## CONTENTS

1 Installation guide .................................................. 1
   1.1 Contents ..................................................... 1
   1.2 Additional references ........................................ 10

2 Configuration Files ................................................. 11
   2.1 krb5.conf .................................................. 11
   2.2 kdc.conf .................................................. 23
   2.3 kadm5.acl .................................................. 31

3 Realm configuration decisions ..................................... 35
   3.1 Realm name ................................................ 35
   3.2 Mapping hostnames onto Kerberos realms ..................... 35
   3.3 Ports for the KDC and admin services ......................... 36
   3.4 Slave KDCs ............................................... 36
   3.5 Hostnames for KDCs ........................................ 36
   3.6 Database propagation ....................................... 37

4 Database administration ............................................ 39
   4.1 kadmin options ............................................ 39
   4.2 Date Format ............................................... 40
   4.3 Principals ............................................... 40
   4.4 Policies ................................................ 45
   4.5 Privileges ............................................... 48
   4.6 Operations on the Kerberos database ......................... 48
   4.7 Operations on the LDAP database ........................... 52
   4.8 Cross-realm authentication ................................ 57
   4.9 Changing the krbtgt key ................................... 58
   4.10 Incremental database propagation ......................... 58

5 Account lockout .................................................... 61
   5.1 Configuring account lockout ................................ 61
   5.2 Testing account lockout ................................... 61
   5.3 Account lockout principal state ............................ 62
   5.4 KDC replication and account lockout ......................... 62
   5.5 KDC performance and account lockout ....................... 62

6 Configuring Kerberos with OpenLDAP back-end .................... 65

7 Application servers ............................................... 69
   7.1 Keytabs ................................................ 69
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2 Clock Skew</td>
<td>70</td>
</tr>
<tr>
<td>7.3 Getting DNS information correct</td>
<td>71</td>
</tr>
<tr>
<td>7.4 Configuring your firewall to work with Kerberos V5</td>
<td>71</td>
</tr>
<tr>
<td><strong>8</strong> Host configuration</td>
<td>73</td>
</tr>
<tr>
<td>8.1 Plugin module configuration</td>
<td>73</td>
</tr>
<tr>
<td><strong>9</strong> Backups of secure hosts</td>
<td>75</td>
</tr>
<tr>
<td>9.1 Backing up the Kerberos database</td>
<td>75</td>
</tr>
<tr>
<td><strong>10</strong> PKINIT configuration</td>
<td>77</td>
</tr>
<tr>
<td>10.1 Generating a certificate authority certificate</td>
<td>77</td>
</tr>
<tr>
<td>10.2 Generating a KDC certificate</td>
<td>77</td>
</tr>
<tr>
<td>10.3 Generating client certificates</td>
<td>78</td>
</tr>
<tr>
<td>10.4 Configuring the KDC</td>
<td>79</td>
</tr>
<tr>
<td>10.5 Configuring the clients</td>
<td>79</td>
</tr>
<tr>
<td>10.6 Anonymous PKINIT</td>
<td>79</td>
</tr>
<tr>
<td><strong>11</strong> Principal names and DNS</td>
<td>81</td>
</tr>
<tr>
<td>11.1 Service principal names</td>
<td>81</td>
</tr>
<tr>
<td>11.2 Service principal canonicalization</td>
<td>81</td>
</tr>
<tr>
<td>11.3 Reverse DNS mismatches</td>
<td>82</td>
</tr>
<tr>
<td>11.4 Overriding application behavior</td>
<td>82</td>
</tr>
<tr>
<td>11.5 Provisioning keytabs</td>
<td>82</td>
</tr>
<tr>
<td>11.6 Specific application advice</td>
<td>82</td>
</tr>
<tr>
<td><strong>12</strong> Encryption types</td>
<td>83</td>
</tr>
<tr>
<td>12.1 Enctypes in requests</td>
<td>83</td>
</tr>
<tr>
<td>12.2 Session key selection</td>
<td>83</td>
</tr>
<tr>
<td>12.3 Choosing enctypes for a service</td>
<td>84</td>
</tr>
<tr>
<td>12.4 Configuration variables</td>
<td>84</td>
</tr>
<tr>
<td>12.5 Enctype compatibility</td>
<td>84</td>
</tr>
<tr>
<td><strong>13</strong> Administration programs</td>
<td>85</td>
</tr>
<tr>
<td>13.1 kadmin</td>
<td>85</td>
</tr>
<tr>
<td>13.2 kadmind</td>
<td>95</td>
</tr>
<tr>
<td>13.3 kdb5_util</td>
<td>96</td>
</tr>
<tr>
<td>13.4 kdb5_ldap_util</td>
<td>100</td>
</tr>
<tr>
<td>13.5 krb5kdc</td>
<td>105</td>
</tr>
<tr>
<td>13.6 kprop</td>
<td>106</td>
</tr>
<tr>
<td>13.7 kpropd</td>
<td>107</td>
</tr>
<tr>
<td>13.8 kproplog</td>
<td>109</td>
</tr>
<tr>
<td>13.9 ktutil</td>
<td>110</td>
</tr>
<tr>
<td>13.10 k5srvutil</td>
<td>112</td>
</tr>
<tr>
<td>13.11 sserver</td>
<td>112</td>
</tr>
<tr>
<td><strong>14</strong> MIT Kerberos defaults</td>
<td>115</td>
</tr>
<tr>
<td>14.1 General defaults</td>
<td>116</td>
</tr>
<tr>
<td>14.2 Slave KDC propagation defaults</td>
<td>117</td>
</tr>
<tr>
<td>14.3 Default paths for Unix-like systems</td>
<td>117</td>
</tr>
<tr>
<td><strong>15</strong> Environment variables</td>
<td>119</td>
</tr>
<tr>
<td><strong>16</strong> Troubleshooting</td>
<td>121</td>
</tr>
<tr>
<td>16.1 Trace logging</td>
<td>121</td>
</tr>
</tbody>
</table>
1.1 Contents

1.1.1 Installing KDCs

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

Warning:
- The Kerberos system relies on the availability of correct time information. Ensure that the master and all slave 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 master 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kerberos.mit.edu</td>
<td>master KDC</td>
</tr>
<tr>
<td>kerberos-1.mit.edu</td>
<td>slave KDC</td>
</tr>
<tr>
<td>ATHENA.MIT.EDU</td>
<td>realm name</td>
</tr>
<tr>
<td>.k5.ATHENA.MIT.EDU</td>
<td>stash file</td>
</tr>
<tr>
<td>admin/admin</td>
<td>admin principal</td>
</tr>
</tbody>
</table>

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 SRV records (see Hostnames for KDCs), 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:

```
[kdcdesigns]
  kdc_ports = 88,750

[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,
    # 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 master 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 “MITys4K5!”, 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 master KDC, using the `kdb5_util` command. Replace ATHENA.MIT.EDU with the name of your Kerberos realm:

```shell
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.

This will create five files in /etc/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 to 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 `/etc/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 master KDC. `kadmin.local` is designed to be run on the master 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

WARNING: 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.
```

Start the Kerberos daemons on the master KDC

At this point, you are ready to start the Kerberos KDC (`krb5kdc`) and administrative daemons on the Master 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 \textit{kinit(1)} succeeds against the principals that you have created on the previous step \textit{(Add administrators to the Kerberos database)}. Run:

\begin{verbatim}
shell% kinit admin/admin@ATHENA.MIT.EDU
\end{verbatim}

**Install the slave KDCs**

You are now ready to start configuring the slave KDCs.

\textbf{Note:} Assuming you are setting the KDCs up so that you can easily switch the master KDC with one of the slaves, you should perform each of these steps on the master KDC as well as the slave KDCs, unless these instructions specify otherwise.

**Create host keytabs for slave 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 master KDC to the secondary KDC servers.

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

\begin{verbatim}
shell% kadmin
kadmin: addprinc -randkey host/kerberos.mit.edu
NOTICE: 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
NOTICE: no policy specified for "host/kerberos-1.mit.edu@ATHENA.MIT.EDU"; assigning "default"
Principal "host/kerberos-1.mit.edu@ATHENA.MIT.EDU" created.
\end{verbatim}

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

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 on a slave KDC called \textit{kerberos-1.mit.edu}, you would execute the following command:

\begin{verbatim}
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 des3-cbc-sha1 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.
\end{verbatim}
Configure slave KDCs

Database propagation copies the contents of the master’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 slave (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 master KDC. kadm5.acl is only needed to allow a slave to swap with the master KDC.

The database is propagated from the master KDC to the slave KDCs via the *kpropd* daemon. You must explicitly specify the principals which are allowed to provide Kerberos dump updates on the slave 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 master and slave 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 master KDC’s host principal in the kpropd.acl files of the slave 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 slave propagation
```

Restart inetd daemon.

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

Now that the slave KDC is able to accept database propagation, you’ll need to propagate the database from the master server.

**NOTE:** Do not start the slave KDC yet; you still do not have a copy of the master’s database.

**Propagate the database to each slave KDC**

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

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

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

```
shell% kprop -f /usr/local/var/krb5kdc/slave_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.

```bash
#!/bin/sh

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

kdb5_util dump /usr/local/var/krb5kdc/slave_datatrans

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

You will need to replace `/usr/local/var/krb5kdc` with the name of the KDC state directory.

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

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 slave KDC has a copy of the Kerberos database, you can start the krb5kdc daemon:

```bash
shell% krb5kdc
```

As with the master 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?**

**Error:** kprop: No route to host while connecting to server

Make sure that the hostname of the slave (as given to kprop) is correct, and that any firewalls between the master and the slave allow a connection on port 754.

**Error:** kprop: Connection refused in call to connect while opening connection

If the slave 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 slave is intended to run kpropd in standalone mode, make sure that it is running.

**Error:** kprop: Server rejected authentication while authenticating to server

Make sure that:

1. The time is synchronized between the master and slave KDCs.
2. The master stash file was copied from the master to the expected location on the slave.
3. The slave has a keytab file in the default location containing a host principal for the slave’s hostname.

**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 Adding, modifying and deleting principals.

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

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

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

If the master KDC is still running, do the following on the **old** master 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 slaves all have the latest copy of the database (see [Propagate the database to each slave KDC](#)).

On the **new** master KDC:

1. Start the `kadmind` daemon (see [Start the Kerberos daemons on the master KDC](#)).
2. Set up the cron job to propagate the database (see [Propagate the database to each slave KDC](#)).
3. Switch the CNAMEs of the old and new master 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 slave 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 `/usr/bin`.

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, 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 `FILE:/etc/krb5.keytab`. 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 *Adding, modifying and deleting principals*. (See *Create host keytabs for slave 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`):

```bash
trillium% kadmin
kadmin5: ktadd host/trillium.mit.edu ftp/trillium.mit.edu pop/trillium.mit.edu
kadmin: Entry for principal host/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption type DES-CBC-CRC added to keytab FILE:/etc/krb5.keytab.
kadmin: Entry for principal ftp/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption type DES-CBC-CRC added to keytab FILE:/etc/krb5.keytab.
kadmin: Entry for principal pop/trillium.mit.edu@ATHENA.MIT.EDU with kvno 3, encryption type DES-CBC-CRC added to keytab FILE:/etc/krb5.keytab.
kadmin5: 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.

---

1.1. Contents 9
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 (FILE:/etc/krb5.keytab). 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
2.1 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`.

2.1.1 Structure

The krb5.conf file is set up in the style of a Windows INI file. Sections are headed by the section name, in square brackets. Each section may contain zero or more relations, of the form:

```plaintext
foo = bar
```

or

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

Placing a `'*'` at the end of a line indicates that this is the *final* value for the tag. This means that neither the remainder of this configuration file nor any other configuration file will be checked for any other values for this tag.

For example, if you have the following lines:

```plaintext
foo = bar*
foo = baz
```

then the second value of `foo` would never be read.

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

```plaintext
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. Included profile files are syntactically independent of their parents, so each included file must begin with a section header.

