This was the dump format produced on releases prior to 1.11.

-hash
: stores the database in hash format, if using the DB2 database type. If this option is not specified, the database will be stored in btree format. This option is not recommended, as databases stored in hash format are known to corrupt data and lose principals.

-verbose
: causes the name of each principal and policy to be printed as it is dumped.

-update
: records from the dump file are added to or updated in the existing database. Otherwise, a new database is created containing only what is in the dump file and the old one destroyed upon successful completion.

ark

ark [**-e** *enc*:*salt*,...] *principal*

Adds new random keys to *principal* at the next available key version number. Keys for the current highest key version number will be preserved. The **-e** option specifies the list of encryption and salt types to be used for the new keys.

add_mkey

add_mkey [**-e** *etype*] [**-s**]

Adds a new master key to the master key principal, but does not mark it as active. Existing master keys will remain. The **-e** option specifies the encryption type of the new master key; see *Encryption types* in *kdc.conf* for a list of possible values. The **-s** option stashes the new master key in the stash file, which will be created if it doesn't already exist.

After a new master key is added, it should be propagated to replica servers via a manual or periodic invocation of *kprop*. Then, the stash files on the replica servers should be updated with the kdb5_util **stash** command. Once those steps are complete, the key is ready to be marked active with the kdb5_util **use_mkey** command.

use_mkey

use_mkey *mkeyVNO* [*time*]

Sets the activation time of the master key specified by *mkeyVNO*. Once a master key becomes active, it will be used to encrypt newly created principal keys. If no *time* argument is given, the current time is used, causing the specified master key version to become active immediately. The format for *time* is getdate string.

After a new master key becomes active, the kdb5_util **update_princ_encryption** command can be used to update all principal keys to be encrypted in the new master key.

list_mkeys

list_mkeys

List all master keys, from most recent to earliest, in the master key principal. The output will show the kvno, enctype, and salt type for each mkey, similar to the output of *kadmin* **getprinc**. A * following an mkey denotes the currently active master key.

purge_mkeys

purge_mkeys [**-f**] [**-n**] [**-v**]

Delete master keys from the master key principal that are not used to protect any principals. This command can be used to remove old master keys all principal keys are protected by a newer master key.

-f
: does not prompt for confirmation.

-n
: performs a dry run, showing master keys that would be purged, but not actually purging any keys.

-v
: gives more verbose output.

update_princ_encryption

update_princ_encryption [**-f**] [**-n**] [**-v**] [*princ-pattern*]

Update all principal records (or only those matching the *princ-pattern* glob pattern) to re-encrypt the key data using the active database master key, if they are encrypted using a different version, and give a count at the end of the number of principals updated. If the **-f** option is not given, ask for confirmation before starting to make changes. The **-v** option causes each principal processed to be listed, with an indication as to whether it needed updating or not. The **-n** option performs a dry run, only showing the actions which would have been taken.

tabdump

tabdump [**-H**] [**-c**] [**-e**] [**-n**] [**-o** *outfile*] *dumptype*

Dump selected fields of the database in a tabular format suitable for reporting (e.g., using traditional Unix text processing tools) or importing into relational databases. The data format is tab-separated (default), or optionally comma-separated (CSV), with a fixed number of columns. The output begins with a header line containing field names, unless suppression is requested using the **-H** option.

The *dumptype* parameter specifies the name of an output table (see below).

Options:

-H
suppress writing the field names in a header line

-c
use comma separated values (CSV) format, with minimal quoting, instead of the default tab-separated (unquoted, unescaped) format

-e
write empty hexadecimal string fields as empty fields instead of as "-1".

-n
produce numeric output for fields that normally have symbolic output, such as enctypes and flag names. Also requests output of time stamps as decimal POSIX time_t values.

-o ***outfile***
write the dump to the specified output file instead of to standard output

Dump types:

alias
principal alias information

aliasname
the name of the alias

targetname
the target of the alias

keydata
principal encryption key information, including actual key data (which is still encrypted in the master key)

name
principal name

keyindex
index of this key in the principal's key list

kvno
key version number

enctype
encryption type

key
key data as a hexadecimal string

salttype
salt type

salt
salt data as a hexadecimal string

keyinfo
principal encryption key information (as in **keydata** above), excluding actual key data

princ_flags
principal boolean attributes. Flag names print as hexadecimal numbers if the **-n** option is specified, and all flag positions are printed regardless of whether or not they are set. If **-n** is not specified, print all known flag names for each principal, but only print hexadecimal flag names if the corresponding flag is set.

name
principal name

flag
flag name

value
boolean value (0 for clear, or 1 for set)

princ_lockout
state information used for tracking repeated password failures

name
principal name

last_success
time stamp of most recent successful authentication

last_failed
time stamp of most recent failed authentication

fail_count
count of failed attempts

princ_meta
principal metadata

name
principal name

modby
name of last principal to modify this principal

modtime
timestamp of last modification

lastpwd
timestamp of last password change

policy
policy object name

mkvno
key version number of the master key that encrypts this principal's key data

hist_kvno
key version number of the history key that encrypts the key history data for this principal

princ_stringattrs
string attributes (key/value pairs)

name
principal name

key
attribute name

value
attribute value

princ_tktpolicy
per-principal ticket policy data, including maximum ticket lifetimes

name
principal name

expiration
principal expiration date

pw_expiration
password expiration date

max_life
maximum ticket lifetime

max_renew_life
maximum renewable ticket lifetime

Examples:

```
$ kdb5_util tabdump -o keyinfo.txt keyinfo
$ cat keyinfo.txt
name        keyindex        kvno    enctype salttype        salt
K/M@EXAMPLE.COM     0       1       aes256-cts-hmac-sha384-192      normal  -1
foo@EXAMPLE.COM     0       1       aes128-cts-hmac-sha1-96 normal  -1
bar@EXAMPLE.COM     0       1       aes128-cts-hmac-sha1-96 normal  -1
$ sqlite3
sqlite> .mode tabs
sqlite> .import keyinfo.txt keyinfo
sqlite> select * from keyinfo where enctype like 'aes256-%';
K/M@EXAMPLE.COM     1       1       aes256-cts-hmac-sha384-192      normal  -1
sqlite> .quit
$ awk -F'\t' '$4 ~ /aes256-/ { print }' keyinfo.txt
K/M@EXAMPLE.COM     1       1       aes256-cts-hmac-sha384-192      normal  -1
```

19.3.5 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.3.6 SEE ALSO

kadmin, kerberos(7)

19.4 kdb5_ldap_util

19.4.1 SYNOPSIS

kdb5_ldap_util [**-D** *user_dn* [**-w** *passwd*]] [**-H** *ldapuri*] **command** [*command_options*]

19.4.2 DESCRIPTION

kdb5_ldap_util allows an administrator to manage realms, Kerberos services and ticket policies.

19.4.3 COMMAND-LINE OPTIONS

-r ***realm***
Specifies the realm to be operated on.

-D ***user_dn***
Specifies the Distinguished Name (DN) of the user who has sufficient rights to perform the operation on the LDAP server.

-w ***passwd***
Specifies the password of *user_dn*. This option is not recommended.

-H ***ldapuri***
Specifies the URI of the LDAP server.

By default, kdb5_ldap_util operates on the default realm (as specified in *krb5.conf*) and connects and authenticates to the LDAP server in the same manner as :ref:kadmind(8)` would given the parameters in *[dbdefaults]* in *kdc.conf*.

