PacketCable™/CableHome™ KDC Lab User Guide

(Last update: 24 February, 2010)

IPfonix, Inc.
7912 Fairview Road
Boulder
CO 80303
USA

Tel: +1 303 494 0394
Fax: +1 781 240 0527

info@ipfonix.com

http://www.ipfonix.com

Table of Contents

1 Intended Audience

This document is designed to help anyone wishing to configure and run the IPfonix PacketCable/CableHome KDC.

2 PacketCable/CableHome KDC Versions

The PacketCable/CableHome KDC is available in versions for Linux, Windows and Solaris. Except for minor differences, the versions are functionally identical[1]. The Windows version is intended for use only under controlled lab conditions, since we do not recommend (and do not support) the Windows platform for use in real deployment networks with paying subscribers.

The IPfonix, Inc. KDC software is identical regardless of its environment (PacketCable, CableHome or both) environment. Which functions and features are available is controlled by the license file provided to you by IPfonix, Inc.

3 Hardware Requirements

The Linux and Windows version of the KDC operate on ordinary PC-class hardware (although for reliability reasons, inexpensive PC-type machines are not recommended except in a lab environment). A reasonable minimal PC configuration is:

- 128 MB Main Memory
- 4 GB Hard drive
- Network Interface Card
- 700 MHz processor

Depending on the use that is made of the KDC, systems that fall substantially short of these requirements may also offer satisfactory performance, especially in a lab environment.

The Solaris version of the KDC operates on Sun SPARC hardware.

4 Software Requirements

The KDC has been tested under common versions of Windows. The Linux version is currently built and undergoes extensive testing on the 32-bit version of Kubuntu 6.06. It also undergoes basic testing on several other common distributions. The codebase is designed with portability in mind, and we believe that the KDC should operate under any reasonably current version of Linux. The Solaris version is built and tested under Solaris 8, and should work correctly on that or any later version of Solaris.

[1]The most important difference among the versions is that IPv6 is supported only under Linux and Windows.

4.1 Linux Software Requirements

The Linux version requires access to the following dynamic (shared) libraries:

- `linux-gate.so.1`
- `libc.so.6`
- `/lib/ld-linux.so.2`
- `libgcc_s.so.1`
- `libpthread.so.0`
- `libstdc++.so.6`
- `libm.so.6`

All these libraries should be supplied by your distribution (and are usually installed by default).

4.2 Solaris Software Requirements

The Solaris version of the KDC requires the presence of several additional libraries or facilities that may or may not be already present on your system:

- `/dev/urandom`
- `libgcc_s.so.1`
- `libstdc++.so.5`

The last two libraries should already be installed if you have a working version of gcc on the system.

5 Overview

The IPfonix PacketCable/CableHome KDC has been designed to implement all the relevant requirements contained in the PacketCable Security Specification and the CableHome Specification (including ECNs applied to the latest published versions). Henceforth, we will simply refer to these as “the specification” or “the security specification”.

As of this writing, there are no known cases in which the KDC fails to meet the requirements contained in the specification.

This document describes how to configure and operate the Linux and Windows versions of the KDC. Installation and configuration under Solaris is identical to configuration under Linux, except that the `<INI>` directory (defined below) is `<KDC>/solaris` rather than `<KDC>/linux`. Reasonable familiarity with the particular operating system on which the KDC is running is assumed. In this document, examples will assume that the software is running under Linux; in cases where a non-obvious change is necessary in order for the software to run correctly under Windows, this will be clearly noted.

6 Installing the Software

The KDC executable is a file called `kdc` (Linux or Solaris) or `KDC.EXE` (Windows). This executable can be placed in any directory. For simplicity, we will refer to this directory as `<KDC>`; typically this is actually a directory such as `/usr/local/kdc/` in Linux or Solaris, or `C:\KDC` in Windows. Because the KDC requires access to system ports, it is important that, under Linux or Solaris, the KDC executable be installed as setuid root. (Usually, the simplest thing to do is merely to install and run the KDC from a root account.)

6.1 Running multiple KDCs on one machine

It is possible to run multiple copies of the KDC simultaneously on a single machine. In order to do this, each KDC should be run out of its own directory hierarchy (*i.e.*, `<KDC>` for the KDCs should be different directories), and the values in the respective `kdc.ini` files must not conflict with one another. In particular, the interface addresses for the KDCs must be unique.