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.

### 2.1.2 Sections

The krb5.conf file may contain the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[libdefaults]</td>
<td>Settings used by the Kerberos V5 library</td>
</tr>
<tr>
<td>[realms]</td>
<td>Realm-specific contact information and settings</td>
</tr>
<tr>
<td>[domain_realm]</td>
<td>Maps server hostnames to Kerberos realms</td>
</tr>
<tr>
<td>[capaths]</td>
<td>Authentication paths for non-hierarchical cross-realm</td>
</tr>
<tr>
<td>[appdefaults]</td>
<td>Settings used by some Kerberos V5 applications</td>
</tr>
<tr>
<td>[plugins]</td>
<td>Controls plugin module registration</td>
</tr>
</tbody>
</table>

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_weak_crypto** If this flag is set to false, then weak encryption types will be filtered out of the previous three lists (as noted in Encryption and salt types in kdc.conf). The default value for this tag is false, which may cause authentication failures in existing Kerberos infrastructures that do not support strong crypto. Users in affected environments should set this tag to true until their infrastructure adopts stronger ciphers.

- **ap_req_checksum_type** An integer which specifies the type of AP-REQ checksum to use in authenticators. This variable should be unset so the appropriate checksum for the encryption key in use will be used. This can be set if backward compatibility requires a specific checksum type. See the kdc_req_checksum_type configuration option for the possible values and their meanings.

- **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.

- **default_ccache_name** This relation specifies the name of the default credential cache. The default is FILE:/tmp/krb5cc_%.uid. This relation is subject to parameter expansion (see below).

- **default_client_keytab_name** This relation specifies the name of the default keytab for obtaining client credentials. The default is FILE:/etc/krb5/user/%{euid}/client.keytab. This relation is subject to parameter expansion (see below).

- **default_keytab_name** This relation specifies the default keytab name to be used by application servers such as sshd. The default is FILE:/etc/krb5.keytab. This relation is subject to parameter expansion (see below).

- **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 and salt types in kdc.conf for a list of the accepted values for this tag. The default value is aes256-cts-hmac-shal-96 aes128-cts-hmac-shal-96 des3-cbc-shal arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac des-cbc-crc des-cbc-md5 des-cbc-md4, but single-DES encryption types will be implicitly removed from this list if the value of allow_weak_crypto is false.

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. The default value for this tag is aes256-cts-hmac-shal-96 aes128-cts-hmac-shal-96 des3-cbc-shal arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac des-cbc-crc des-cbc-md5 des-cbc-md4, but single-DES encryption types will be implicitly removed from this list if the value of allow_weak_crypto is false.

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_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.

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.

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.

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.
**kdc_req_checksum_type** An integer which specifies the type of checksum to use for the KDC requests, for compatibility with very old KDC implementations. This value is only used for DES keys; other keys use the preferred checksum type for those keys.

The possible values and their meanings are as follows.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CRC32</td>
</tr>
<tr>
<td>2</td>
<td>RSA MD4</td>
</tr>
<tr>
<td>3</td>
<td>RSA MD4 DES</td>
</tr>
<tr>
<td>4</td>
<td>DES CBC</td>
</tr>
<tr>
<td>7</td>
<td>RSA MD5</td>
</tr>
<tr>
<td>8</td>
<td>RSA MD5 DES</td>
</tr>
<tr>
<td>9</td>
<td>NIST SHA</td>
</tr>
<tr>
<td>12</td>
<td>HMAC SHA1 DES3</td>
</tr>
<tr>
<td>-138</td>
<td>Microsoft MD5 HMAC checksum type</td>
</tr>
</tbody>
</table>

**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 all encryption types that are permitted for use in session key encryption. The default value for this tag is `aes256-cts-hmac-shal-96 aes128-cts-hmac-shal-96 des3-cbc-shal arcfour-hmac-md5 camellia256-cts-cmac camellia128-cts-cmac des-cbc-crc des-cbc-md5 des-cbc-md4`, but single-DES encryption types will be implicitly removed from this list if the value of `allow_weak_crypto` is false.

**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.

**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.

**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. 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.

**safe_checksum_type** An integer which specifies the type of checksum to use for the KRB-SAFE requests. By default it is set to 8 (RSA MD5 DES). For compatibility with applications linked against DCE version 1.1 or earlier Kerberos libraries, use a value of 3 to use the RSA MD4 DES instead. This field is ignored when its value is incompatible with the session key type. See the `kdc_req_checksum_type` configuration option for the possible values and their meanings.

**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.
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 master 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:

```plaintext
[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/
    auto_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/hostame.domain`).

**kdc** The name or address of a host running a KDC for that realm. 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** Points to the server where all the password changes are performed. If there is no such entry, the port 464 on the `admin_server` host will be tried.

**master_kdc** Identifies the master 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 master KDC, in case the user’s password has just been changed, and the updated database has not been propagated to the slave servers yet.
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 a domain name or hostname to a Kerberos realm name. The tag name can be a host name or domain name, where domain names are indicated by a prefix of a period (.). The value of the relation is the Kerberos realm name for that particular host or domain. The Kerberos realm may be identified either in the realms section or using DNS SRV records. Host names and domain names should be in lower case. For example:

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

maps the host with the exact name crash.mit.edu into the TEST.ATHENA.MIT.EDU realm. The period prefix in .mit.edu denotes that all systems in the mit.edu domain belong to ATHENA.MIT.EDU realm. The third entry maps the host mit.edu itself to the ATHENA.MIT.EDU realm.

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 than 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
    }
```
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
    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]
teinnet = {
    ATHENA.MIT.EDU = {
        option1 = false
    }
}
teinnet = {
    option1 = true
    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.

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].

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

**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

**kadmd5_hook interface**

The kadmd5_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.

**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.

### 2.1.3 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:** 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 file 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* 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 opensc-pkcs11.so. *slotid= and/or token= may be specified to force the use of a particular smart 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. See the **pkinit_cert_match** configuration option for more ways to select a particular certificate to use for PKINIT.

**ENV:** *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

<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 as used by
  Microsoft will be accepted.
- 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 size of the Diffie-Hellman key the client will attempt to use. The acceptable values
are 1024, 2048, and 4096. The default is 2048.

pkinit_identities Specifies the location(s) to be used to find the user’s X.509 identity information. This option may be
specified multiple times. Each value is attempted in order until identity information is found and authentication
is attempted. 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 dDNSName 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_longhorn If this flag is set to true, we are talking to the Longhorn KDC.

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.

**pkinit_win2k** This flag specifies whether the target realm is assumed to support only the old, pre-RFC version of the protocol. The default is false.

**pkinit_win2k_require_binding** If this flag is set to true, it expects that the target KDC is patched to return a reply with a checksum rather than a nonce. The default is false.

### 2.1.4 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

### 2.1.5 Sample krb5.conf file

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

```
[libdefaults]
default_realm = ATHENA.MIT.EDU
default_tkt_enctypes = des3-hmac-sha1 des-cbc-crc
default_tgs_enctypes = des3-hmac-sha1 des-cbc-crc
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:750
    admin_server = kerberos.mit.edu
    master_kdc = kerberos.mit.edu
}
```
default_domain = mit.edu
}
EXAMPLE.COM = {
    kdc = kerberos.example.com
    kdc = kerberos-1.example.com
    admin_server = kerberos.example.com
}

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

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

2.1.6 FILES

/etc/krb5.conf

2.1.7 SEE ALSO

syslog(3)

2.2 kdc.conf

The kdc.conf file supplements krb5.conf for programs which are typically only used on a KDC, such as the krb5kdc and kadm5 daemons and the kdb5_util program. Relations documented here may also be specified in krb5.conf.

Normally, the kdc.conf file is found in the KDC state directory, /etc/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.

2.2.1 Structure

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

2.2.2 Sections

The kdc.conf file may contain the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[kdcdefaults]</td>
<td>Default values for KDC behavior</td>
</tr>
<tr>
<td>[realms]</td>
<td>Realm-specific database configuration and settings</td>
</tr>
<tr>
<td>[dbdefaults]</td>
<td>Default database settings</td>
</tr>
<tr>
<td>[dbmodules]</td>
<td>Per-database settings</td>
</tr>
<tr>
<td>[logging]</td>
<td>Controls how Kerberos daemons perform logging</td>
</tr>
</tbody>
</table>

2.2. kdc.conf
[kdcdefaults]

With one exception, 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_ports
- kdc_tcp_ports
- no_host_referral
- restrict_anonymous_to_tgt

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

[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.

For each realm, the following tags may be specified:

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. The default value is /etc/krb5kdc/kadm5.acl. For more information on Kerberos ACL file see kadm5.acl.

database_module  This relation indicates the name of the configuration section under [dbmodules] for database specific parameters used by the loadable database library.

database_name  (String.) This string specifies the location of the Kerberos database for this realm, if the DB2 back-end is being used. If a database_module is specified for the realm and the corresponding module contains a database_name parameter, that value will take precedence over this one. The default value is /etc/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 principal to obtain a session key for another user, permitting user-to-user authentication 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 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.

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. If none is specified or if there is no policy assigned to the principal, no dictionary checks of passwords will be performed.

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_master_ugsize (Integer.) Specifies the maximum number of log entries to be retained for incremental propagation. The maximum value is 2500; the default value is 1000.

iprop_slave_poll (Delta time string.) Specifies how often the slave KDC polls for new updates from the master. The default value is 2m (that is, two minutes).

iprop_port (Port number.) Specifies the port number to be used for incremental propagation. This is required in both master and slave configuration files.

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 slave KDCs only. The default value is 5 minutes (5m).

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_port (Port number.) Specifies the port on which the kadmind daemon is to listen for this realm. 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 /etc/krb5kdc/.k5.REALM, where REALM is the Kerberos realm.

kdc_ports (Whitespace- or comma-separated list.) Lists the ports on which the Kerberos server should listen for UDP requests, as a comma-separated list of integers. The default value is 88,750, which are the assigned Kerberos port and the port historically used by Kerberos V4.
**kdc_tcp_ports** (Whitespace- or comma-separated list.) Lists the ports on which the Kerberos server should listen for TCP connections, as a comma-separated list of integers. If this relation is not specified, the compiled-in default is not to listen for TCP connections at all.

If you wish to change this (note that the current implementation has little protection against denial-of-service attacks), the standard port number assigned for Kerberos TCP traffic is port 88.

**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 and salt 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.

**des_crc_session_supported** (Boolean value.) If set to true, the KDC will assume that service principals support des-cbc-crc for session key enctype negotiation purposes. If `allow_weak_crypto` in `[libdefaults]` is false, or if des-cbc-crc is not a permitted enctype, then this variable has no effect. Defaults to true.