19.4.4 COMMANDS

create

create [**-subtrees** *subtree_dn_list*] [**-sscope** *search_scope*] [**-containerref** *container_reference_dn*] [**-k** *mkeytype*] [**-kv** *mkeyVNO*] [**-M** *mkeyname*] [**-m|-P** *password*|**-sf** *stashfilename*] [**-s**] [**-maxtktlife** *max_ticket_life*] [**-maxrenewlife** *max_renewable_ticket_life*] [*ticket_flags*]

Creates realm in directory. Options:

-subtrees ***subtree_dn_list***
Specifies the list of subtrees containing the principals of a realm. The list contains the DNs of the subtree objects separated by colon (**:**).

-sscope ***search_scope***
Specifies the scope for searching the principals under the subtree. The possible values are 1 or one (one level), 2 or sub (subtrees).

-containerref ***container_reference_dn***
Specifies the DN of the container object in which the principals of a realm will be created. If the container reference is not configured for a realm, the principals will be created in the realm container.

-k ***mkeytype***
Specifies the key type of the master key in the database. The default is given by the **master_key_type** variable in *kdc.conf*.

-kv ***mkeyVNO***
Specifies the version number of the master key in the database; the default is 1. Note that 0 is not allowed.

-M ***mkeyname***
Specifies the principal name for the master key in the database. If not specified, the name is determined by the **master_key_name** variable in *kdc.conf*.

-m
Specifies that the master database password should be read from the TTY rather than fetched from a file on the disk.

-P *password*
Specifies the master database password. This option is not recommended.

-sf *stashfilename*
Specifies the stash file of the master database password.

-s
Specifies that the stash file is to be created.

-maxtktlife *max_ticket_life*
(getdate string) Specifies maximum ticket life for principals in this realm.

-maxrenewlife *max_renewable_ticket_life*
(getdate string) Specifies maximum renewable life of tickets for principals in this realm.

ticket_flags
Specifies global ticket flags for the realm. Allowable flags are documented in the description of the **add_principal** command in *kadmin*.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU create -subtrees o=org -sscope SUB
Password for "cn=admin,o=org":
Initializing database for realm 'ATHENA.MIT.EDU'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:
```

modify

modify [**-subtrees** *subtree_dn_list*] [**-sscope** *search_scope*] [**-containerref** *container_reference_dn*] [**-maxtktlife** *max_ticket_life*] [**-maxrenewlife** *max_renewable_ticket_life*] [*ticket_flags*]

Modifies the attributes of a realm. Options:

-subtrees *subtree_dn_list*
Specifies the list of subtrees containing the principals of a realm. The list contains the DNs of the subtree objects separated by colon (:). This list replaces the existing list.

-sscope *search_scope*
Specifies the scope for searching the principals under the subtrees. The possible values are 1 or one (one level), 2 or sub (subtrees).

-containerref *container_reference_dn* **Specifies the DN of the**
container object in which the principals of a realm will be created.

-maxtktlife *max_ticket_life*
(getdate string) Specifies maximum ticket life for principals in this realm.

-maxrenewlife *max_renewable_ticket_life*
(getdate string) Specifies maximum renewable life of tickets for principals in this realm.

ticket_flags
Specifies global ticket flags for the realm. Allowable flags are documented in the description of the **add_principal** command in *kadmin*.

Example:

```
shell% kdb5_ldap_util -r ATHENA.MIT.EDU -D cn=admin,o=org -H
    ldaps://ldap-server1.mit.edu modify +requires_preauth
Password for "cn=admin,o=org":
shell%
```

view

view

Displays the attributes of a realm.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU view
Password for "cn=admin,o=org":
Realm Name: ATHENA.MIT.EDU
Subtree: ou=users,o=org
Subtree: ou=servers,o=org
SearchScope: ONE
Maximum ticket life: 0 days 01:00:00
Maximum renewable life: 0 days 10:00:00
Ticket flags: DISALLOW_FORWARDABLE REQUIRES_PWCHANGE
```

destroy

destroy [**-f**]

Destroys an existing realm. Options:

-f
: If specified, will not prompt the user for confirmation.

Example:

```
shell% kdb5_ldap_util -r ATHENA.MIT.EDU -D cn=admin,o=org -H
    ldaps://ldap-server1.mit.edu destroy
Password for "cn=admin,o=org":
Deleting KDC database of 'ATHENA.MIT.EDU', are you sure?
(type 'yes' to confirm)? yes
OK, deleting database of 'ATHENA.MIT.EDU'...
shell%
```

list

list

Lists the names of realms under the container.

Example:

```
shell% kdb5_ldap_util -D cn=admin,o=org -H
    ldaps://ldap-server1.mit.edu list
Password for "cn=admin,o=org":
ATHENA.MIT.EDU
OPENLDAP.MIT.EDU
MEDIA-LAB.MIT.EDU
shell%
```

stashsrvpw

stashsrvpw [**-f** *filename*] *name*

Allows an administrator to store the password for service object in a file so that KDC and Administration server can use it to authenticate to the LDAP server. Options:

-f ***filename***
Specifies the complete path of the service password file. By default, `/usr/local/var/service_passwd` is used.

name
Specifies the name of the object whose password is to be stored. If *krb5kdc* or *kadmind* are configured for simple binding, this should be the distinguished name it will use as given by the **ldap_kdc_dn** or **ldap_kadmind_dn** variable in *kdc.conf*. If the KDC or kadmind is configured for SASL binding, this should be the authentication name it will use as given by the **ldap_kdc_sasl_authcid** or **ldap_kadmind_sasl_authcid** variable.

Example:

```
kdb5_ldap_util stashsrvpw -f /home/andrew/conf_keyfile
    cn=service-kdc,o=org
Password for "cn=service-kdc,o=org":
Re-enter password for "cn=service-kdc,o=org":
```

create_policy

create_policy [**-maxtktlife** *max_ticket_life*] [**-maxrenewlife** *max_renewable_ticket_life*] [*ticket_flags*] *policy_name*

Creates a ticket policy in the directory. Options:

-maxtktlife ***max_ticket_life***
(getdate string) Specifies maximum ticket life for principals.

-maxrenewlife ***max_renewable_ticket_life***
(getdate string) Specifies maximum renewable life of tickets for principals.

ticket_flags
Specifies the ticket flags. If this option is not specified, by default, no restriction will be set by the policy. Allowable flags are documented in the description of the **add_principal** command in *kadmin*.