6.2 Directory hierarchy

As described in this section, the KDC requires that a particular structure of subdirectories be present, and that certain files exist in those subdirectories. If any required file is absent, the KDC will abort on start-up and provide a message on the console indicating the nature of the problem.

Under Windows, the directory `<KDC>\windows` must be present; the corresponding directory in Linux is `<KDC>/linux` and in Solaris `<KDC>/solaris`. We refer to this directory as `<INI>`, because this is where the configuration file, `kdc.ini`, is located.

In addition, the directories `<INI>/keys`, `<INI>/packetcable/certificates*` and `<INI>/cablehome/certificates*` must exist and, as explained below, must be correctly populated.

6.2.1 Files in <KDC>

Only two files are required in `<KDC>`: the executable and the license file. The license file is `kdc.license` and should have been included with your software package. The license file is plain ASCII, and, at a minimum, indicates the entity to which the license has been issued, as well as the expiration date. This information is also written to the console when the KDC starts to execute.

6.2.2 Files in <INI>

Only one file is required in `<INI>`: the file `kdc.ini`, which is the KDC configuration file. This file is read at KDC start-up, and defines the running “context” for the KDC. See below for details of the contents of the `kdc.ini` file.

If no kdc.ini file is present, the KDC will ask the user for the IP address that the KDC is to use, and will then try to continue to start, using default values for all its configuration parameters.

7 Files in <INI>/keys

The directory <INI>/keys contains the service keys known to the KDC. In order to make testing easier, these keys are unencrypted[2]. In PacketCable and CableHome, all service keys are currently 24-octet 3DES keys. The format for a key file[3] is:

- Two octets: key version number as a 16-bit integer, in network order
- 24 octets: the actual 3DES key

Thus, a key file is 26 octets in length. It is important to recognise that this file is raw *binary*; it does *not* contain ASCII-encoded hexadecimal characters. When used for testing, the value of the first two octets is generally unimportant, since neither PacketCable nor CableHome currently have a defined mechanism for key versioning.

The name of key files reflects the full Kerberos name of the service to which the key applies. The normal format for a Kerberos principal identifier is:

component/component/.../component@REALM

However, the forward slash character (/) is not a valid character in a filename in Linux, Solaris or Windows. Therefore, the KDC replaces the forward slash with the comma character (,) when encoding the name of the Kerberos service name to create a file name.

So, for example, suppose that we wish to provision the KDC with a service key for a CMS whose FQDN is cms1.ratco.com, in the realm RATCO.COM. Then, in the directory <INI>/keys we would create a file called cms,cms1.ratco.com@RATCO.COM. This file would contain exactly 26 octets. The first two octets are the version number (typically 0x0000) and the remaining 24 octets are the service key that is shared with the CMS.

If a service key file is present but its length is incorrect, the KDC will create a new service key for the named principal identifier, and write that key to the file at start-up time.

Note that the MTA FQDN messaging has a service name of mtafqdnmap. Therefore, if the KDC is expected to support MTA FQDN messaging to a provisioning server, then there must be a service key present for the service mtafqdnmap,<provserver FQDN>@<REALM>.

The service key that the KDC will use for PKCROSS tickets should be placed in a file whose name is of the form pkcross@<REALM>. For example, a KDC in the realm IPFONIX.COM that is intended to issue PKCROSS tickets should have a key in the file pkcross@IPFONIX.COM.

[2]They should therefore be made readable only by the root user.

[3] Service keys generated by the dynamic key mechanism described in section Error: Reference source not found are stored in a more complicated format. These cannot easily be edited directly. They can, however, be replaced by keys in the 26-octet format described in this section, should the need arise.

8 Files in <INI>/packetcable/certificates* and <INI>/cablehome/certificates*

These directories are used to store several X.509 certificates in *standard DER-encoded binary format*[4], along with the KDC private key. The KDC will separately examine every directory of the form `<INI>/packetcable/certificates*` and `<INI>/packetcable/certificates*`, and check the contents for validity. As might be expected, certificates related to PacketCable (or Euro-PacketCable or IPCablecom) functionality reside in the `<INI>/packetcable/certificates*` directories; certificates for CableHome functions reside in the `<INI>/cablehome/certificates*` directories.

For example, a user who wishes the KDC to interact with clients using PacketCable certificates and also with those using Euro-PacketCable certificates, may have two directories on the KDC, possibly called `<INI>/packetcable/certificates-packetcable` and `<INI>/packetcable/certificates-europacketcable`.