**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.

**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-shal-96:normal aes128-cts-hmac-shal-96:normal des3-cbc-shal:normal arcfour-hmac-md5:normal`. For lists of possible values, see *Encryption and salt types*.

### [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_kadmind_dn`
- `ldap_service_password_file`
• ldap_servers
• ldap_conns_per_server

[dbmodules]

The [dbmodules] section contains parameters used by the KDC database library and database modules.

The following tag may be specified in the [dbmodules] section:

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

Other tags in the [dbmodules] section name a configuration subsection for parameters which can be referred to by a realm’s database_module parameter. The following tags may be specified in the subsection:

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

db_library  This tag indicates the name of the loadable database module. The value should be db2 for the DB2 module and 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 version 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 version 1.9.

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

ldap_kadmind_dn  This LDAP-specific tag indicates the default bind DN for the kadmind daemon. kadmind does a login to the directory as this object. This object should have the rights to read and write the Kerberos data in the LDAP database.

ldap_kdc_dn  This LDAP-specific tag indicates the default bind DN for the krb5kdc daemon. The KDC does a login to the directory as this object. This object should have the rights to read the Kerberos data in the LDAP database, and to write data unless disable_lockout and disable_last_success are true.

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_kadmind_dn and ldap_kdc_dn objects. This file must be kept secure.

[logging]

The [logging] section indicates how krb5kdc and kadmind perform logging. The keys in this section are daemon names, which may be one of:

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.

Values are of 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.

   The severity argument specifies the default severity of system log messages. This may be any of the following
   severities supported by the syslog(3) call, minus the LOG_ prefix: EMERG, ALERT, CRIT, ERR, WARNING,
   NOTICE, INFO, and DEBUG.

   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 severity is specified, the default is ERR. 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 with default severity of LOG_INFO; 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

2.2.3 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 cer-

              tificates. 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 number of bits the KDC is willing to accept for a client’s Diffie-Hellman key. The default is 2048.

pkinit_allow_upn  Specifies that the KDC is willing to accept client certificates with the Microsoft UserPrincipal-Name (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_eke_checking  This option specifies what Extended Key Usage (EKE) 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 EKE as defined in RFC 4556.

  scLogin  If scLogin is specified, client certificates with the Microsoft Smart Card Login EKE (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 EKE.

  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_kdc_ocsp  Specifies the location of the KDC’s OCSP.

pkinit_mapping_file  Specifies the name of the ACL pkinit mapping file. This file maps principals to the certificates that they can use.

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.

2.2.4 Encryption and salt 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” are available for compatibility but not recommended for use.
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 -des” would be the default set of encryption types with DES 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 AES keys in the KDC database.

Kerberos keys for users are usually derived from passwords. 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:

<table>
<thead>
<tr>
<th>Salt Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>default for Kerberos Version 5</td>
</tr>
<tr>
<td>v4</td>
<td>the only type used by Kerberos Version 4 (no salt)</td>
</tr>
<tr>
<td>noreal</td>
<td>same as the default, without using realm information</td>
</tr>
<tr>
<td>onlyreal</td>
<td>uses only realm information as the salt</td>
</tr>
<tr>
<td>afs3</td>
<td>AFS version 3, only used for compatibility with Kerberos 4 in AFS</td>
</tr>
<tr>
<td>special</td>
<td>generate a random salt</td>
</tr>
</tbody>
</table>

### 2.2.5 Sample kdc.conf File

Here’s an example of a kdc.conf file:

```
[kdcdefaults]
    kdc_ports = 88

[realms]
    ATHENA.MIT.EDU = {
        kadmnd_port = 749
    }
```
max_life = 12h 0m 0s
max_renewable_life = 7d 0h 0m 0s
master_key_type = des3-hmac-sha1
supported_enctypes = des3-hmac-sha1:normal des-cbc-crc:normal des-cbc-crc:v4
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
}

2.2.6 FILES
/etc/krb5kdc/kdc.conf

2.2.7 SEE ALSO
krb5.conf, krb5kdc, kadm5.acl

2.3 kadm5.acl

2.3.1 DESCRIPTION
The Kerberos kadm5 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 /etc/krb5kdc/kadm5.acl unless this is overridden by the acl_file variable in kdc.conf.

2.3.2 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.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Disallows the addition of principals or policies</td>
</tr>
<tr>
<td>c</td>
<td>Disallows the changing of passwords for principals</td>
</tr>
<tr>
<td>d</td>
<td>Disallows the deletion of principals or policies</td>
</tr>
<tr>
<td>i</td>
<td>Disallows inquiries about principals or policies</td>
</tr>
<tr>
<td>l</td>
<td>Disallows the listing of principals or policies</td>
</tr>
<tr>
<td>m</td>
<td>Disallows the modification of principals or policies</td>
</tr>
<tr>
<td>p</td>
<td>Disallows the propagation of the principal database (used in Incremental database propagation)</td>
</tr>
<tr>
<td>s</td>
<td>Disallows the explicit setting of the key for a principal</td>
</tr>
<tr>
<td>x</td>
<td>Short for admcil. All privileges</td>
</tr>
<tr>
<td>*</td>
<td>Same as x.</td>
</tr>
</tbody>
</table>

**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 component number 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 the + and - flags for the kadmin add_principal and modify_principal commands.
- **-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.

### 2.3.3 EXAMPLE

Here is an example of a kadm5.acl file.

```plaintext
*/admin@ATHENA.MIT.EDU * # line 1
joeadmin@ATHENA.MIT.EDU ADMCIL # line 2
joeadmin/*@ATHENA.MIT.EDU il */root@ATHENA.MIT.EDU # line 3
*/root@ATHENA.MIT.EDU cil *l@ATHENA.MIT.EDU # line 4
```
(line 1) Any principal in the ATHENA.MIT.EDU realm with an admin instance has all administrative privileges.

(lines 1-3) The user joeadmin has all permissions 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 and list permissions with any principal that has the instance root (matches line 3).

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

(line 5) Any principal in the realm ATHENA.MIT.EDU (except for joeadmin@ATHENA.MIT.EDU, as mentioned above) has inquire privileges.

(line 6) Finally, any principal with an admin instance in EXAMPLE.COM has all permissions, but any principal that they create or modify will not be able to get postdateable tickets or tickets with a life of longer than 9 hours.

2.3.4 SEE ALSO

kdc.conf, kadminl
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 kadmind services will use, if they will not be using the default ports.
- How many slave KDCs you need and where they should be located.
- The hostnames of your master and slave KDCs.
- How frequently you will propagate the database from the master KDC to the slave 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 Slave KDCs

Slave KDCs provide an additional source of Kerberos ticket-granting services in the event of inaccessibility of the master KDC. The number of slave 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 slave KDC to take up the slack.

Some considerations include:

- Have at least one slave KDC as a backup, for when the master 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 slave KDC accessible to each segment.
- If possible, have at least one slave KDC in a different building from the master, 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 master KDC and `kerberos-1`, `kerberos-2`, ... for the slave 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. The MIT KDC by default will not listen on any TCP ports, so unless you’ve changed the configuration or you’re running another KDC implementation, you should leave this unspecified. If you do enable TCP support, normally you should use port 88.

**_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 master 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 master 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`.

**_kpasswd._udp** This should list port 464 on your master 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 749.

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:

```plaintext
$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
_kerberos._udp SRV 0 0 88 use-the-force-luke
_gerberos._udp SRV 0 0 88 bunny-rabbit
_gerberos-master._udp SRV 0 0 88 daisy
_gerberos-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.

## 3.6 Database propagation

The Kerberos database resides on the master KDC, and must be propagated regularly (usually by a cron job) to the slave 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 slaves, 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 master KDC propagate the database to one set of slaves, and then have each of these slaves propagate the database to additional slaves.

See also *Incremental database propagation*
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/HOST (where HOST is the hostname of the admin server) or kadmin/admin. 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 kadmin options

You can invoke kadmin or kadmin.local with any of the following options:


- **kadmin.local** [-r realm] [-p principal] [-q query] [-d dbname] [-e enc:salt ...] [-m] [-x db_args]

**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.

- `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/ADMINHOST (where `ADMINHOST` is the fully-qualified hostname of the admin server) or kadmin/admin 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. This can be useful for writing scripts.

- `-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 list of encryption types and salt types to be used for any new keys created. See Encryption and salt types 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. Options supported for the LDAP database module are:
  
  - `-x host=hostname` Specifies the LDAP server to connect to by a LDAP URI.
  
  - `-x binddn=bind_dn` Specifies the DN of the object used by the administration server to bind to the LDAP server. This object should have the read and write privileges on the realm container, the principal container, and the subtree that is referenced by the realm.
  
  - `-x bindpwd=bind_password` Specifies the password for the above mentioned binddn. 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`.

4.2 Date Format

For the supported date-time formats see `getdate` section in `datetime`.

4.3 Principals

Each entry in the Kerberos database contains a Kerberos principal and the attributes and policies associated with that principal.
4.3.1 Adding, modifying and deleting principals

To add a principal to the database, use the `kadmin add_principal` command.

To modify attributes of a principal, use the `kadmin modify_principal` command.

To delete a principal, use the `kadmin delete_principal` command.

4.3.2 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` (getdate string) The maximum ticket life for the principal.
- `-maxrenewlife maxrenewlife` (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 this principal from obtaining a session key for another user. `+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.
- `{|-}requires_hwauth +requires_hwauth` requires this principal to preauthenticate using a hardware device before being allowed to `kinit`. `-requires_hwauth` clears this flag.
- `{|-}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. `+allow_svr` clears this flag.

4.3. Principals
{-|+}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.

-randkey Sets the key of the principal to a random value.

-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 list of enctype-salttype pairs for setting the key of the principal.

-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
  WARNING: 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:
```

4.3.3 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.

### 4.3.4 delete_principal

**delete_principal** [-force] **principal**

Deletes the specified **principal** from the database. This command prompts for deletion, unless the **-force** option is given.

This command requires the **delete** privilege.

Alias: delprinc

#### Examples

If you want to create a principal which is contained by a LDAP object, all you need to do is:

```bash
cadmin: addprinc -x dn=cn=jennifer,dc=example,dc=com jennifer
WARNING: no policy specified for "jennifer@ATHENA.MIT.EDU";
defaulting to no policy.
Enter password for principal jennifer@ATHENA.MIT.EDU: <= Type the password.
Re-enter password for principal jennifer@ATHENA.MIT.EDU: <=Type it again.
Principal "jennifer@ATHENA.MIT.EDU" created.
cadmin:
```

If you want to create a principal under a specific LDAP container and link to an existing LDAP object, all you need to do is:

```bash
cadmin: addprinc -x containerdn=dc=example,dc=com -x linkdn=cn=david,dc=example,dc=com david
WARNING: no policy specified for "david@ATHENA.MIT.EDU";
defaulting to no policy.
Enter password for principal david@ATHENA.MIT.EDU: <= Type the password.
Re-enter password for principal david@ATHENA.MIT.EDU: <=Type it again.
Principal "david@ATHENA.MIT.EDU" created.
cadmin:
```

If you want to associate a ticket policy to a principal, all you need to do is:

```bash
cadmin: modprinc -x tktpolicy=userpolicy david
Principal "david@ATHENA.MIT.EDU" modified.
cadmin:
```

If, on the other hand, you want to set up an account that expires on January 1, 2000, that uses a policy called "stduser", with a temporary password (which you want the user to change immediately), you would type the following:

```bash
cadmin: addprinc david -expire "1/1/2000 12:01am EST" -policy stduser +needchange
Enter password for principal david@ATHENA.MIT.EDU: <= Type the password.
Re-enter password for principal david@ATHENA.MIT.EDU: <=Type it again.
Principal "david@ATHENA.MIT.EDU" created.
cadmin:
```

If you want to delete a principal

```bash
cadmin: delprinc jennifer
Are you sure you want to delete the principal "jennifer@ATHENA.MIT.EDU"? (yes/no): yes
Principal "jennifer@ATHENA.MIT.EDU" deleted.
cadmin:
```
Make sure that you have removed this principal from all ACLs before reusing.