policy_name
Specifies the name of the ticket policy.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU create_policy -maxtktlife "1 day"
    -maxrenewlife "1 week" -allow_postdated +needchange
    -allow_forwardable tktpolicy
Password for "cn=admin,o=org":
```

modify_policy

modify_policy [**-maxtktlife** *max_ticket_life*] [**-maxrenewlife** *max_renewable_ticket_life*] [*ticket_flags*] *policy_name*

Modifies the attributes of a ticket policy. Options are same as for **create_policy**.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H
    ldaps://ldap-server1.mit.edu -r ATHENA.MIT.EDU modify_policy
    -maxtktlife "60 minutes" -maxrenewlife "10 hours"
    +allow_postdated -requires_preauth tktpolicy
Password for "cn=admin,o=org":
```

view_policy

view_policy *policy_name*

Displays the attributes of the named ticket policy.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU view_policy tktpolicy
Password for "cn=admin,o=org":
Ticket policy: tktpolicy
Maximum ticket life: 0 days 01:00:00
Maximum renewable life: 0 days 10:00:00
Ticket flags: DISALLOW_FORWARDABLE REQUIRES_PWCHANGE
```

destroy_policy

destroy_policy [**-force**] *policy_name*

Destroys an existing ticket policy. Options:

-force
: Forces the deletion of the policy object. If not specified, the user will be prompted for confirmation before deleting the policy.

policy_name
: Specifies the name of the ticket policy.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU destroy_policy tktpolicy
Password for "cn=admin,o=org":
This will delete the policy object 'tktpolicy', are you sure?
(type 'yes' to confirm)? yes
** policy object 'tktpolicy' deleted.
```

list_policy

list_policy

Lists ticket policies.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
    -r ATHENA.MIT.EDU list_policy
Password for "cn=admin,o=org":
tktpolicy
tmppolicy
userpolicy
```

19.4.5 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.4.6 SEE ALSO

kadmin, kerberos(7)

19.5 krb5kdc

19.5.1 SYNOPSIS

krb5kdc [**-x** *db_args*] [**-d** *dbname*] [**-k** *keytype*] [**-M** *mkeyname*] [**-p** *portnum*] [**-m**] [**-r** *realm*] [**-n**] [**-w** *numworkers*] [**-P** *pid_file*] [**-T** *time_offset*]

19.5.2 DESCRIPTION

krb5kdc is the Kerberos version 5 Authentication Service and Key Distribution Center (AS/KDC).

19.5.3 OPTIONS

The **-r** *realm* option specifies the realm for which the server should provide service. This option may be specified multiple times to serve multiple realms. If no **-r** option is given, the default realm (as specified in *krb5.conf*) will be served.

The **-d** *dbname* option specifies the name under which the principal database can be found. This option does not apply to the LDAP database.

The **-k** *keytype* option specifies the key type of the master key to be entered manually as a password when **-m** is given; the default is `aes256-cts-hmac-sha1-96`.

The **-M** *mkeyname* option specifies the principal name for the master key in the database (usually `K/M` in the KDC's realm).

The **-m** option specifies that the master database password should be fetched from the keyboard rather than from a stash file.

The **-n** option specifies that the KDC does not put itself in the background and does not disassociate itself from the terminal.

The **-P** *pid_file* option tells the KDC to write its PID into *pid_file* after it starts up. This can be used to identify whether the KDC is still running and to allow init scripts to stop the correct process.

The **-p** *portnum* option specifies the default UDP and TCP port numbers which the KDC should listen on for Kerberos version 5 requests, as a comma-separated list. This value overrides the port numbers specified in the *[kdcdefaults]* section of *kdc.conf*, but may be overridden by realm-specific values. If no value is given from any source, the default port is 88.

The **-w** *numworkers* option tells the KDC to fork *numworkers* processes to listen to the KDC ports and process requests in parallel. The top level KDC process (whose pid is recorded in the pid file if the **-P** option is also given) acts as a supervisor. The supervisor will relay SIGHUP signals to the worker subprocesses, and will terminate the worker subprocess if the it is itself terminated or if any other worker process exits.

The **-x** *db_args* option specifies database-specific arguments. See *Database Options* in *kadmin* for supported arguments.

The **-T** *offset* option specifies a time offset, in seconds, which the KDC will operate under. It is intended only for testing purposes.

19.5.4 EXAMPLE

The KDC may service requests for multiple realms (maximum 32 realms). The realms are listed on the command line. Per-realm options that can be specified on the command line pertain for each realm that follows it and are superseded by subsequent definitions of the same option.

For example:

```
krb5kdc -p 2001 -r REALM1 -p 2002 -r REALM2 -r REALM3
```

specifies that the KDC listen on port 2001 for REALM1 and on port 2002 for REALM2 and REALM3. Additionally, per-realm parameters may be specified in the *kdc.conf* file. The location of this file may be specified by the **KRB5_KDC_PROFILE** environment variable. Per-realm parameters specified in this file take precedence over options specified on the command line. See the *kdc.conf* description for further details.

19.5.5 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

As of release 1.22, krb5kdc supports systemd socket activation via the LISTEN_PID and LISTEN_FDS environment variables. Sockets provided by the caller must correspond to configured listener addresses (via the **kdc_listen** variable or equivalent) or they will be ignored. Any configured listener addresses that do not correspond to caller-provided sockets will be ignored if socket activation is used.

19.5.6 SEE ALSO

kdb5_util, *kdc.conf*, *krb5.conf*, *kdb5_ldap_util*, kerberos(7)

19.6 kprop

19.6.1 SYNOPSIS

kprop [**-r** *realm*] [**-f** *file*] [**-d**] [**-P** *port*] [**-s** *keytab*] *replica_host*

19.6.2 DESCRIPTION

kprop is used to securely propagate a Kerberos V5 database dump file from the primary Kerberos server to a replica Kerberos server, which is specified by *replica_host*. The dump file must be created by *kdb5_util*.

19.6.3 OPTIONS

-r *realm*
: Specifies the realm of the primary server.

-f *file*
: Specifies the filename where the dumped principal database file is to be found; by default the dumped database file is normally *LOCALSTATEDIR*`/krb5kdc/replica_datatrans`.

-P *port*
: Specifies the port to use to contact the *kpropd* server on the remote host.

-d
: Prints debugging information.

-s *keytab*
: Specifies the location of the keytab file.

19.6.4 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.6.5 SEE ALSO

kpropd, *kdb5_util*, *krb5kdc*, kerberos(7)

19.7 kpropd

19.7.1 SYNOPSIS

kpropd [**-r** *realm*] [**-A** *admin_server*] [**-a** *acl_file*] [**-f** *replica_dumpfile*] [**-F** *principal_database*] [**-p** *kdb5_util_prog*] [**-P** *port*] [**–pid-file**=*pid_file*] [**-D**] [**-d**] [**-s** *keytab_file*]

19.7.2 DESCRIPTION

The *kpropd* command runs on the replica KDC server. It listens for update requests made by the *kprop* program. If incremental propagation is enabled, it periodically requests incremental updates from the primary KDC.

When the replica receives a kprop request from the primary, kpropd accepts the dumped KDC database and places it in a file, and then runs *kdb5_util* to load the dumped database into the active database which is used by *krb5kdc*. This allows the primary Kerberos server to use *kprop* to propagate its database to the replica servers. Upon a successful download of the KDC database file, the replica Kerberos server will have an up-to-date KDC database.

Where incremental propagation is not used, kpropd is commonly invoked out of inetd(8) as a nowait service. This is done by adding a line to the `/etc/inetd.conf` file which looks like this:

```
kprop  stream  tcp  nowait  root  /usr/local/sbin/kpropd  kpropd
```

kpropd can also run as a standalone daemon, backgrounding itself and waiting for connections on port 754 (or the port specified with the **-P** option if given). Standalone mode is required for incremental propagation. Starting in release 1.11, kpropd automatically detects whether it was run from inetd and runs in standalone mode if it is not. Prior to release 1.11, the **-S** option is required to run kpropd in standalone mode; this option is now accepted for backward compatibility but does nothing.

Incremental propagation may be enabled with the **iprop_enable** variable in *kdc.conf*. If incremental propagation is enabled, the replica periodically polls the primary KDC for updates, at an interval determined by the **iprop_replica_poll** variable. If the replica receives updates, kpropd updates its log file with any updates from the primary. *kproplog* can be used to view a summary of the update entry log on the replica KDC. If incremental propagation is enabled, the principal `kiprop/replicahostname@REALM` (where *replicahostname* is the name of the replica KDC host, and *REALM* is the name of the Kerberos realm) must be present in the replica's keytab file.

kproplog can be used to force full replication when iprop is enabled.

19.7.3 OPTIONS

-r *realm*
Specifies the realm of the primary server.

-A *admin_server*
Specifies the server to be contacted for incremental updates; by default, the primary admin server is contacted.

-f *file*
Specifies the filename where the dumped principal database file is to be stored; by default the dumped database file is *LOCALSTATEDIR*`/krb5kdc/from_master`.

-F *kerberos_db*
Path to the Kerberos database file, if not the default.

-p
Allows the user to specify the pathname to the *kdb5_util* program; by default the pathname used is *SBINDIR*`/kdb5_util`.

-D
In this mode, kpropd will not detach itself from the current job and run in the background. Instead, it will run in the foreground.

-d
Turn on debug mode. kpropd will print out debugging messages during the database propogation and will run in the foreground (implies **-D**).

-P
Allow for an alternate port number for kpropd to listen on. This is only useful in combination with the **-S** option.

-a *acl_file*
Allows the user to specify the path to the kpropd.acl file; by default the path used is *LOCALSTATEDIR*`/krb5kdc/kpropd.acl`.

–pid-file=*pid_file*
In standalone mode, write the process ID of the daemon into *pid_file*.

-s *keytab_file*
Path to a keytab to use for acquiring acceptor credentials.

-x *db_args*
Database-specific arguments. See *Database Options* in *kadmin* for supported arguments.

19.7.4 FILES

kpropd.acl
Access file for kpropd; the default location is `/usr/local/var/krb5kdc/kpropd.acl`. Each entry is a line containing the principal of a host from which the local machine will allow Kerberos database propagation via *kprop*.

19.7.5 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.7.6 SEE ALSO

kprop, *kdb5_util*, *krb5kdc*, kerberos(7), inetd(8)

19.8 kproplog

19.8.1 SYNOPSIS

kproplog [**-h**] [**-e** *num*] [-v] **kproplog** [-R]

19.8.2 DESCRIPTION

The kproplog command displays the contents of the KDC database update log to standard output. It can be used to keep track of incremental updates to the principal database. The update log file contains the update log maintained by the *kadmind* process on the primary KDC server and the *kpropd* process on the replica KDC servers. When updates occur, they are logged to this file. Subsequently any KDC replica configured for incremental updates will request the current data from the primary KDC and update their log file with any updates returned.

The kproplog command requires read access to the update log file. It will display update entries only for the KDC it runs on.

If no options are specified, kproplog displays a summary of the update log. If invoked on the primary, kproplog also displays all of the update entries. If invoked on a replica KDC server, kproplog displays only a summary of the updates, which includes the serial number of the last update received and the associated time stamp of the last update.

19.8.3 OPTIONS

-R
: Reset the update log. This forces full resynchronization. If used on a replica then that replica will request a full resync. If used on the primary then all replicas will request full resyncs.

-h
: Display a summary of the update log. This information includes the database version number, state of the database, the number of updates in the log, the time stamp of the first and last update, and the version number of the first and last update entry.

-e *num*
: Display the last *num* update entries in the log. This is useful when debugging synchronization between KDC servers.

-v
: Display individual attributes per update. An example of the output generated for one entry:

```
Update Entry
   Update serial # : 4
   Update operation : Add
   Update principal : test@EXAMPLE.COM
```

(continues on next page)

(continued from previous page)

```
Update size : 424
Update committed : True
Update time stamp : Fri Feb 20 23:37:42 2004
Attributes changed : 6
      Principal
      Key data
      Password last changed
      Modifying principal
      Modification time
      TL data
```

19.8.4 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.8.5 SEE ALSO

kpropd, kerberos(7)

19.9 ktutil

19.9.1 SYNOPSIS

ktutil

19.9.2 DESCRIPTION

The ktutil command invokes a command interface from which an administrator can read, write, or edit entries in a keytab. (Kerberos V4 srvtab files are no longer supported.)

19.9.3 COMMANDS

list

list [**-t**] [**-k**] [**-e**]

Displays the current keylist. If **-t**, **-k**, and/or **-e** are specified, also display the timestamp, key contents, or enctype (respectively).

Alias: **l**

read_kt

read_kt *keytab*

Read the Kerberos V5 keytab file *keytab* into the current keylist.

Alias: **rkt**

write_kt

write_kt *keytab*

Write the current keylist into the Kerberos V5 keytab file *keytab*.

Alias: **wkt**

clear_list

clear_list

Clear the current keylist.

Alias: **clear**

delete_entry

delete_entry *slot*

Delete the entry in slot number *slot* from the current keylist.

Alias: **delent**

add_entry

add_entry {**-key**|**-password**} **-p** *principal* **-k** *kvno* [**-e** *enctype*] [**-f**|**-s** *salt*]

Add *principal* to keylist using key or password. If the **-f** flag is specified, salt information will be fetched from the KDC; in this case the **-e** flag may be omitted, or it may be supplied to force a particular enctype. If the **-f** flag is not specified, the **-e** flag must be specified, and the default salt will be used unless overridden with the **-s** option.

Alias: **addent**

list_requests

list_requests

Displays a listing of available commands.

Aliases: **lr**, **?**

quit

quit

Quits ktutil.

Aliases: **exit**, **q**

19.9.4 EXAMPLE

```
ktutil:  add_entry -password -p alice@BLEEP.COM -k 1 -e
    aes128-cts-hmac-sha1-96
Password for alice@BLEEP.COM:
ktutil:  add_entry -password -p alice@BLEEP.COM -k 1 -e
    aes256-cts-hmac-sha1-96
Password for alice@BLEEP.COM:
ktutil:  write_kt alice.keytab
ktutil:
```

19.9.5 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.9.6 SEE ALSO

kadmin, *kdb5_util*, kerberos(7)

19.10 k5srvutil

19.10.1 SYNOPSIS

k5srvutil *operation* [**-i**] [**-f** *filename*] [**-e** *keysalts*]

19.10.2 DESCRIPTION

k5srvutil allows an administrator to list keys currently in a keytab, to obtain new keys for a principal currently in a keytab, or to delete non-current keys from a keytab.

operation must be one of the following:

list
: Lists the keys in a keytab, showing version number and principal name.

change
: Uses the kadmin protocol to update the keys in the Kerberos database to new randomly-generated keys, and updates the keys in the keytab to match. If a key's version number doesn't match the version number stored in the Kerberos server's database, then the operation will fail. If the **-i** flag is given, k5srvutil will prompt for confirmation before changing each key. If the **-k** option is given, the old and new keys will be displayed. Ordinarily, keys will be generated with the default encryption types and key salts. This can be overridden with

the **-e** option. Old keys are retained in the keytab so that existing tickets continue to work, but **delold** should be used after such tickets expire, to prevent attacks against the old keys.

delold
Deletes keys that are not the most recent version from the keytab. This operation should be used some time after a change operation to remove old keys, after existing tickets issued for the service have expired. If the **-i** flag is given, then k5srvutil will prompt for confirmation for each principal.

delete
Deletes particular keys in the keytab, interactively prompting for each key.

In all cases, the default keytab is used unless this is overridden by the **-f** option.

k5srvutil uses the *kadmin* program to edit the keytab in place.

19.10.3 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.10.4 SEE ALSO

kadmin, *ktutil*, kerberos(7)

19.11 sserver

19.11.1 SYNOPSIS

sserver [**-p** *port*] [**-S** *keytab*] [*server_port*]

19.11.2 DESCRIPTION

sserver and sclient(1) are a simple demonstration client/server application. When sclient connects to sserver, it performs a Kerberos authentication, and then sserver returns to sclient the Kerberos principal which was used for the Kerberos authentication. It makes a good test that Kerberos has been successfully installed on a machine.

The service name used by sserver and sclient is sample. Hence, sserver will require that there be a keytab entry for the service `sample/hostname.domain.name@REALM.NAME`. This keytab is generated using the *kadmin* program. The keytab file is usually installed as *DEFKTNAME*.

The **-S** option allows for a different keytab than the default.

sserver is normally invoked out of inetd(8), using a line in `/etc/inetd.conf` that looks like this:

```
sample stream tcp nowait root /usr/local/sbin/sserver sserver
```

Since `sample` is normally not a port defined in `/etc/services`, you will usually have to add a line to `/etc/services` which looks like this:

```
sample          13135/tcp
```

When using sclient, you will first have to have an entry in the Kerberos database, by using *kadmin*, and then you have to get Kerberos tickets, by using kinit(1). Also, if you are running the sclient program on a different host than the sserver

it will be connecting to, be sure that both hosts have an entry in /etc/services for the sample tcp port, and that the same port number is in both files.

When you run sclient you should see something like this:

```
sendauth succeeded, reply is:
reply len 32, contents:
You are nlgilman@JIMI.MIT.EDU
```

19.11.3 COMMON ERROR MESSAGES

1) kinit returns the error:

```
kinit: Client not found in Kerberos database while getting
       initial credentials
```

This means that you didn't create an entry for your username in the Kerberos database.

2) sclient returns the error:

```
unknown service sample/tcp; check /etc/services
```

This means that you don't have an entry in /etc/services for the sample tcp port.

3) sclient returns the error:

```
connect: Connection refused
```

This probably means you didn't edit /etc/inetd.conf correctly, or you didn't restart inetd after editing inetd.conf.

4) sclient returns the error:

```
sclient: Server not found in Kerberos database while using
         sendauth
```

This means that the `sample/hostname@LOCAL.REALM` service was not defined in the Kerberos database; it should be created using *kadmin*, and a keytab file needs to be generated to make the key for that service principal available for sclient.

5) sclient returns the error:

```
sendauth rejected, error reply is:
    "No such file or directory"
```

This probably means sserver couldn't find the keytab file. It was probably not installed in the proper directory.

19.11.4 ENVIRONMENT

See kerberos(7) for a description of Kerberos environment variables.

19.11.5 SEE ALSO

sclient(1), kerberos(7), services(5), inetd(8)

CHAPTER

TWENTY

MIT KERBEROS DEFAULTS

20.1 General defaults

Description	Default	Environment
keytab_definit file	*DEFKTNAME*	**KRB5_KTNAME**
Client keytab_definit file	*DEFCKTNAME*	**KRB5_CLIENT_KTNAME**
Kerberos config file *krb5.conf*	`/etc/krb5.conf:`*SYSCONFDIR*`/krb5.conf`	**KRB5_CONFIG**
KDC config file *kdc.conf*	*LOCALSTATEDIR*`/krb5kdc/kdc.conf`	**KRB5_KDC_PROFILE**
GSS mechanism config file	*SYSCONFDIR*`/gss/mech`	**GSS_MECH_CONFIG**
KDC database path (DB2)	*LOCALSTATEDIR*`/krb5kdc/principal`	
Master key stash_definitic	*LOCALSTATEDIR*`/krb5kdc/.k5.`*realm*	
Admin server ACL file *kadm5.acl*	*LOCALSTATEDIR*`/krb5kdc/kadm5.acl`	
OTP socket directory	*RUNSTATEDIR*`/krb5kdc`	
Plugin base directory	*LIBDIR*`/krb5/plugins`	
rcache_definit directory	`/var/tmp`	**KRB5RCACHEDIR**
Master key default enctype	`aes256-cts-hmac-sha1-96`	
Default *keysalt list*	`aes256-cts-hmac-sha1-96:normal aes128-cts-hmac-sha1-96:normal`	
Permitted enctypes	`aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 aes128-cts-hmac-sha256-128 des3-cbc-sha1 arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac`	
KDC default port	88	
Admin server port	749	

20.2 Replica KDC propagation defaults

This table shows defaults used by the *kprop* and *kpropd* programs.

Description	Default	Environment
kprop database dump file	*LOCALSTATEDIR*/krb5kdc/replica_datatrans	
kpropd temporary dump file	*LOCALSTATEDIR*/krb5kdc/from_master	
kdb5_util location	*SBINDIR*/kdb5_util	
kprop location	*SBINDIR*/kprop	
kpropd ACL file	*LOCALSTATEDIR*/krb5kdc/kpropd.acl	
kprop port	754	KPROP_PORT

20.3 Default paths for Unix-like systems

On Unix-like systems, some paths used by MIT krb5 depend on parameters chosen at build time. For a custom build, these paths default to subdirectories of `/usr/local`. When MIT krb5 is integrated into an operating system, the paths are generally chosen to match the operating system's filesystem layout.

Description	Symbolic name	Custom build path	Typical OS path
User programs	BINDIR	`/usr/local/bin`	`/usr/bin`
Libraries and plugins	LIBDIR	`/usr/local/lib`	`/usr/lib`
Parent of KDC state dir	LOCALSTATEDIR	`/usr/local/var`	`/var`
Parent of KDC runtime dir	RUNSTATEDIR	`/usr/local/var/run`	`/run`
Administrative programs	SBINDIR	`/usr/local/sbin`	`/usr/sbin`
Alternate krb5.conf dir	SYSCONFDIR	`/usr/local/etc`	`/etc`
Default ccache name	DEFCCNAME	`FILE:/tmp/krb5cc_%{uid}`	`FILE:/tmp/krb5cc_%{uid}`
Default keytab name	DEFKTNAME	`FILE:/etc/krb5.keytab`	`FILE:/etc/krb5.keytab`
Default PKCS11 module	PKCS11_MODNAME	`opensc-pkcs11.so`	`opensc-pkcs11.so`

The default client keytab name (DEFCKTNAME) typically defaults to `FILE:/usr/local/var/krb5/user/%{euid}/client.keytab` for a custom build. A native build will typically use a path which will vary according to the operating system's layout of `/var`.

CHAPTER

TWENTYONE

ENVIRONMENT VARIABLES

This content has moved to kerberos(7).

CHAPTER

TWENTYTWO

TROUBLESHOOTING

22.1 Trace logging

Most programs using MIT krb5 1.9 or later can be made to provide information about internal krb5 library operations using trace logging. To enable this, set the **KRB5_TRACE** environment variable to a filename before running the program. On many operating systems, the filename `/dev/stdout` can be used to send trace logging output to standard output.

Some programs do not honor **KRB5_TRACE**, either because they use secure library contexts (this generally applies to setuid programs and parts of the login system) or because they take direct control of the trace logging system using the API.

Here is a short example showing trace logging output for an invocation of the kvno(1) command:

```
shell% env KRB5_TRACE=/dev/stdout kvno krbtgt/KRBTEST.COM
[9138] 1332348778.823276: Getting credentials user@KRBTEST.COM ->
    krbtgt/KRBTEST.COM@KRBTEST.COM using ccache
    FILE:/me/krb5/build/testdir/ccache
[9138] 1332348778.823381: Retrieving user@KRBTEST.COM ->
    krbtgt/KRBTEST.COM@KRBTEST.COM from
    FILE:/me/krb5/build/testdir/ccache with result: 0/Unknown code 0
krbtgt/KRBTEST.COM@KRBTEST.COM: kvno = 1
```

22.2 List of errors

22.2.1 Frequently seen errors

1. *KDC has no support for encryption type while getting initial credentials*
2. *credential verification failed: KDC has no support for encryption type*
3. *Cannot create cert chain: certificate has expired*

22.2.2 Errors seen by admins

1. *kprop: No route to host while connecting to server*
2. *kprop: Connection refused while connecting to server*
3. *kprop: Server rejected authentication (during sendauth exchange) while authenticating to server*

KDC has no support for encryption type while getting initial credentials

credential verification failed: KDC has no support for encryption type

This most commonly happens when trying to use a principal with only DES keys, in a release (MIT krb5 1.7 or later) which disables DES by default. DES encryption is considered weak due to its inadequate key size. If you cannot migrate away from its use, you can re-enable DES by adding `allow_weak_crypto = true` to the *[libdefaults]* section of *krb5.conf*.

Cannot create cert chain: certificate has expired

This error message indicates that PKINIT authentication failed because the client certificate, KDC certificate, or one of the certificates in the signing chain above them has expired.

If the KDC certificate has expired, this message appears in the KDC log file, and the client will receive a "Preauthentication failed" error. (Prior to release 1.11, the KDC log file message erroneously appears as "Out of memory". Prior to release 1.12, the client will receive a "Generic error".)

If the client or a signing certificate has expired, this message may appear in *trace_logging* output from kinit(1) or, starting in release 1.12, as an error message from kinit or another program which gets initial tickets. The error message is more likely to appear properly on the client if the principal entry has no long-term keys.

kprop: No route to host while connecting to server

Make sure that the hostname of the replica KDC (as given to kprop) is correct, and that any firewalls between the primary and the replica allow a connection on port 754.

kprop: Connection refused while connecting to server

If the replica KDC is intended to run kpropd out of inetd, make sure that inetd is configured to accept krb5_prop connections. inetd may need to be restarted or sent a SIGHUP to recognize the new configuration. If the replica is intended to run kpropd in standalone mode, make sure that it is running.

kprop: Server rejected authentication (during sendauth exchange) while authenticating to server

Make sure that:

1. The time is synchronized between the primary and replica KDCs.
2. The master stash file was copied from the primary to the expected location on the replica.
3. The replica has a keytab file in the default location containing a `host` principal for the replica's hostname.

CHAPTER

TWENTYTHREE

ADVANCED TOPICS

23.1 Retiring DES

Version 5 of the Kerberos protocol was originally implemented using the Data Encryption Standard (DES) as a block cipher for encryption. While it was considered secure at the time, advancements in computational ability have rendered DES vulnerable to brute force attacks on its 56-bit keyspace. As such, it is now considered insecure and should not be used (**RFC 6649**).

23.1.1 History

DES was used in the original Kerberos implementation, and was the only cryptosystem in krb5 1.0. Partial support for triple-DES (3DES) was added in version 1.1, with full support following in version 1.2. The Advanced Encryption Standard (AES), which supersedes DES, gained partial support in version 1.3.0 of krb5 and full support in version 1.3.2. However, deployments of krb5 using Kerberos databases created with older versions of krb5 will not necessarily start using strong crypto for ordinary operation without administrator intervention.

MIT krb5 began flagging deprecated encryption types with release 1.17, and removed DES (single-DES) support in release 1.18. As a consequence, a release prior to 1.18 is required to perform these migrations.

23.1.2 Types of keys

- The database master key: This key is not exposed to user requests, but is used to encrypt other key material stored in the kerberos database. The database master key is currently stored as `K/M` by default.
- Password-derived keys: User principals frequently have keys derived from a password. When a new password is set, the KDC uses various string2key functions to generate keys in the database for that principal.
- Keytab keys: Application server principals generally use random keys which are not derived from a password. When the database entry is created, the KDC generates random keys of various enctypes to enter in the database, which are conveyed to the application server and stored in a keytab.
- Session keys: These are short-term keys generated by the KDC while processing client requests, with an enctype selected by the KDC.

For details on the various enctypes and how enctypes are selected by the KDC for session keys and client/server long-term keys, see *Encryption types*. When using the *kadmin* interface to generate new long-term keys, the **-e** argument can be used to force a particular set of enctypes, overriding the KDC default values.

Note: When the KDC is selecting a session key, it has no knowledge about the kerberos installation on the server which will receive the service ticket, only what keys are in the database for the service principal. In order to allow uninterrupted operation to clients while migrating away from DES, care must be taken to ensure that kerberos installations

on application server machines are configured to support newer encryption types before keys of those new encryption types are created in the Kerberos database for those server principals.

23.1.3 Upgrade procedure

This procedure assumes that the KDC software has already been upgraded to a modern version of krb5 that supports non-DES keys, so that the only remaining task is to update the actual keys used to service requests. The realm used for demonstrating this procedure, ZONE.MIT.EDU, is an example of the worst-case scenario, where all keys in the realm are DES. The realm was initially created with a very old version of krb5, and **supported_enctypes** in *kdc.conf* was set to a value appropriate when the KDC was installed, but was not updated as the KDC was upgraded:

```
[realms]
        ZONE.MIT.EDU = {
                [...]
                master_key_type = des-cbc-crc
                supported_enctypes = des-cbc-crc:normal des:normal des:v4 des:norealm␣
↪des:onlyrealm des:afs3
        }
```

This resulted in the keys for all principals in the realm being forced to DES-only, unless specifically requested using *kadmin*.

Before starting the upgrade, all KDCs were running krb5 1.11, and the database entries for some "high-value" principals were:

```
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc krbtgt/ZONE.MIT.EDU'
[...]
Number of keys: 1
Key: vno 1, des-cbc-crc:v4
[...]
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc kadmin/admin'
[...]
Number of keys: 1
Key: vno 15, des-cbc-crc
[...]
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc kadmin/changepw'
[...]
Number of keys: 1
Key: vno 14, des-cbc-crc
[...]
```

The `krbtgt/REALM` key appears to have never been changed since creation (its kvno is 1), and all three database entries have only a des-cbc-crc key.

The krbtgt key and KDC keys

Perhaps the biggest single-step improvement in the security of the cell is gained by strengthening the key of the ticket-granting service principal, krbtgt/REALM—if this principal's key is compromised, so is the entire realm. Since the server that will handle service tickets for this principal is the KDC itself, it is easy to guarantee that it will be configured to support any encryption types which might be selected. However, the default KDC behavior when creating new keys is to remove the old keys, which would invalidate all existing tickets issued against that principal, rendering the TGTs cached by clients useless. Instead, a new key can be created with the old key retained, so that existing tickets will still function until their scheduled expiry (see *Changing the krbtgt key*).