Inside each directory, certificates may have any name, but each must end with the file extension `.cer`. The directory must contain certificates corresponding to the complete Service Provider hierarchy, and also the root of the client hierarchy. At start-up, the KDC will check that the certificates in the service provider hierarchy chain correctly; it will also subject the certificates to a number of tests to ensure that they meet the applicable CableHome, PacketCable, Euro-PacketCable or IPCablecom requirements.

Each certificate directory is treated independently of the others, except that the public keys of the MTA Root certificates must all be unique. This is to ensure that, when the KDC receives a PKINIT request from a client, it can correctly determine which set of certificates is to be used to generate the reply.

In addition to the certificates themselves, each `certificates*` directory must contain the RSA private key that corresponds to the KDC certificate in the directory (this is necessary in order that the KDC can correctly sign AS-REP messages).The KDC expects that the key will be in a file named `KDC_private_key`.

There are several formats commonly used to transfer RSA keys among systems. The KDC allows you to use PKCS#8, PKCS#1 or a proprietary format to load the key into the KDC.

The KDC will attempt to read the file `KDC_private_key` in each of the different possible formats (in order: PKCS#8, PKCS#1, proprietary).

Note that if you use one of the PKCS formats, the file must contain the binary information exactly as defined by the relevant PKCS specification[5].

As an alternative to the PKCS formats, you can use a simple proprietary ASCII hex format that allows operators to create a more friendly file that can more easily be parsed

[4]The KDC will also recognize most PEM-formatted certificates, but since this format is not standardized and differs slightly according to the application that created the certificate, the KDC might fail to recognize such a certificate.

[5]As for certificates, the KDC can read most PEM-encoded PKCS#1 and PKCS#8 keys.

by a human in the case that some operation related to decryption or signature creation appears to be behaving incorrectly.

In this format, each element of the private key is stored on a separate line, and each element is represented by a series of hexadecimal characters. The elements, in order, are:

modulus

public exponent

exponent

prime_0

prime_1

prime_exponent_0

prime_exponent_1

coefficient

So, for example, a `KDC_private_key` file in proprietary format might look like this:

```
a6 b0 5a c0 2a 6e 8d 36 3b 40 cf 86 c5 09 ef  ...  ee c7 53
01 00 01
6e 54 de cc c1 89 49 58 f3 21 73 fb bd a2 54  ...  e2 31 11
dd 77 92 67 58 20 df 24 da db 9f 1c 59 1e d9  ...  57 4a ab
c0 ae 2b 8d 19 05 00 2e 21 f1 1e 89 87 c2 88  ...  b5 75 f9
0b c0 ca be 3d 49 11 4e 8d 66 d6 5c d4 c5 f4  ...  53 61 9d
07 38 7f b9 51 ee b6 0b 04 8a 9c b2 5a bc 17  ...  a2 c3 41
75 0e f5 16 9f 6e 33 1a 44 f0 1f 8d 5a 22 18  ...  c8 c2 23
```

where the public exponent is the value 0x010001 and the value of the second prime is 0xc0ae2b...75f9.

8.1 Certificate Use

In the AS-REQ message, the client sends to the KDC a chained pair of certificates. When it receives an AS-REQ, the KDC looks at the sets of certificates that were read at start-up in order to determine which client root certificate was used to sign the received MTA Manufacturer (for PacketCable) or Manufacturer (for CableHome) certificate. For the remainder of the processing related to the received message, it uses only the certificates (and private key) that were in the same directory as the matching client root certificate.

The KDC package includes a set of compliant certificates that may be used in a testing environment. Naturally, if the user prefers to use his own certificates, he may do so simply by ensuring that they are distributed correctly to the client(s) and the KDC.

Note that the MTA Root certificate and the CableLabs Manufacturer Root certificate distributed in the package are *not* the same as the official MTA Root and CableLabs Manufacturer Root certificates. This allows a user to test without the need for the delay and expense concomitant with obtaining an "official" MTA Manufacturer or Manufacturer certificate. If the user already possesses an MTA Manufacturer or

Manufacturer certificate signed by the appropriate official VeriSign root certificate, then the user can use this certificate simply by replacing the `MTA_Root.cer` or `CableLabs_Manufacturer_Root.cer` file with the root certificate distributed by VeriSign. The official root certificates are not included in this package in order to avoid any issues related to possible copyright infringement.