Kadmin:

### 4.3.5 Retrieving information about a principal

To retrieve a listing of the attributes and/or policies associated with a principal, use the `kadmin get_principal` command.

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

### 4.3.6 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 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: [none]
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: 2
  Key: vno 1, DES cbc mode with CRC-32, no salt
  Key: vno 1, DES cbc mode with CRC-32, Version 4
Attributes:
  Policy: [none]
```

```
kadmin: getprinc -terse systest
systest@BLEEP.COM 386400 604800 1
785926535 753241234 785900000
tlyu/admin@BLEEP.COM 786100034 0 0
kadmin:
```

### 4.3.7 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, get_princs

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:
```

## 4.3.8 Changing passwords

To change a principal’s password use the kadmin change_password command.

### 4.3.9 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 enctype-salt** Uses the specified list of enctype-salttype pairs for setting the key of the principal.
- **-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:
```

**Note:** Password changes through kadmin are subject to the same password policies as would apply to password changes through kpasswd(1).

## 4.4 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.
4.4.1 Adding, modifying and deleting policies

To add a new policy, use the `kadmin add_policy` command.
To modify attributes of a principal, use the `kadmin modify_policy` command.
To delete a policy, use the `kadmin delete_policy` command.

4.4.2 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` (getdate string) Sets the maximum lifetime of a password.
- `-minlife time` (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` (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` (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 Encryption and salt types in `kdc.conf` for a list of the accepted values, but note that key/salt tuples must be separated with commas (‘,’) only. To clear the allowed key/salt policy use a value of ‘-‘.

Example:

```
kadmin: add_policy -maxlife "2 days" -minlength 5 guests
kadmin: 
```

4.4.3 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.
4.4.4 delete_policy

del_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:

Note: You must cancel the policy from all principals before deleting it. The delete_policy command will fail if the policy is in use by any principals.

4.4.5 Retrieving policies

To retrieve a policy, use the kadmin get_policy command.

You can retrieve the list of policies with the kadmin list_policies command.

4.4.6 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.
4.4.7 list_policies

```plaintext
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:

```plaintext
kadmin: listpols
test-pol
dict-only
once-a-min
test-pol-nopw

kadmin: listpols t*
test-pol
test-pol-nopw
kadmin:
```

4.4.8 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:

```plaintext
cadmin: 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.

4.5 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.

4.6 Operations on the Kerberos database

The `kdb5_util` command is the primary tool for administrating the Kerberos database.

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.

### 4.6.1 Dumping a Kerberos database to a file

To dump a Kerberos database into a file, use the `kdb5_util dump` command on one of the KDCs.

```
```

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 6”. If filename is not specified, or is the string “-”, the dump is sent to standard output. Options:

- **-old** causes the dump to be in the Kerberos 5 Beta 5 and earlier dump format (“kdb5_edit load_dump version 2.0”).

- **-b6** causes the dump to be in the Kerberos 5 Beta 6 format (“kdb5_edit load_dump version 3.0”).

- **-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.

- **-ov** causes the dump to be in “ovsec_adm_export” format.

- **-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.

### 4.6. Operations on the Kerberos database
-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.

Examples

```
shell% kdb5_util dump dumpfile
```

```
shell% kdb5_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
```

If you specify which principals to dump, you must use the full principal, as in the following example:

```
shell% kdb5_util dump -verbose dumpfile K/M@ATHENA.MIT.EDU kadmin/admin@ATHENA.MIT.EDU
```

```
shell% kdb5_util dump -verbose dumpfile K/M@ATHENA.MIT.EDU kadmin/admin@ATHENA.MIT.EDU
kadmin/admin@ATHENA.MIT.EDU
K/M@ATHENA.MIT.EDU
```

If you do not specify a dump file, kdb5_util will dump the database to the standard output.

### 4.6.2 Restoring a Kerberos database from a dump file

To restore a Kerberos database dump from a file, use the `kdb5_util load` command on one of the KDCs.

```
```

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:

- **-old** requires the database to be in the Kerberos 5 Beta 5 and earlier format (“kdb5_edit load_dump version 2.0”).
- **-b6** requires the database to be in the Kerberos 5 Beta 6 format (“kdb5_edit load_dump version 3.0”).
- **-b7** requires the database to be in the Kerberos 5 Beta 7 format (“kdb5_util load_dump version 4”).
- **-ov** requires the database to be in “ovsec_adm_import” format. Must be used with the -update option.
- **-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** requires the database to be stored as a hash. If this option is not specified, the database will be stored as a btree. 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. (This is useful in conjunction with an ovsec_adm_export format dump if you want to preserve per-principal policy information, since the current default format does not contain this data.) Otherwise, a new database is created containing only what is in the dump file and the old one destroyed upon successful completion.

If specified, **dbname** overrides the value specified on the command line or the default.

**Examples**

To load a single principal, either replacing or updating the database:

shell% kdb5_util load dumpfile principal
shell%

shell% kdb5_util load -update dumpfile principal
shell%

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

Using **kdb5_util** to upgrade a master KDC from krb5 1.1.x:

shell% kdb5_util dump old-kdb-dump
shell% kdb5_util dump -ov old-kdb-dump.ov
   [Create a new KDC installation, using the old stash file/master password]
shell% kdb5_util load old-kdb-dump
shell% kdb5_util load -update old-kdb-dump.ov

The use of old-kdb-dump.ov for an extra dump and load is necessary to preserve per-principal policy information, which is not included in the default dump format of krb5 1.1.x.

**Note:** Using **kdb5_util** to dump and reload the principal database is only necessary when upgrading from versions of krb5 prior to 1.2.0—newer versions will use the existing database as-is.

### 4.6.3 Creating a stash file

A stash file allows a KDC to authenticate itself to the database utilities, such as **kadmind**, **krb5kdc**, and **kdb5_util**. To create a stash file, use the **kdb5_util stash** command.

    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**.

**Example**

    shell% 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. shell%
If you do not specify a stash file, kdb5_util will stash the key in the file specified in your \texttt{kdc.conf} file.

### 4.6.4 Creating and destroying a Kerberos database

If you need to create a new Kerberos database, use the \texttt{kdb5_util create} command.

\begin{verbatim}
create [-s]
\end{verbatim}

Creates a new database. If the \texttt{-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.

If you need to destroy the current Kerberos database, use the \texttt{kdb5_util destroy} command.

\begin{verbatim}
destroy [-f]
\end{verbatim}

Destroys the database, first overwriting the disk sectors and then unlinking the files, after prompting the user for confirmation. With the \texttt{-f} argument, does not prompt the user.

**Examples**

\begin{verbatim}
shell% kdb5_util -r ATHENA.MIT.EDU create -s
Loading random data
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%

shell% kdb5_util -r ATHENA.MIT.EDU destroy
Deleting KDC database stored in `/usr/local/var/krb5kdc/principal’, are you sure?
(type ’yes’ to confirm)? <= yes
OK, deleting database '/usr/local/var/krb5kdc/principal’...
** Database '/usr/local/var/krb5kdc/principal’ destroyed.
shell%
\end{verbatim}

### 4.7 Operations on the LDAP database

The \texttt{kdb5_ldap_util} is the primary tool for administrating the Kerberos LDAP database. It allows an administrator to manage realms, Kerberos services (KDC and Admin Server) and ticket policies.

\begin{verbatim}
kdb5_ldap_util [-D user_dn [-w passwd]] [-H ldapuri] command [command_options]
\end{verbatim}

**OPTIONS**

\texttt{-D user_dn} Specifies the Distinguished Name (DN) of the user who has sufficient rights to perform the operation on the LDAP server.

\texttt{-w passwd} Specifies the password of \texttt{user_dn}. This option is not recommended.

\texttt{-H ldapuri} Specifies the URI of the LDAP server. It is recommended to use \texttt{ldapi://} or \texttt{ldaps://} to connect to the LDAP server.
4.7.1 Creating a Kerberos realm

If you need to create a new realm, use the `kdb5_ldap_util create` command as follows.

```
create [-subtrees subtree_dn_list] [-sscope search_scope] [-containerref container_reference_dn]
```

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` 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.

- `-r realm` Specifies the Kerberos realm of the database.

- `-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
              create -subtrees o=org -sscope SUB -r ATHENA.MIT.EDU
```

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:
```

4.7.2 Modifying a Kerberos realm

If you need to modify a realm, use the `kdb5_ldap_util modify` command as follows.

```
```

4.7. Operations on the LDAP database 53
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.

- **r** _realm_ Specifies the Kerberos realm of the database.

- **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 -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu modify +requires_preauth -r ATHENA.MIT.EDU
Password for "cn=admin,o=org":
```

### 4.7.3 Destroying a Kerberos realm

If you need to destroy a Kerberos realm, use the _kdb5_ldap_util destroy command as follows.

```
destroy [-f] [-r realm]
```

Destroys an existing realm. Options:

- **-f** If specified, will not prompt the user for confirmation.

- **-r** _realm_ Specifies the Kerberos realm of the database.

Example:

```
shell% kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu destroy -r ATHENA.MIT.EDU
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'...
```

### 4.7.4 Retrieving information about a Kerberos realm

If you need to display the attributes of a realm, use the _kdb5_ldap_util view command as follows.

```
view [-r realm]
```

Displays the attributes of a realm. Options:

- **-r** _realm_ Specifies the Kerberos realm of the database.

Example:
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
  view -r ATHENA.MIT.EDU
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

4.7.5 Listing available Kerberos realms

If you need to display the list of the realms, use the kdb5_ldap_util list command as follows.

   list

Lists the name of realms.

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%

4.7.6 Stashing service object’s password

The kdb5_ldap_util stashsrvpw command allows an administrator to store the password of service object in a file. The KDC and Administration server uses this password to authenticate to the LDAP server.

    stashsrvpw [-f filename] servicedn

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.

    servicedn Specifies Distinguished Name (DN) of the service object whose password is to be stored in file.

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":

4.7.7 Ticket Policy operations

Creating a Ticket Policy

To create a new ticket policy in directory , use the kdb5_ldap_util create_policy command. Ticket policy objects are created under the realm container.
**create_policy** [-r *realm*] [-maxktlifemax_ticket_life] [-maxrenewlife max_renewable_ticket_life] [ticket_flags] policy_name

Creates a ticket policy in the directory. Options:

- **-r** *realm* Specifies the Kerberos realm of the database.