```
[root@casio krb5kdc]# enctypes=aes256-cts-hmac-sha1-96:normal,\
> aes128-cts-hmac-sha1-96:normal,des3-hmac-sha1:normal,des-cbc-crc:normal
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q "cpw -e ${enctypes} -randkey \
> -keepold krbtgt/ZONE.MIT.EDU"
Authenticating as principal root/admin@ZONE.MIT.EDU with password.
Key for "krbtgt/ZONE.MIT.EDU@ZONE.MIT.EDU" randomized.
```

Note: The new krbtgt@REALM key should be propagated to replica KDCs immediately so that TGTs issued by the primary KDC can be used to issue service tickets on replica KDCs. Replica KDCs will refuse requests using the new TGT kvno until the new krbtgt entry has been propagated to them.

It is necessary to explicitly specify the enctypes for the new database entry, since **supported_enctypes** has not been changed. Leaving **supported_enctypes** unchanged makes a potential rollback operation easier, since all new keys of new enctypes are the result of explicit administrator action and can be easily enumerated. Upgrading the krbtgt key should have minimal user-visible disruption other than that described in the note above, since only clients which list the new enctypes as supported will use them, per the procedure in *Session key selection*. Once the krbtgt key is updated, the session and ticket keys for user TGTs will be strong keys, but subsequent requests for service tickets will still get DES keys until the service principals have new keys generated. Application service remains uninterrupted due to the key-selection procedure on the KDC.

After the change, the database entry is now:

```
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc krbtgt/ZONE.MIT.EDU'
[...]
Number of keys: 5
Key: vno 2, aes256-cts-hmac-sha1-96
Key: vno 2, aes128-cts-hmac-sha1-96
Key: vno 2, des3-cbc-sha1
Key: vno 2, des-cbc-crc
Key: vno 1, des-cbc-crc:v4
[...]
```

Since the expected disruptions from rekeying the krbtgt principal are minor, after a short testing period, it is appropriate to rekey the other high-value principals, kadmin/admin@REALM and kadmin/changepw@REALM. These are the service principals used for changing user passwords and updating application keytabs. The kadmin and password-changing services are regular kerberized services, so the session-key-selection algorithm described in *Session key selection* applies. It is particularly important to have strong session keys for these services, since user passwords and new long-term keys are conveyed over the encrypted channel.

```
[root@casio krb5kdc]# enctypes=aes256-cts-hmac-sha1-96:normal,\
> aes128-cts-hmac-sha1-96:normal,des3-hmac-sha1:normal
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q "cpw -e ${enctypes} -randkey \
> kadmin/admin"
```

(continues on next page)

(continued from previous page)

```
Authenticating as principal root/admin@ZONE.MIT.EDU with password.
Key for "kadmin/admin@ZONE.MIT.EDU" randomized.
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q "cpw -e ${enctypes} -randkey \
> kadmin/changepw"
Authenticating as principal root/admin@ZONE.MIT.EDU with password.
Key for "kadmin/changepw@ZONE.MIT.EDU" randomized.
```

It is not necessary to retain a single-DES key for these services, since password changes are not part of normal daily workflow, and disruption from a client failure is likely to be minimal. Furthermore, if a kerberos client experiences failure changing a user password or keytab key, this indicates that that client will become inoperative once services are rekeyed to non-DES enctypes. Such problems can be detected early at this stage, giving more time for corrective action.

Adding strong keys to application servers

Before switching the default enctypes for new keys over to strong enctypes, it may be desired to test upgrading a handful of services with the new configuration before flipping the switch for the defaults. This still requires using the **-e** argument in *kadmin* to get non-default enctypes:

```
[root@casio krb5kdc]# enctypes=aes256-cts-hmac-sha1-96:normal,\
> aes128-cts-hmac-sha1-96:normal,des3-cbc-sha1:normal,des-cbc-crc:normal
[root@casio krb5kdc]# kadmin -r ZONE.MIT.EDU -p zephyr/zephyr@ZONE.MIT.EDU -k -t \
> /etc/zephyr/krb5.keytab  -q "ktadd -e ${enctypes} \
> -k /etc/zephyr/krb5.keytab zephyr/zephyr@ZONE.MIT.EDU"
Authenticating as principal zephyr/zephyr@ZONE.MIT.EDU with keytab /etc/zephyr/krb5.
↪keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 4, encryption type aes256-cts-
↪hmac-sha1-96 added to keytab WRFILE:/etc/zephyr/krb5.keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 4, encryption type aes128-cts-
↪hmac-sha1-96 added to keytab WRFILE:/etc/zephyr/krb5.keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 4, encryption type des3-cbc-
↪sha1 added to keytab WRFILE:/etc/zephyr/krb5.keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 4, encryption type des-cbc-crc␣
↪added to keytab WRFILE:/etc/zephyr/krb5.keytab.
```

Be sure to remove the old keys from the application keytab, per best practice.

```
[root@casio krb5kdc]# k5srvutil -f /etc/zephyr/krb5.keytab delold
Authenticating as principal zephyr/zephyr@ZONE.MIT.EDU with keytab /etc/zephyr/krb5.
↪keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 3 removed from keytab WRFILE:/
↪etc/zephyr/krb5.keytab.
```

Adding strong keys by default

Once the high-visibility services have been rekeyed, it is probably appropriate to change *kdc.conf* to generate keys with the new encryption types by default. This enables server administrators to generate new enctypes with the **change** subcommand of *k5srvutil*, and causes user password changes to add new encryption types for their entries. It will probably be necessary to implement administrative controls to cause all user principal keys to be updated in a reasonable period of time, whether by forcing password changes or a password synchronization service that has access to the current password and can add the new keys.

```
[realms]
        ZONE.MIT.EDU = {
                supported_enctypes = aes256-cts-hmac-sha1-96:normal aes128-cts-hmac-sha1-
↪96:normal des3-cbc-sha1:normal des3-hmac-sha1:normal des-cbc-crc:normal
```

Note: The krb5kdc process must be restarted for these changes to take effect.

At this point, all service administrators can update their services and the servers behind them to take advantage of strong cryptography. If necessary, the server's krb5 installation should be configured and/or upgraded to a version supporting non-DES keys. See *Encryption types* for krb5 version and configuration settings. Only when the service is configured to accept non-DES keys should the key version number be incremented and new keys generated (`k5srvutil change && k5srvutil delold`).