The private keys corresponding to the unofficial MTA Root and CableLabs Manufacturer Root certificate is also distributed with the KDC software. This allows MTA vendors to generate correctly-chained MTA Manufacturer, Manufacturer, MTA Device and PS Element certificates.

Similarly, the Service Provider Certificate hierarchy distributed with the KDC uses certificates generated independently by IPfonix, Inc. These certificates conform to the specification, but are not rooted in the official root key pair (which is held by Verisign; only Verisign has the private key from this key pair, hence only they can create certificates rooted in the official key pair).

In normal operation, the KDC performs compliance checks on certificates at start-up and when an AS-REQ message is received, to ensure that the certificates meet the requirements in the specification.

9 The `<INI>/allow` and `<INI>/deny` directories

If the directories `<INI>/allow` or `<INI>/deny` exist, the KDC will examine each of these directories for valid binary X.509 certificates contained in files with the `.cer` extension. Certificates in the `<INI>/allow` directory are added to an internal "allow" list maintained by the KDC; similarly, certificates in the `<INI>/deny` directory are added to an internal "deny" list.

When the KDC receives a PKINIT request, it examines all the certificates in the request, and compares them to the "allow" and "deny" lists. The subsequent processing of the request depends on whether the certificate was found on one of these lists:

- If the "allow" and "deny" lists are both empty, the KDC will process the request as normal;
- If the "deny" list is empty, but the "allow" list is not empty, the KDC will process the request only if it contains one or more certificates that are present in the "allow" list;
- If the "allow" list is empty, but the "deny" list is not empty, the KDC will process the request unless one or more of the certificates in the request is present in the "deny" list;
- If both the "allow" list and the "deny" list are populated, the KDC will process the request only if one or more certificates in the request are present in the "allow" list and none of the certificates is present in the "deny" list.

The "allow" and "deny" lists may be used, for example, to ensure that only clients from a particular set of vendors are provided with tickets.

10 Time

It is important that the KDC have a correct notion of UTC. AS-REQ messages coming from a client contain the current UTC time, and if the KDC has a different notion of time, then it will return a clock skew error, as required by Kerberos and the security specification. Usually, the easiest way to ensure that the KDC can correctly derive UTC is simply to keep the system clock on UTC. Whatever method is chosen, though, the system must return the correct UTC when the gmtime() call is made; the KDC uses this call to check the clock skew between itself and clients.

Depending on the stability of the hardware clock, it might be useful to run NTP on the KDC. Usually this is not necessary in a lab environment, but if the hardware clock gains or loses more than a few seconds per day, running NTP will ensure that the KDC remains correctly synchronised with UTC.

11 Log file

The KDC maintains a log file in real time. The log file is located in the directory `<KDC>/log`. The name of the logfile is in the form `IPfonixyyyymmddhhmm.log` and it contains the time at which the KDC was started. By default, the KDC automatically closes the log file and opens a new one at the start of a new UTC day. A continuously-running KDC maintains a week's worth of logs. If the KDC is left running for more than a week, it will delete files more than a week old at the start of each day. This culling affects only files that were created since the last time that the KDC was started.

The KDC permits an alternative method of managing log files. Instead of creating a new file per day, entries in the `<INI>/kdc.ini` file allow the user to cause a new logfile to be created when the current file exceeds a certain size. When this method is chosen, the KDC also allows the user to decide how many log files to keep. In lab situations, there is probably nothing to be gained by changing the default behaviour.

The log file contains real-time status information (most of this is also written to the standard output device). It also includes binary copies of all messaging into and out of the KDC (this latter information is not written to the standard output). All entries are time-stamped, contain the name or number of the thread responsible for the output, and provide an indication of which threads are currently busy. In the case of KDC malfunction, you are strongly encouraged to mail a copy of the current log file to IPfonix, Inc. to help us to debug the problem. In addition to the log file, it helps us if you send a copy of the `<INI>/kdc.ini` file.

When copying or e-mailing the log file, please remember that *it is a binary file* and take any appropriate steps to ensure that it is not mutilated during transport.

In the event of a KDC crash, we strongly request that you provide us with all the details possible, including copies of the `<INI>/kdc.ini` and log files. We have heavily stress-tested the KDC code (under Linux) and believe that no sequence of incoming packets should be able to crash the KDC.