- **-maxktlifemax_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:

```
kbd5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
create_policy -r ATHENA.MIT.EDU -maxktlifemax_ticket_life "1 day"
-maxrenewlife "1 week" -allow_postdated +needchange
-allow_forwardable tktpolicy
```

**Modifying a Ticket Policy**

To modify a ticket policy in directory, use the **kbd5_ldap_util modify_policy** command.

```
modify_policy [-r *realm*] [-maxktlifemax_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:

```
kbd5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
modify_policy -r ATHENA.MIT.EDU -maxktlifemax_ticket_life "60 minutes"
-maxrenewlife "10 hours" -allow_postdated -requires_preauth tktpolicy
```

**Retrieving Information About a Ticket Policy**

To display the attributes of a ticket policy, use the **kbd5_ldap_util view_policy** command.

```
view_policy [-r *realm*] policy_name
```

Displays the attributes of a ticket policy. Options:

- **policy_name** Specifies the name of the ticket policy.

Example:

```
kbd5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
view_policy -r ATHENA.MIT.EDU 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
Destroying a Ticket Policy

To destroy an existing ticket policy, use the `kdb5_ldap_util destroy_policy` command.

```
destroy_policy [-r realm] [-force] policy_name
```

Destroys an existing ticket policy. Options:

- `-r realm` Specifies the Kerberos realm of the database.
- `-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
destroy_policy -r ATHENA.MIT.EDU 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.

Listing available Ticket Policies

To list the name of ticket policies in a realm, use the `kdb5_ldap_util list_policy` command.

```
list_policy [-r realm]
```

Lists the ticket policies in realm if specified or in the default realm. Options:

- `-r realm` Specifies the Kerberos realm of the database.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
list_policy -r ATHENA.MIT.EDU
```

Password for "cn=admin,o=org":

tktpolicy
tmppolicy
userpolicy

4.8 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:

cadmin: 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:

cadmin:

**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.

### 4.9 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 `cadmin 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:

```
cadmin: 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 cadmin `purgekeys` command to delete keys with older kvnos, ideally first making sure that all tickets issued with the old keys have expired.

### 4.10 Incremental database propagation

#### 4.10.1 Overview

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

With incremental propagation enabled, all programs on the master 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 slave KDC connects to a service on the master 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. If the database has just been modified in the previous several seconds (currently the threshold is hard-coded at 10 seconds), the slave will not retrieve updates, but instead will pause and try again soon after. This reduces the likelihood that incremental update queries will cause delays for an administrator trying to make a bunch of changes to the database at the same time.

Incremental propagation uses the following entries in the per-realm data in the KDC config file (See `kdc.conf`):
Both master and slave 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 (`FILE:/etc/krb5.keytab`).

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

The normal kprop mechanism is disabled by the incremental propagation support. However, if the slave has been unable to fetch changes from the master KDC for too long (network problems, perhaps), the log on the master may wrap around and overwrite some of the updates that the slave has not yet retrieved. In this case, the slave will instruct the master 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 slave should resume fetching incremental updates. Thus, all the keytab and ACL setup previously described for kprop propagation is still needed.

There are several known bugs and restrictions in the current implementation:

- The “call out to kprop” mechanism is a bit fragile; if the kprop propagation fails to connect for some reason, the process on the slave may hang waiting for it, and will need to be restarted.
- The master and slave must be able to initiate TCP connections in both directions, without an intervening NAT.

### 4.10.2 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 master and slave sides.
The Sun implementation hard-codes pathnames in /var/krb5 for the update log and the per-slave kprop dump files. In the MIT implementation, the pathname for the update log is specified in the config file, and the per-slave dump files are stored in /etc/krb5kdc/slave_datatrans_hostname.
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.

5.1 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
```

5.2 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
Password for user@KRBTEST.COM:
kinit: Password incorrect while getting initial credentials
```

$ kinit user
kinit: Clients credentials have been revoked while getting initial credentials

5.3 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.

5.4 KDC replication and account lockout

The account lockout state of a principal is not replicated between KDCs. 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 master and three slaves), 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 master to the slave KDCs during the next propagation. Propagation of an administrative unlock will cause the counter of failed attempts on each slave to reset to 1 on the next failure.

5.5 KDC performance and account lockout

In order to fully track account lockout state, the KDC must write to 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
```
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.
CHAPTER SIX

CONFIGURING KERBEROS WITH OPENLDAP BACK-END

1. Set up SSL on the OpenLDAP server and client to ensure secure communication when the KDC service and LDAP server are on different machines. ldapi:// can be used if the LDAP server and KDC service are running on the same machine.

   (a) Setting up SSL on the OpenLDAP server:

   (a) Get a CA certificate using OpenSSL tools

   (b) Configure OpenLDAP server for using SSL/TLS

       For the latter, you need to specify the location of CA certificate location in slapd.conf file.

       Refer to the following link for more information: http://www.openldap.org/doc/admin23/tls.html

   (a) Setting up SSL on OpenLDAP client:

       i. For the KDC and Admin Server, you need to do the client-side configuration in ldap.conf. For example:

       TLS_CACERT /etc/openldap/certs/cacert.pem

2. Include the Kerberos schema file (kerberos.schema) in the configuration file (slapd.conf) on the LDAP Server, by providing the location where it is stored:

   include /etc/openldap/schema/kerberos.schema

3. Choose DNs for the krb5kdc and kadmin servers to bind to the LDAP server, and create them if necessary. These DNs will be specified with the ldap_kdc_dn and ldap_kadmind_dn directives in kdc.conf; their passwords can be stashed with “kdb5_ldap_util stashsrvpw” and the resulting file specified with the ldap_service_password_file directive.

4. Choose a DN for the global Kerberos container entry (but do not create the entry at this time). This DN will be specified 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.

5. Configure the LDAP server ACLs to enable the KDC and kadmin server DNs to read and write the Kerberos data.

   Sample access control information:
access to dn.base="" 
  by * read 

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

access to attrs=userPassword,userPKCS12 
  by self write 
  by * auth 

access to attrs=shadowLastChange 
  by self write 
  by * read 

# Providing access to 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" read 
  by dn.exact="cn=adm-service,dc=example,dc=com" write 
  by * none 

# Providing access to principals, if not underneath realm container 
access to dn.subtree= "ou=users,dc=example,dc=com" 
  by dn.exact="cn=kdc-service,dc=example,dc=com" read 
  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.

6. Start the LDAP server as follows:  
   slapd -h "ldapi:/// ldaps:///"

7. Modify the kdc.conf file to include LDAP specific items listed below: 
   
   realms 
       database_module 

   dbmodules 
       db_library 
       db_module_dir 
       ldap_kdc_dn 
       ldap_kadmind_dn 
       ldap_service_password_file 
       ldap_servers 
       ldap_conns_per_server 

8. Create the realm using kdb5_ldap_util (see Creating a Kerberos realm): 
   
   kdb5_ldap_util -D cn=admin,dc=example,dc=com create -subtrees ou=users,dc=example,dc=com -r EXAMPLE.COM 

   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 will be used to store information related to all realms.

9. Stash the password of the service object used by the KDC and Administration service to bind to the LDAP server using the `kdb5_ldap_util stashsrvpw` command (see Stashing service object’s password). The object DN should be the same as `ldap_kdc_dn` and `ldap_kadmind_dn` values specified in the `kdc.conf` file:

   ```
   kdb5_ldap_util -D cn=admin,dc=example,dc=com stashsrvpw -f /etc/kerberos/service.keyfile cn=krbadmin,dc=example,dc=com
   ```

10. Add `krbPrincipalName` to the indexes in slapd.conf to speed up the access.

With the LDAP back end it is possible to provide aliases for principal entries. Currently we provide no mechanism provided for creating aliases, so it must be done by 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.

**See Also:**

`LDAP backend on Ubuntu 10.4 (lucid)"
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 Adding, modifying and deleting 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.

7.1 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.

7.1.1 Adding principals to keytabs

To generate a keytab, or to add a principal to an existing keytab, use the ktadd command from kadmin.

7.1.2 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 encisalt,... Use the specified list of enctype-saltpassword pairs for setting the new keys of the principal.
-q Display less verbose information.
-norandkey Do not randomize the keys. The keys and their version numbers stay unchanged. This option is only available in kadmin.local, and 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.

Example:

```
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: 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

```
kadmin: ktadd host/daffodil.mit.edu@ATHENA.MIT.EDU
```

Entry for principal host/daffodil.mit.edu with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab

**Examples**

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

```
kadmin: ktadd host/daffodil.mit.edu@ATHENA.MIT.EDU
```

Entry for principal host/daffodil.mit.edu with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab FILE:/etc/krb5.keytab

7.1.3 Removing principals from keytabs

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

7.1.4 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** keytab  Use keytab as the keytab file. Otherwise, the default keytab is used.
- **-q** Display less verbose information.

Example:

```
kadmin: ktremove kadmin/admin all
```

Entry for principal kadmin/admin with kvno 3 removed from keytab FILE:/etc/krb5.keytab

7.2 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].
7.3 Getting DNS information correct

Several aspects of Kerberos rely on name service. When a hostname is used to name a service, the Kerberos library canonicalizes the hostname using forward and reverse name resolution. (The reverse name resolution step can be turned off using the rdns variable in [libdefaults].) The result of this canonicalization must match the principal entry in the host’s keytab, or authentication will fail.

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:

<table>
<thead>
<tr>
<th>IP address</th>
<th>fully-qualified hostname</th>
<th>aliases</th>
</tr>
</thead>
</table>

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.

7.4 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 slave 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.
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.

### 8.1 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 `otp` installed at `/path/to/otp.so`, you could write:

```conf
[plugins]
  clpreauth = {
    module = otp:/path/to/otp.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:

```conf
[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:

```conf
[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.

### 8.1.1 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 `/usr/lib/x86_64-linux-gnu/krb5/plugins`. For example, Samba’s winbind krb5 locator plugin would be registered by placing its shared object in `/usr/lib/x86_64-linux-gnu/krb5/plugins/libkrb5/winbind_krb5_locator.so`.

### 8.1.2 GSSAPI mechanism modules

GSSAPI mechanism module are registered using the file `/etc/gss/mech`. Each line in this file has the form:

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

Only the oid and pathname are required. `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.

### 8.1.3 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.
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.

9.1 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 slave KDCs, you might never notice, since the next automatic propagation of the database would install a fresh copy. However, if it happens to the master KDC, the corrupted database would be propagated to all of the slaves during the next propagation. For this reason, MIT recommends that you back up your Kerberos database regularly. Because the master KDC is continuously dumping the database to a file in order to propagate it to the slave 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 master KDC. (See Restoring a Kerberos database from a dump file.)
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.

Configuring PKINIT requires establishing a certificate authority (or using an existing one), and using the authority to sign certificates for the KDC and for each client principal. These instructions will describe how to generate the necessary certificates using OpenSSL, and then explain how to configure the KDC and clients once the certificates are in hand.

10.1 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
```

The second command will ask for the values of several certificate fields. These fields can be set to any values.

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.

10.2 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
```

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

```bash
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 \
    -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.

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.