```
root@dr-willy:~# k5srvutil change
Authenticating as principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with keytab /etc/krb5.
↪keytab.
Entry for principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with kvno 3, encryption type␣
↪AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with kvno 3, encryption type␣
↪AES-128 CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with kvno 3, encryption type␣
↪Triple DES cbc mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with kvno 3, encryption type␣
↪DES cbc mode with CRC-32 added to keytab WRFILE:/etc/krb5.keytab.
root@dr-willy:~# klist -e -k -t /etc/krb5.keytab
Keytab name: WRFILE:/etc/krb5.keytab
KVNO Timestamp         Principal
---- ----------------- --------------------------------------------------------
   2 10/10/12 17:03:59 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU (DES cbc mode with CRC-32)
   3 12/12/12 15:31:19 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU (AES-256 CTS mode with 96-
↪bit SHA-1 HMAC)
   3 12/12/12 15:31:19 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU (AES-128 CTS mode with 96-
↪bit SHA-1 HMAC)
   3 12/12/12 15:31:19 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU (Triple DES cbc mode with␣
↪HMAC/sha1)
   3 12/12/12 15:31:19 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU (DES cbc mode with CRC-32)
root@dr-willy:~# k5srvutil delold
Authenticating as principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with keytab /etc/krb5.
↪keytab.
Entry for principal host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU with kvno 2 removed from␣
↪keytab WRFILE:/etc/krb5.keytab.
```

When a single service principal is shared by multiple backend servers in a load-balanced environment, it may be necessary to schedule downtime or adjust the population in the load-balanced pool in order to propagate the updated

keytab to all hosts in the pool with minimal service interruption.

Removing DES keys from usage

This situation remains something of a testing or transitory state, as new DES keys are still being generated, and will be used if requested by a client. To make more progress removing DES from the realm, the KDC should be configured to not generate such keys by default.

Note: An attacker posing as a client can implement a brute force attack against a DES key for any principal, if that key is in the current (highest-kvno) key list. This attack is only possible if **allow_weak_crypto = true** is enabled on the KDC. Setting the **+requires_preauth** flag on a principal forces this attack to be an online attack, much slower than the offline attack otherwise available to the attacker. However, setting this flag on a service principal is not always advisable; see the entry in *add_principal* for details.

The following KDC configuration will not generate DES keys by default:

```
[realms]
        ZONE.MIT.EDU = {
                supported_enctypes = aes256-cts-hmac-sha1-96:normal aes128-cts-hmac-sha1-
↪96:normal des3-cbc-sha1:normal des3-hmac-sha1:normal
```

Note: As before, the KDC process must be restarted for this change to take effect. It is best practice to update kdc.conf on all KDCs, not just the primary, to avoid unpleasant surprises should the primary fail and a replica need to be promoted.

It is now appropriate to remove the legacy single-DES key from the `krbtgt/REALM` entry:

```
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q "cpw -randkey -keepold \
> krbtgt/ZONE.MIT.EDU"
Authenticating as principal host/admin@ATHENA.MIT.EDU with password.
Key for "krbtgt/ZONE.MIT.EDU@ZONE.MIT.EDU" randomized.
```

After the maximum ticket lifetime has passed, the old database entry should be removed.

```
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'purgekeys krbtgt/ZONE.MIT.EDU'
Authenticating as principal root/admin@ZONE.MIT.EDU with password.
Old keys for principal "krbtgt/ZONE.MIT.EDU@ZONE.MIT.EDU" purged.
```

After the KDC is restarted with the new **supported_enctypes**, all user password changes and application keytab updates will not generate DES keys by default.

```
contents-vnder-pressvre:~> kpasswd zonetest@ZONE.MIT.EDU
Password for zonetest@ZONE.MIT.EDU:  [enter old password]
Enter new password:                  [enter new password]
Enter it again:                      [enter new password]
Password changed.
contents-vnder-pressvre:~> kadmin -r ZONE.MIT.EDU -q 'getprinc zonetest'
[...]
Number of keys: 3
Key: vno 9, aes256-cts-hmac-sha1-96
Key: vno 9, aes128-cts-hmac-sha1-96
```

(continues on next page)

(continued from previous page)

```
Key: vno 9, des3-cbc-sha1
[...]

[kaduk@glossolalia ~]$ kadmin -p kaduk@ZONE.MIT.EDU -r ZONE.MIT.EDU -k \
> -t kaduk-zone.keytab -q 'ktadd -k kaduk-zone.keytab kaduk@ZONE.MIT.EDU'
Authenticating as principal kaduk@ZONE.MIT.EDU with keytab kaduk-zone.keytab.
Entry for principal kaduk@ZONE.MIT.EDU with kvno 3, encryption type aes256-cts-hmac-sha1-
↪96 added to keytab WRFILE:kaduk-zone.keytab.
Entry for principal kaduk@ZONE.MIT.EDU with kvno 3, encryption type aes128-cts-hmac-sha1-
↪96 added to keytab WRFILE:kaduk-zone.keytab.
Entry for principal kaduk@ZONE.MIT.EDU with kvno 3, encryption type des3-cbc-sha1 added␣
↪to keytab WRFILE:kaduk-zone.keytab.
```

Once all principals have been re-keyed, DES support can be disabled on the KDC (**allow_weak_crypto = false**), and client machines can remove **allow_weak_crypto = true** from their *krb5.conf* configuration files, completing the migration. **allow_weak_crypto** takes precedence over all places where DES enctypes could be explicitly configured. DES keys will not be used, even if they are present, when **allow_weak_crypto = false**.

Support for legacy services

If there remain legacy services which do not support non-DES enctypes (such as older versions of AFS), **allow_weak_crypto** must remain enabled on the KDC. Client machines need not have this setting, though—applications which require DES can use API calls to allow weak crypto on a per-request basis, overriding the system krb5.conf. However, having **allow_weak_crypto** set on the KDC means that any principals which have a DES key in the database could still use those keys. To minimize the use of DES in the realm and restrict it to just legacy services which require DES, it is necessary to remove all other DES keys. The realm has been configured such that at password and keytab change, no DES keys will be generated by default. The task then reduces to requiring user password changes and having server administrators update their service keytabs. Administrative outreach will be necessary, and if the desire to eliminate DES is sufficiently strong, the KDC administrators may choose to randkey any principals which have not been rekeyed after some timeout period, forcing the user to contact the helpdesk for access.

23.1.4 The Database Master Key

This procedure does not alter `K/M@REALM`, the key used to encrypt key material in the Kerberos database. (This is the key stored in the stash file on the KDC if stash files are used.) However, the security risk of a single-DES key for `K/M` is minimal, given that access to material encrypted in `K/M` (the Kerberos database) is generally tightly controlled. If an attacker can gain access to the encrypted database, they likely have access to the stash file as well, rendering the weak cryptography broken by non-cryptographic means. As such, upgrading `K/M` to a stronger encryption type is unlikely to be a high-priority task.

Is is possible to upgrade the master key used for the database, if desired. Using *kdb5_util*'s **add_mkey**, **use_mkey**, and **update_princ_encryption** commands, a new master key can be added and activated for use on new key material, and the existing entries converted to the new master key.

CHAPTER

TWENTYFOUR

VARIOUS LINKS

24.1 Whitepapers

1. https://kerberos.org/software/whitepapers.html

24.2 Tutorials

1. Fulvio Ricciardi <https://www.kerberos.org/software/tutorial.html>_

24.3 Troubleshooting

1. https://wiki.ncsa.illinois.edu/display/ITS/Windows+Kerberos+Troubleshooting
2. https://www.shrubbery.net/solaris9ab/SUNWaadm/SYSADV6/p27.html
3. https://docs.oracle.com/cd/E19253-01/816-4557/trouble-1/index.html
4. https://docs.microsoft.com/en-us/previous-versions/tn-archive/bb463167(v=technet.10)#EBAA
5. https://bugs.launchpad.net/ubuntu/+source/libpam-heimdal/+bug/86528

INDEX

R

RFC
    RFC 2253, 25
    RFC 2782, 47
    RFC 4556, 26, 38, 39
    RFC 5480, 16
    RFC 6649, 151
    RFC 7553, 48
    RFC 7748, 16