12 PacketCable 2.0

The KDC can be easily configured to support PacketCable 2.0. In the [general] section of the `kdc.ini` file, place the following line:

```
pc20 = true
```

This will enable IPv6 support and support for the PacketCable 2.0 EUE interface to the Provisioning Server.

13 Command Line Options

The following options may be passed to the KDC on start-up:

13.1 -append

Appends the standard (console) output instead of overwriting the destination file. Works only if the `-cout` option is also used.

13.2 -cout <filename>

Sets the standard (console) output to <filename>.

13.3 -cwd <directory-name>

Sets the working directory to <directory-name>.

13.4 -D

Runs the KDC as a daemon (Linux only).

13.5 -named-pipe <named-pipe-name>

Sends log output to a named pipe instead of a file. This option works only in Linux.

13.6 -verbose-cert-check

Runs a verbose check of the certificates available to the KDC. You should start he KDC with this option if you are having difficulty configuring the KDC with certificates correctly. When used with this option, the KDC will perform a verbose check of the certificates and then exit.

13.7 --version

Causes the KDC to print information relating to the particular build and then to exit.

14 Operating the PacketCable pseudo-provserver

The IPfonix, Inc. KDC includes code that permits it to respond to PacketCable clients attempting to provision themselves using the flows described in the PacketCable provisioning specification. We call this code the "pseudo-provserver" or "pseudo-ps".

This does *not* mean that the KDC acts as a fully fledged PacketCable Provisioning Server (which typically also performs MTA management functions, although that is not required by the PacketCable specifications). It does mean that an MTA can exchange provisioning messages with the KDC as if it were a Provisioning Server. This allows an MTA vendor to test and debug the client code without the need for a real Provisioning Server in the lab. Since the KDC can be configured to leak SNMPv3 keys[6], it also allows MSOs with in-place backend provisioning systems to create systems in which the KDC acts as a front-end to the pre-existing provisioning system, without the additional expense of adding a PacketCable-compliant provisioning server.

The messages supported by the pseudo-ps are (referencing Figure 5 in the PacketCable provisioning specification):

- MTA-13: processing incoming the AP-REQ;
- MTA-14: creating and returning the AP-REP;
- MTA-15: processing incoming the INFORM (including sending a RESPONSE that is not included in Figure 5);
- MTA-19: creating and sending the SET;
- MTA-25: processing incoming the INFORM (including sending a RESPONSE that is not included in Figure 5).

The operation of the pseudo-ps is controlled entirely by entries in the `kdc.ini` file. For details of this operation, see section 19.6.

14.1 Leaking SNMPv3 keys

In the Linux version of the KDC, entries in the `kdc.ini` file allow the user to leak SNMPv3 keys to a file or to a named pipe (see section 19.7.5).

Each record in the file or named pipe is delimited with a single LF character (the Linux End-Of-Line marker).

For each leaked key, the format defined in Table 1 applies (the separator between fields is a space).

Table 1

Field	Format
KEYSTART	
USM User Name	hexified
Auth key	hexified
Priv key	hexified

[6] Linux version only

Client IP address	dotted decimal
Client SNMP Engine ID	hexified
Client engine boots	number
Client engine time	number
Server SNMP Engine ID	hexified
Server engine boots	number
Server engine time	number
KEYEND	

The format "hexified" means the following:

A number <n>, followed by a space, followed by <n> hexadecimal characters.

So, for example, the hexified form of the string `test` would be: `4 74 65 73 74.`

For example, suppose that the complete record for a leaked key looks like this (all on one line): `KEYSTART 26 4d 54 41 2d 50 72 6f 76 2d 30 30 3a 34 30 3a 37 42 3a 38 38 3a 41 34 3a 34 37 16 88 bb 84 11 f5 82 20 ab 1b 50 0e 03 3f 1d c6 7e 0 10.1.21.130 12 80 00 03 e3 03 00 40 7b 88 a4 45 77 10 25 04 74 65 73 74 123 456 KEYEND`

In this example, the USM User Name corresponds to

26 4d 54 41 2d 50 72 6f 76 2d 30 30 3a 34 30 3a 37 42 3a 38 38 3a 41 34 3a 34 37

which, when decoded, is:

MTA-Prov-00:40:7B:88:A4:47

The Auth key is:

16 88 bb 84 11 f5 82 20 ab 1b 50 0e 03 3f 1d c6 7e

So the actual SNMP authorization key is:

88 bb 84 11 f5 82 20 ab 1b 50 0e 03 3f 1d c6 7e

The Priv key is:

0

So there is no encryption key (meaning that the SNMPv3 messages will be unencrypted).