### 10.3 Generating client certificates

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

```bash
[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:

```bash
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 \
```
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.

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.

### 10.4 Configuring the KDC

The KDC must have filesystem access to the CA certificate (cacert.pem), the KDC certificate (kdc.pem), and the KDC private key (kdckey.pem). Configure the following relations in the KDC’s kdc.conf file, either in the [kdcdefaults] section or in a [realms] subsection:

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

Adjust the pathnames to match the paths of the three files. Because of the larger size of requests and responses using PKINIT, you may also need to allow TCP access to the KDC:

```
kdc_tcp_ports = 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'
```

### 10.5 Configuring the clients

To perform PKINIT authentication, a client host must have filesystem access to the CA certificate (cacert.pem), the client certificate (client.pem), and the client private key (clientkey.pem). Configure the following relations in the client host’s krb5.conf file in the appropriate [realms] subsection:

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

Adjust the pathnames to match the paths of the three files.

If the KDC and client are properly configured, it should now be possible to run kinit username without entering a password.

### 10.6 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 follow the steps above for generating a KDC certificate and configuring 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 cacert.pem file if you won’t have any client certificates to verify. On client hosts, you must store the cacert.pem file and set the `pkinit_anchors` variable in order to verify 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`. 

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.

11.1 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.

11.2 Service principal canonicalization

MIT Kerberos clients currently always do forward resolution (looking up the IPv4 and possibly IPv6 addresses using getaddrinfo()) of the hostname part of a host-based service principal to canonicalize the hostname. They obtain the “canonical” name of the host when doing so. By default, MIT Kerberos clients will also then do reverse DNS resolution (looking up the hostname associated with the IPv4 or IPv6 address using getnameinfo()) of the hostname. Using the krb5.conf setting

```
[libdefaults]
rdns = false
```

will disable reverse DNS lookup on clients. The default setting is “true”.

Operating system bugs may prevent a setting of rdns = false from disabling reverse DNS lookup. Some versions of GNU libc have a bug in getaddrinfo() that cause them to look up PTR records even when not required. MIT Kerberos releases krb5-1.10.2 and newer have a workaround for this problem, as does the krb5-1.9.x series as of release krb5-1.9.4.
11.3 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.

11.4 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.

11.5 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.

11.6 Specific application advice

11.6.1 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.
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.

### 12.1 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.

### 12.2 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. If allow_weak_crypto is true, all services are assumed to support des-cbc-crc.

Starting in krb5-1.11, des_crc_session_supported in kdc.conf allows additional control over whether the KDC issues des-cbc-crc session keys.

Also 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. See set_string in kadmin for details.
12.3 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.

12.4 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 single-DES enctypes (and other 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.

- `permitted_enctypes` controls the set of enctypes that a service will accept as session keys.

- `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

12.5 Enctype compatibility

See Encryption and salt types for additional information about enctypes.

<table>
<thead>
<tr>
<th>enctype</th>
<th>weak?</th>
<th>krb5</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>des-cbc-crc</td>
<td>weak</td>
<td>all</td>
<td>&gt;=2000</td>
</tr>
<tr>
<td>des-cbc-md4</td>
<td>weak</td>
<td>all</td>
<td>?</td>
</tr>
<tr>
<td>des-cbc-md5</td>
<td>weak</td>
<td>all</td>
<td>&gt;=2000</td>
</tr>
<tr>
<td>des3-cbc-sha1</td>
<td>&gt;=1.1</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>arcfour-hmac</td>
<td>&gt;=1.3</td>
<td>&gt;=2000</td>
<td></td>
</tr>
<tr>
<td>arcfour-hmac-exp</td>
<td>weak</td>
<td>&gt;=1.3</td>
<td>&gt;=2000</td>
</tr>
<tr>
<td>aes128-cts-hmac-sha1-96</td>
<td>&gt;=1.3</td>
<td>&gt;=Vista</td>
<td></td>
</tr>
<tr>
<td>aes256-cts-hmac-sha1-96</td>
<td>&gt;=1.3</td>
<td>&gt;=Vista</td>
<td></td>
</tr>
<tr>
<td>camellia128-cts-cmac</td>
<td>&gt;=1.9</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>camellia256-cts-cmac</td>
<td>&gt;=1.9</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

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.
13.1 kadmin

13.1.1 SYNOPSIS

kadmin [-O|-N] [-r realm] [-p principal] [-q query] [ [-c cache_name][ -k [ -t keytab]]] -n] [-w password] [-s admin_server[:port]]

kadmin.local [-r realm] [-p principal] [-q query] [ -d dbname] [-e enc:salt ...] [-m] [-x db_args]

13.1.2 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/ADMINHOST (where ADMINHOST is the fully-qualified hostname of the admin server) or kadmin/admin. 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 master 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.

13.1.3 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.

credentials_cache Use credentials_cache as the credentials cache. The cache should contain a service ticket for the kadmin/ADMINHOST (where ADMINHOST is the fully-qualified hostname of the admin server) or kadmin/admin service; it can be acquired with the kinit() program. If this option is not specified, kadmin requests a new service ticket from the KDC, and stores it in its own temporary cache.

-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. This can be useful for writing scripts.

dbname Specifies the name of the KDC database. This option does not apply to the LDAP database module.

-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 list of encryption types and salt types to be used for any new keys created. See Encryption and salt types 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.

db_args Specifies the database specific arguments. Options supported for the LDAP database module are:

-x host=hostname Specifies the LDAP server to connect to by a LDAP URI.

-x binddn=bind_dn Specifies the DN of the object used by the administration server to bind to the LDAP server. This object should have the read and write privileges on the realm container, the principal container, and the subtree that is referenced by the realm.

-x bindpwd=bind_password Specifies the password for the above mentioned binddn. 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.

13.1.4 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.
- **-pwexpiry pwexpdate** (*getdate* string) The password expiration date.
- **-maxlife maxlife** (*getdate* string) The maximum ticket life for the principal.
- **-maxrenewlife maxrenewlife** (*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 -allowDup_skey disables user-to-user authentication for this principal by prohibiting this principal from obtaining a session key for another user. +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.
{-|+}requires_hwauth +requires.hwauth requires this principal to preauthenticate using a hardware device before being allowed to kinit. -requires.hwauth clears this flag.
{-|+}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. +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.
```

- **-randkey** Sets the key of the principal to a random value.
- **-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 encsalt...** Uses the specified list of enctype-salttype pairs for setting the key of the principal.
- **-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:**
```bash
kadmin: addprinc jennifer
WARNING: 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**
delete_principal

deadline_principal [-force] principal

Deletes the specified principal from the database. This command prompts for deletion, unless the -force option is given.

This command requires the delete privilege.

Alias: delprinc

change_password

cchange_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 encsalt,...  Uses the specified list of entype-salttype pairs for setting the key of the principal.

- 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 [-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.

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: [none]
    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: 2
    Key: vno 1, DES cbc mode with CRC-32, no salt
    Key: vno 1, DES cbc mode with CRC-32, Version 4
    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, get_princs

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: getstr
set_string

    set_string principal key 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 attributes are recognized by the KDC:

**session_enctypes** Specifies the encryption types supported for session keys when the principal is authenticated to as a server. See *Encryption and salt types* in *kdc.conf* for a list of the accepted values.

This command requires the **modify** privilege.

Alias: setstr

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 (getdate string) Sets the maximum lifetime of a password.
- **minlife** time (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 (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 (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 Encryption and salt types in kdc.conf for a list of the accepted values, but note that key/salt tuples must be separated with commas (’,’) only. To clear the allowed key/salt policy use a value of ‘-’.

Example:

```
  kadmin: add_policy -maxlife "2 days" -minlength 5 guests
```

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
```

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
```
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
```

**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,...**  Use the specified list of enctype-salttype pairs for setting the new keys of the principal.
- **-q**  Display less verbose information.
- **-norandkey**  Do not randomize the keys. The keys and their version numbers stay unchanged. This option is only available in kadmin.local, and 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.

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-shal-96 added to keytab
```

13.1. **kadmin**
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.

Example:

```
kadmin: ktremlve 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**

### 13.1.5 HISTORY

The kadmin program was originally written by Tom Yu at MIT, as an interface to the OpenVision Kerberos administration program.
13.1.6 SEE ALSO

kpasswd(1), kadmind

13.2 kadmind

13.2.1 SYNOPSIS


13.2.2 DESCRIPTION

kadmind starts the Kerberos administration server. kadmind typically runs on the master 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 /etc/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 slave 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 kprop/MASTER\@REALM (where MASTER is the master KDC’s canonical host name, and REALM the realm name) to be registered in the database.

13.2.3 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. In normal operation, you should allow the server to place itself in the background.

-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.
-kprop_path specifies the path to the kprop command to use to send full dumps to slaves in response to full resync requests.

-d dump_file specifies the file path to be used for dumping the KDB in response to full resync requests when iprop is enabled.

-db_args specifies database-specific arguments.

Options supported for LDAP database are:

-x nconns=number_of_connections specifies the number of connections to be maintained per LDAP server.

-x host=ldapuri specifies the LDAP server to connect to by URI.

-x binddn=binddn specifies the DN of the object used by the administration server to bind to the LDAP server. This object should have read and write privileges on the realm container, the principal container, and the subtree that is referenced by the realm.

-x bindpwd=bind_password specifies the password for the above mentioned binddn. 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.

13.2.4 SEE ALSO

kpasswd(1), kadmin, kdb5_util, kdb5_ldap_util, kadm5.acl

13.3 kdb5_util

13.3.1 SYNOPSIS


13.3.2 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.

13.3.3 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.

13.3.4 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


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 6”. If filename is not specified, or is the string “-”, the dump is sent to standard output. Options:

-old  causes the dump to be in the Kerberos 5 Beta 5 and earlier dump format (“kdb5_edit load_dump version 2.0”).

-b6  causes the dump to be in the Kerberos 5 Beta 6 format (“kdb5_edit load_dump version 3.0”).

-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.

-ov  causes the dump to be in “ovsec_adm_export” format.
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.

load


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:

-old requires the database to be in the Kerberos 5 Beta 5 and earlier format ("kdb5_edit load_dump version 2.0").

-b6 requires the database to be in the Kerberos 5 Beta 6 format ("kdb5_edit load_dump version 3.0").

-b7 requires the database to be in the Kerberos 5 Beta 7 format ("kdb5_util load_dump version 4").

-ov requires the database to be in "ovsec_adm_import" format. Must be used with the -update option.

-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 requires the database to be stored as a hash. If this option is not specified, the database will be stored as a btree. 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. (This is useful in conjunction with an ovsec_adm_export format dump if you want to preserve per-principal policy information, since the current default format does not contain this data.) Otherwise, a new database is created containing only what is in the dump file and the old one destroyed upon successful completion.

If specified, dbname overrides the value specified on the command line or the default.
**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 and salt 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 slave servers via a manual or periodic invocation of kprop. Then, the stash files on the slave 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 older versions, 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.

13.3.5 SEE ALSO

kadmin

13.4 kdb5_ldap_util

13.4.1 SYNOPSIS

kdb5_ldap_util [-D user_dn [-w passwd]] [-H ldapuri] command [command_options]

13.4.2 DESCRIPTION

kdb5_ldap_util allows an administrator to manage realms, Kerberos services and ticket policies.

13.4.3 COMMAND-LINE OPTIONS

-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. It is recommended to use ldapi:// or ldaps:// to connect to the LDAP server.

13.4.4 COMMANDS

create


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 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.

-r realm Specifies the Kerberos realm of the database.

-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
create -subtrees o=org -sscope SUB -r ATHENA.MIT.EDU
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


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.

-r realm Specifies the Kerberos realm of the database.