The IP address of the client is:

10.1.21.130

The SNMP Engine ID of the client is:

12 80 00 03 e3 03 00 40 7b 88 a4 45 77

which corresponds to an actual Engine ID of:

`80 00 03 e3 03 00 40 7b 88 a4 45 77`

The Engine Boots value of the client is:

10

The Engine Time value of the client is:

25

The SNMP Engine ID of the server is:

04 74 65 73 74

which corresponds to an actual Engine ID of:

```
74 65 73 74
```

The Engine Boots value of the server is:

123

The Engine Time value of the server is:

456

15 OCSP

The KDC contains an OCSP (Online Certificate Status Protocol) client that allows the user to check for revoked certificates in real time. Operation of the OCSP client is controlled by the entries in the `[ocsp]` section of the `kdc.ini` file. By default, the OCSP client is turned off, but it can be turned on by including the line:

```
ocsp = true
```

in the `[ocsp]` section.

When operating the OCSP client, it is important to define the correct behaviour for the KDC when it is unable to determine the status of a certificate. That is, if the status of a certificate cannot be ascertained, should the KDC accept or reject the certificate? By default, the KDC will accept such certificates; this behavior can be changed by including the line:

```
default = revoked
```

in the `[ocsp]` section.

The OCSP client formats requests as HTTP messages (see RFC 2560), and sends them over TCP to the OCSP responder.

16 Jabber

The KDC includes a minimal jabber client, which is controlled by entries in the `[jabber]` section of the `kdc.ini` file.

When configured to send jabber messages, the following messages are sent to the configured jabber client:

1. A message that states that the KDC has started;
2. A message whenever an AS-REP is sent to a client;
3. A message that states that the KDC has been stopped.

17 Administration

The administration API is available to OEM resellers under NDA. A simple example of an administration program is included with the Linux and Windows versions of the KDC. The documentation for this program is included in a separate Example Administration Guide.

18 Monitoring Statistics

The Linux and Solaris versions of the KDC support monitoring ongoing operational statistics through the use of a named pipe. This feature is controlled by providing a name for the named pipe in the `[statistics]` section of the `kdc.ini` file.

If a pipe is named in the `kdc.ini` file, then the pipe is created at startup and statistics are written to this pipe once per minute as long as the KDC is operational. The information written to the pipe can be parsed and presented by whatever mechanism the user prefers.

The output comprises a single line of text comprising several fields separated by the semi-colon (;) character. Each field is of the form:

```
statistic-name = statistic-value
```

The following table describes all of the statistics written to the pipe by the KDC.

Name of Statistic	Meaning
`curtime`	The current time (UTC)
`starttime`	Time at which the KDC started (UTC)
`uptime`	The amount of time, in days, that the KDC has been running. Given to two decimal places.
`rcvd-AS-REQ-total`	Total number of AS-REQ messages received
`rcvd-AS-REQ-rate`	Number of AS-REQ messages received in the last minute
`sent-AS-REP-total`	Total number of AS-REP messages transmitted
`sent-AS-REP-rate`	Number of AS-REP messages sent in the last minute
`sent-KRB-ERROR-total`	Total number of KRB-ERROR messages transmitted
`sent-KRB-ERROR-rate`	Number of KRB-ERROR messages transmitted in the last minute
`unexpired-tickets`	Number of tickets that have been issued and have not yet expired
`unexpired-tickets-change`	Change in the last minute in the number of tickets that have been issued and have not yet