-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 -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu modify +requires_preauth -r ATHENA.MIT.EDU
Password for "cn=admin,o=org":
shell%
```

**view**

```
view [-r realm]
```

Displays the attributes of a realm. Options:

- `-r realm` Specifies the Kerberos realm of the database.

Example:

```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu view -r ATHENA.MIT.EDU
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] [-r realm]
```

Destroys an existing realm. Options:

- `-f` If specified, will not prompt the user for confirmation.
- `-r realm` Specifies the Kerberos realm of the database.

Example:

```
shell% kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu destroy -r ATHENA.MIT.EDU
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 name of realms.

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] servicedn

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.

- **servicedn** Specifies Distinguished Name (DN) of the service object whose password is to be stored in file.

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 [-r realm] [-maxtktlife max_ticket_life] [-maxrenewlife max_renewable_ticket_life] [ticket_flags] policy_name

Creates a ticket policy in the directory. Options:

- **-r realm** Specifies the Kerberos realm of the database.

- **-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
    create_policy -r ATHENA.MIT.EDU -maxtktlife "1 day"
    -maxrenewlife "1 week" -allow_postdated +needchange
    -allow_forwardable tktpolicy
    Password for "cn=admin,o=org":

modify_policy

modify_policy [-r realm] [-maxtktlife max_ticket_life] [-maxrenewlife max_renewable_ticket_life] [ticket_flags] policy_name

13.4. kdb5_ldap_util 103
Modifies the attributes of a ticket policy. Options are same as for `create_policy`.

Example:

```bash
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu modify_policy -r ATHENA.MIT.EDU -maxtktlife "60 minutes" -maxrenewlife "10 hours" +allow_postdated -requires_preauth tktpolicy
```

Password for "cn=admin,o=org":

```
view_policy

** view_policy [-r realm] policy_name **
Displays the attributes of a ticket policy. Options:

policy_name  Specifies the name of the ticket policy.

Example:

```bash
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu view_policy -r ATHENA.MIT.EDU 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
```

```bash
destroy_policy

** destroy_policy [-r realm] [-force] policy_name **
Destroys an existing ticket policy. Options:

-r realm  Specifies the Kerberos realm of the database.
-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:

```bash
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu destroy_policy -r ATHENA.MIT.EDU 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.
```

```bash
list_policy

** list_policy [-r realm] **
Lists the ticket policies in realm if specified or in the default realm. Options:

-r realm  Specifies the Kerberos realm of the database.

Example:

```bash
```
kdb5_ldap_util -D cn=admin,o=org -H ldaps://ldap-server1.mit.edu
   list_policy -r ATHENA.MIT.EDU
Password for "cn=admin,o=org":
tktpolicy
tmppolicy
userpolicy

13.4.5 SEE ALSO

kadmin

13.5 krb5kdc

13.5.1 SYNOPSIS

   [-P pid_file] [-T time_offset]

13.5.2 DESCRIPTION

krb5kdc is the Kerberos version 5 Authentication Service and Key Distribution Center (AS/KDC).

13.5.3 OPTIONS

The -r realm option specifies the realm for which the server should provide service.

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 des-cbc-crc.

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. In normal operation, you should always allow the KDC to place itself in the background.

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 port numbers which the KDC should listen on for Kerberos version 5 requests, as a comma-separated list. This value overrides the UDP 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 ports are 88 and 750.

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.
Note: On operating systems which do not have `pktinfo` support, using worker processes will prevent the KDC from listening for UDP packets on network interfaces created after the KDC starts.

The `-x db_args` option specifies database-specific arguments. Options supported for the LDAP database module are:

- `-x nconns=<number_of_connections>` Specifies the number of connections to be maintained per LDAP server.
- `-x host=<ldapuri>` Specifies the LDAP server to connect to by URI.
- `-x binddn=<binddn>` Specifies the DN of the object used by the KDC server to bind to the LDAP server. This object should have read and write privileges to the realm container, the principal container, and the subtree that is referenced by the realm.
- `-x bindpwd=<bind_password>` Specifies the password for the above mentioned binddn. 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`.

The `-T offset` option specifies a time offset, in seconds, which the KDC will operate under. It is intended only for testing purposes.

### 13.5.4 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:

```bash
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.

### 13.5.5 ENVIRONMENT

`krb5kdc` uses the following environment variables:

- `KRB5_CONFIG`
- `KRB5_KDC_PROFILE`

### 13.5.6 SEE ALSO

`kdb5_util`, `kdc.conf`, `krb5.conf`, `kdb5_ldap_util`

### 13.6 kprop

#### 13.6.1 SYNOPSIS

13.6.2 DESCRIPTION

kprop is used to securely propagate a Kerberos V5 database dump file from the master Kerberos server to a slave Kerberos server, which is specified by slave_host. The dump file must be created by kdb5_util.

13.6.3 OPTIONS

- `-r realm` Specifies the realm of the master server.
- `-f file` Specifies the filename where the dumped principal database file is to be found; by default the dumped database file is normally /etc/krb5kdc/slave_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.

13.6.4 ENVIRONMENT

Kprop uses the following environment variable:

- `KR5_CONFIG`

13.6.5 SEE ALSO

kpropd, kdb5_util, krb5kdc

13.7 kpropd

13.7.1 SYNOPSIS


13.7.2 DESCRIPTION

The kpropd command runs on the slave KDC server. It listens for update requests made by the kprop program. If incremental propagation is enabled, it periodically requests incremental updates from the master KDC.

When the slave receives a kprop request from the master, 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 master Kerberos server to use kprop to propagate its database to the slave servers. Upon a successful download of the KDC database file, the slave 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. This is required for incremental propagation. But this is also useful for debugging purposes.

Incremental propagation may be enabled with the `iprop_enable` variable in `kdc.conf`. If incremental propagation is enabled, the slave periodically polls the master KDC for updates, at an interval determined by the `iprop_slave_poll` variable. If the slave receives updates, kpropd updates its log file with any updates from the master. `kproplog` can be used to view a summary of the update entry log on the slave KDC. If incremental propagation is enabled, the principal `kiprop/slavehostname@REALM` (where `slavehostname` is the name of the slave KDC host, and `REALM` is the name of the Kerberos realm) must be present in the slave’s keytab file.

`kproplog` can be used to force full replication when iprop is enabled.

### 13.7.3 OPTIONS

- `-r realm` Specifies the realm of the master server.
- `-f file` Specifies the filename where the dumped principal database file is to be stored; by default the dumped database file is `/etc/krb5kdc/from_master`.
- `-p` Allows the user to specify the pathname to the `kdb5_util` program; by default the pathname used is `/usr/sbin/kdb5_util`.
- `-S [DEPRECATED]` Enable standalone mode. Normally kpropd is invoked by inetd(8) so it expects a network connection to be passed to it from inetd(8). If the `-S` option is specified, or if standard input is not a socket, kpropd will put itself into the background, and wait for connections on port 754 (or the port specified with the `-P` option if given).
- `-d` Turn on debug mode. In this mode, if the `-S` option is selected, kpropd will not detach itself from the current job and run in the background. Instead, it will run in the foreground and print out debugging messages during the database propagation.
- `-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 `/etc/krb5kdc/kpropd.acl`.

### 13.7.4 ENVIRONMENT

kpropd uses the following environment variables:

- `KR5_CONFIG`
- `KR5_KDC_PROFILE`

### 13.7.5 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`.

### 13.7.6 SEE ALSO

`kprop, kdb5_util, krb5kdc, inetd(8)`
13.8 kproplog

13.8.1 SYNOPSIS

kproplog [-h] [-e num] [-v] kproplog [-R]

13.8.2 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 master KDC server and the kpropd process on the slave KDC servers. When updates occur, they are logged to this file. Subsequently any KDC slave configured for incremental updates will request the current data from the master 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 master, kproplog also displays all of the update entries. If invoked on a slave 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.

13.8.3 OPTIONS

- R  Reset the update log. This forces full resynchronization. If used on a slave then that slave will request a full resync. If used on the master then all slaves 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
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
```

13.8.4 ENVIRONMENT

kproplog uses the following environment variables:

- KRB5_KDC_PROFILE
13.8.5 SEE ALSO

kpropd

13.9 ktutil

13.9.1 SYNOPSIS

ktutil

13.9.2 DESCRIPTION

The ktutil command invokes a command interface from which an administrator can read, write, or edit entries in a keytab or Kerberos V4 srvtab file.

13.9.3 COMMANDS

list

    list
Displays the current keylist.
Alias: l

read_kt

    read_kt keytab
Read the Kerberos V5 keytab file keytab into the current keylist.
Alias: rkt

read_st

    read_st srvtab
Read the Kerberos V4 srvtab file srvtab into the current keylist.
Alias: rst

write_kt

    write_kt keytab
Write the current keylist into the Kerberos V5 keytab file keytab.
Alias: wkt
**write_st**

```bash
write_st srvtab
```

Write the current keylist into the Kerberos V4 srvtab file `srvtab`.

Alias: `wst`

**clear_list**

```bash
clear_list
```

Clear the current keylist.

Alias: `clear`

**delete_entry**

```bash
delete_entry slot
```

Delete the entry in slot number `slot` from the current keylist.

Alias: `delent`

**add_entry**

```bash
add_entry [-key|-password] -p principal -k kvno -e enctype
```

Add `principal` to keylist using key or password.

Alias: `addent`

**list_requests**

```bash
list_requests
```

Displays a listing of available commands.

Aliases: `lr`, `?`

**quit**

```bash
quit
```

Quits ktutil.

Aliases: `exit`, `q`

### 13.9.4 EXAMPLE

```shell
ktutil: add_entry -password -p alice@BLEEP.COM -k 1 -e aes128-cts-hmac-shal-96
Password for alice@BLEEP.COM:
ktutil: add_entry -password -p alice@BLEEP.COM -k 1 -e aes256-cts-hmac-shal-96
Password for alice@BLEEP.COM:
```
13.9.5 SEE ALSO

kadmind, kdb5_util

13.10 k5srvutil

13.10.1 SYNOPSIS

k5srvutil operation [-i] [-f filename]

13.10.2 DESCRIPTION

k5srvutil allows an administrator to list or change keys currently in a keytab or to add new keys to the 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. Old keys are retained in the keytab so that existing tickets continue to work. 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.

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.

13.10.3 SEE ALSO

kadmind, ktutil

13.11 sserv

13.11.1 SYNOPSIS

sserv [ -p port ] [ -S keytab ] [ server_port ]
13.11.2 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 FILE:/etc/krb5.keytab.

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
```

13.11.3 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.

13.11.4 SEE ALSO

sclient(1), services(5), inetd(8)
# MIT KERBEROS DEFAULTS

## 14.1 General defaults

<table>
<thead>
<tr>
<th>Description</th>
<th>Default</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>keytab_definition file</td>
<td>FILE:/etc/krb5.keytab</td>
<td>KRB5_KTNAME</td>
</tr>
<tr>
<td>Client keytab_definition file</td>
<td>FILE:/etc/krb5/user/%{euid}/client.keytab</td>
<td>KRB5_CLIENT_KTNAME</td>
</tr>
<tr>
<td>Kerberos config file krb5.conf</td>
<td>/etc/krb5.conf:/usr/etc/krb5.conf</td>
<td>KRB5_CONFIG</td>
</tr>
<tr>
<td>KDC config file kdc.conf</td>
<td>/etc/krb5kdc/kdc.conf</td>
<td>KRB5_KDC_PROFILE</td>
</tr>
<tr>
<td>KDC database path (DB2)</td>
<td>/etc/krb5kdc/principal</td>
<td></td>
</tr>
<tr>
<td>Master key stash_definition</td>
<td>/etc/krb5kdc/.k5.realm</td>
<td></td>
</tr>
<tr>
<td>Admin server ACL file kadm5.acl</td>
<td>/etc/krb5kdc/kadm5.acl</td>
<td></td>
</tr>
<tr>
<td>Plugin base directory</td>
<td>/usr/lib/x86_64-linux-gnu/krb5/plugins</td>
<td></td>
</tr>
<tr>
<td>rcache_definition directory</td>
<td>/var/tmp</td>
<td>KRB5RCACHEDIR</td>
</tr>
<tr>
<td>Master key default enctype</td>
<td>aes256-cts-hmac-sha1-96</td>
<td></td>
</tr>
<tr>
<td>KDC default port</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>Second KDC default port</td>
<td>750</td>
<td></td>
</tr>
<tr>
<td>Admin server port</td>
<td>749</td>
<td></td>
</tr>
<tr>
<td>Password change port</td>
<td>464</td>
<td></td>
</tr>
</tbody>
</table>
14.2 Slave KDC propagation defaults

This table shows defaults used by the `kprop` and `kpropd` programs.

<table>
<thead>
<tr>
<th>Description</th>
<th>Default</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>kprop database dump file</td>
<td>/etc/krb5kdc/slave_datatrans</td>
<td></td>
</tr>
<tr>
<td>kpropd temporary dump file</td>
<td>/etc/krb5kdc/from_master</td>
<td></td>
</tr>
<tr>
<td>kdb5_util location</td>
<td>/usr/sbin/kdb5_util</td>
<td></td>
</tr>
<tr>
<td>kprop location</td>
<td>/usr/sbin/kprop</td>
<td></td>
</tr>
<tr>
<td>kpropd ACL file</td>
<td>/etc/krb5kdc/kpropd.acl</td>
<td></td>
</tr>
<tr>
<td>kprop port</td>
<td>754</td>
<td>KPROP_PORT</td>
</tr>
</tbody>
</table>

14.3 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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbolic name</th>
<th>Custom build path</th>
<th>Typical OS path</th>
</tr>
</thead>
<tbody>
<tr>
<td>User programs</td>
<td>BINDIR</td>
<td>/usr/local/bin</td>
<td>/usr/bin</td>
</tr>
<tr>
<td>Libraries and plugins</td>
<td>LIBDIR</td>
<td>/usr/local/lib</td>
<td>/usr/lib</td>
</tr>
<tr>
<td>Parent of KDC state dir</td>
<td>LOCALSTATEDIR</td>
<td>/usr/local/var</td>
<td>/var</td>
</tr>
<tr>
<td>Administrative programs</td>
<td>SBINDIR</td>
<td>/usr/local/sbin</td>
<td>/usr/sbin</td>
</tr>
<tr>
<td>Alternate krb5.conf dir</td>
<td>SYSCONFDIR</td>
<td>/usr/local/etc</td>
<td>/etc</td>
</tr>
<tr>
<td>Default ccache name</td>
<td>DEFCCNAME</td>
<td>FILE:/tmp/krb5cc_$(uid)</td>
<td>FILE:/tmp/krb5cc_$(uid)</td>
</tr>
<tr>
<td>Default keytab name</td>
<td>DEFKTNAME</td>
<td>FILE:/etc/krb5.keytab</td>
<td>FILE:/etc/krb5.keytab</td>
</tr>
</tbody>
</table>

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`.
The following environment variables can be used during runtime:

**KRB5_CONFIG** Main Kerberos configuration file. (See *MIT Kerberos defaults* for the default name.)

**KRB5_KDC_PROFILE** KDC configuration file. (See *MIT Kerberos defaults* for the default name.)

**KRB5_KTNAME** Default keytab file name. (See *MIT Kerberos defaults* for the default name.)

**KRB5_CLIENT_KTNAME** Default client keytab file name. (See *MIT Kerberos defaults* for the default name.)

**KRB5_CCNAME** Default name for the credentials cache file, in the form *type:residual*. The type of the default cache may determine the availability of a cache collection. For instance, a default cache of type *DIR* causes caches within the directory to be present in the global cache collection.

**KRB5RCACHETYPE** Default replay cache type. Defaults to *df1*. A value of *none* disables the replay cache.

**KRB5RCACHEDIR** Default replay cache directory. (See *MIT Kerberos defaults* for the default location.)

**KPROP_PORT** *kprop* port to use. Defaults to 754.

**KRB5_TRACE** Filename for trace-loggin output (introduced in release 1.9). For example, `env KRB5_TRACE=/dev/stdout kinit` would send tracing information for `kinit` to `/dev/stdout`. Some programs may ignore this variable (particularly setuid or login system programs).
16.1 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
```

16.2 List

**Error:** KDC has no support for encryption type while getting initial credentials

**Error:** 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.

Seen in: clients

**Error:** kprop: No route to host while connecting to server
Make sure that the hostname of the slave (as given to kprop) is correct, and that any firewalls between the master and the slave allow a connection on port 754.

**Error:** kprop: Connection refused in call to connect while opening connection

If the slave 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 slave is intended to run kpropd in standalone mode, make sure that it is running.

**Error:** kprop: Server rejected authentication while authenticating to server

Make sure that:

1. The time is synchronized between the master and slave KDCs.
2. The master stash file was copied from the master to the expected location on the slave.
3. The slave has a keytab file in the default location containing a host principal for the slave’s hostname.
17.1 LDAP backend on Ubuntu 10.4 (lucid)

Setting up Kerberos v1.9 with LDAP backend on Ubuntu 10.4 (Lucid Lynx)

17.1.1 Prerequisites

Install the following packages: slapd, ldap-utils and libldap2-dev

You can install the necessary packages with these commands:

```
sudo apt-get install slapd
sudo apt-get install ldap-utils
sudo apt-get install libldap2-dev
```

Extend the user schema using schemas from standard OpenLDAP distribution: cosine, mics, nis, inetcomperson

```
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/cosine.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/mics.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/nis.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/inetcomperson.ldif
```

17.1.2 Building Kerberos from source

```
./configure --with-ldap
make
sudo make install
```

17.1.3 Setting up Kerberos

Configuration

Update kdc.conf with the LDAP back-end information:

```
[realms]
  EXAMPLE.COM = {
    database_module = LDAP
  }
```
[dbmodules]
LDAP = {
    db_library = kldap
    ldap_kerberos_container_dn = cn=krbContainer,dc=example,dc=com
    ldap_kdc_dn = cn=admin,dc=example,dc=com
    ldap_kadmind_dn = cn=admin,dc=example,dc=com
    ldap_service_password_file = /usr/local/var/krb5kdc/admin.stash
    ldap_servers = ldapi:///
}

**Schema**

From the source tree copy src/plugins/kdb/ldap/libkdb_ldap/kerberos.schema into /etc/ldap/schema

Warning: this step should be done after slapd is installed to avoid problems with slapd installation.

To convert kerberos.schema to run-time configuration (cn=config) do the following:

1. Create a temporary file /tmp/schema_convert.conf with the following content:

   ```
   include /etc/ldap/schema/kerberos.schema
   ```

2. Create a temporary directory /tmp/krb5_ldif.

3. Run:

   ```
   slaptest -f /tmp/schema_convert.conf -F /tmp/krb5_ldif
   ```

   This should in a new file named /tmp/krb5_ldif/cn=config/cn=schema/cn={0}kerberos.ldif.

4. Edit /tmp/krb5_ldif/cn=config/cn=schema/cn={0}kerberos.ldif by replacing the lines:

   ```
   dn: cn={0}kerberos
   cn: {0}kerberos
   ```

   with

   ```
   dn: cn=kerberos,cn=schema,cn=config cn: kerberos
   ```

   Also, remove following attribute-value pairs:

   ```
   structuralObjectClass: olcSchemaConfig
   entryUUID: ...
   creatorsName: cn=config
   createTimestamp: ...
   entryCSN: ...
   modifiersName: cn=config
   modifyTimestamp: ...
   ```

5. Load the new schema with ldapadd (with the proper authentication):

   ```
   ldapadd -Y EXTERNAL -H ldapi:/// -f /tmp/krb5_ldif/cn=config/cn=schema/cn={0}kerberos.ldif
   ```

   which should result the message adding new entry "cn=kerberos,cn=schema,cn=config".

**17.1.4 Create Kerberos database**

Using LDAP administrator credentials, create Kerberos database and master key stash:
kdb5_ldap_util -D cn=admin,dc=example,dc=com -H ldapi:/// create -s

Stash the LDAP administrative passwords:
kdb5_ldap_util -D cn=admin,dc=example,dc=com -H ldapi:/// stashsrvpw cn=admin,dc=example,dc=com

Start krb5kdc:

krb5kdc

To destroy database run:

kdb5_ldap_util -D cn=admin,dc=example,dc=com -H ldapi:/// destroy -f

17.1.5 Useful references

- Kerberos and LDAP

17.2 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).

17.2.1 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.

17.2.2 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.

17.2.3 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
}
```

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, Version 4
[...]
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc kadmin/admin'
[...]
Number of keys: 1
Key: vno 15, des-cbc-crc, no salt
[...]
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q 'getprinc kadmin/changepw'
[...]
Number of keys: 1
Key: vno 14, des-cbc-crc, no salt
[...]
```

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).
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-shal-96, no salt
Key: vno 2, aes128-cts-hmac-shal-96, no salt
Key: vno 2, des3-cbc-shal, no salt
Key: vno 2, des-cbc-crc, no salt
Key: vno 1, des-cbc-crc, Version 4
[...]
```

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-shal-96: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.

```
[root@casio krb5kdc]# kadmin.local -r ZONE.MIT.EDU -q "cpw -e ${enctypes} -randkey 
> kadmin/admin"
```

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.

17.2. Retiring DES

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,
[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"
> -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-shal-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-shal-96 added to keytab WRFILE:/etc/zephyr/krb5.keytab.
Entry for principal zephyr/zephyr@ZONE.MIT.EDU with kvno 4, encryption type des3-cbc-shal 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 = {
des3-cbc-shal:normal des3-hmac-shal: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 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
3 12/12/12 15:31:19 host/dr-willy.xvm.mit.edu@ZONE.MIT.EDU
```

Kerberos Administration Guide, Release 1.11.1

Chapter 17. Advanced topics
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.

```bash
[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.

```bash
[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.

```bash
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.
```

### 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 master, to avoid unpleasant surprises should the master fail and a slave need to be promoted.

It is now appropriate to remove the legacy single-DES key from the krbtgt/REALM entry:

```bash
[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.

```bash
[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.

```bash
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.
```
Once all principals have been re-keyed, DES support can be disabled on the KDC, and client machines can remove `allow_weak_crypt = true` from their `krb5.conf` configuration files, completing the migration.

**Support for legacy services**

If there remain legacy services which do not support non-DES enctypes (such as AFS), `allow_weak_crypt` 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_crypt` 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.

**17.2.4 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.

It 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.
18.1 Whitepapers


18.2 Tutorials

1. Fulvio Ricciardi <http://www.kerberos.org/software/tutorial.html>

18.3 Troubleshooting

7. https://bugs.launchpad.net/ubuntu/+source/libpam-heimdal/+bug/86528
INDEX

R
RFC
RFC 2253, 20
RFC 2782, 37
RFC 4556, 21, 29
RFC 6649, 125