	expired
pct-licensed-capacity	The current percentage of the licensed number of valid tickets, given to two decimal places
sent-MTA-FQDN-REQ-total	Total number of MTA FQDN requests that have been transmitted
sent-MTA-FQDN-REQ-rate	Number of MTA FQDN requests that were transmitted in the last minute
rcvd-MTA-FQDN-REP-total	Total number of MTA FQDN replies that have been received
rcvd-MTA-FQDN-REP-rate	Number of MTA FQDN replies that have been received in the last minute (not including errors)
rcvd-MTA-FQDN-KRB-ERROR-total	Total number of KRB-ERROR replies that have been received to MTA FQDN requests
rcvd-MTA-FQDN-KRB-ERROR-rate	Number of KRB-ERROR replies that were received to MTA FQDN requests in the last minute
rcvd-TGS-REQ-total	Total number of TGS-REQ messages that have been received
rcvd-TGS-REQ-rate	Number of TGS-REQ messages that were received in the last minute
rcvd-msgs-SNMP-port-total	Total number of messages received on the SNMP port *(only if pseudo provserver is enabled)*
rcvd-msgs-SNMP-port-rate	Number of minutes received on the SNMP port in the last minute *(only if pseudo provserver is enabled)*
rcvd-AP-REQ-total	Total number of AP-REQ messages received *(only if pseudo provserver is enabled)*
rcvd-AP-REQ-rate	Number of AP-REQ messages received in the last minute *(only if pseudo provserver is enabled)*
sent-AP-REP-total	Total number of transmitted AP-REP messages *(only if pseudo provserver is enabled)*
sent-AP-REP-rate	Number of AP-REP messages transmitted in the last minute *(only if pseudo provserver is enabled)*
sent-PS-KRB-ERROR-total	Total number of KRB-ERROR messages transmitted by the pseudo provserver *(only if pseudo provserver is enabled)*
sent-PS-KRB-ERROR-rate	Number of KRB-ERROR messages transmitted by the pseudo provserver in the last minute *(only*

	if pseudo provserver is enabled)

The values relating to the pseudo provserver are output only if the pseudo provserver functions are enabled by the `[pseudo ps]` section of the `kdc.ini` file.

19 INI configuration file

The file `<INI>/kdc.ini` contains configuration information read by the KDC at start-up time.

The format of `kdc.ini` closely follows the usual Windows format for `.ini` files. The file contains a number of named sections, and within each section is a series of configuration commands. Please note that all entries in the `kdc.ini` file are case sensitive. The file parser is rather unforgiving, so if the KDC appears to be misbehaving, the first thing to do is to look carefully at the exact contents of the `kdc.ini` file to make sure that there are no typographical errors.

Within the `kdc.ini` file, a section name is indicated by square brackets. Most commands go in the `[general]` section, so that the start of a valid `kdc.ini` file might look like this:

```
[general]
interface address = 127.0.0.1
realm = IPFONIX.COM
FQDN = kdc.ipfonix.com
```

Lines in the `kdc.ini` file are ignored if they begin with a hash sign (`#`). Blank lines are ignored, as are lines that contain commands that the parser does not understand.

The next section of this manual details the `[compliance]` section. Following this is a complete list of all the commands that may be placed in the `[admin]`, `[general]`, `[jabber]`, `[ocsp]`, `[pkcross]`, `[pseudo ps]`, `[statistics]`, `[testing]`, `[checks]`, `[violations]` and `[ipv6]` sections. Other sections and their contents are not currently documented. Please contact IPfonix, Inc. if you require further details.

19.1 [compliance] section

19.1.1 compliant =

By default, the KDC attempts to operate in a PacketCable-compliant manner. However, many of the commands in the `[general]` section attempt to customize the behaviour of the KDC so that it operates in ways that are not PacketCable-compliant, but which may be appropriate for a testing environment. *By default, the KDC will ignore such commands.* This is a safety measure, so that an operator cannot easily operate a deployed KDC in an insecure or non-compliant manner, even if the `[general]` section contains commands that would result in such operation.

The user should exercise caution when operating the KDC in non-compliant mode. Many of the commands that are accepted when operating in this mode seriously degrade the level of security offered by the KDC.

In order to force the KDC to process all the commands in the `[general]` section, even if they result in non-compliant behaviour, the `kdc.ini` file must contain a `[compliance]` section with a command that sets the value of `compliant` to `false`.

The default value is `true`.

This command is not required.

Example:

```
compliant = false
```

19.2 [admin] section

19.2.1 admin =

This parameter is used to globally control whether the KDC administration function is enabled.

The default value is `false`.

This command is not required.

Example:

```
admin = true
```

19.2.2 password =

This sets the password that the KDC will use to authenticate incoming administration requests.

The default value is the empty string.

This command is not required.

Example:

```
password = ipfonixpw
```

19.2.3 port =

This controls the TCP port on which the KDC will listen for administrative commands.

The default value is `411`.

This command is not required.

Example:

```
port = 9999
```

19.3 [general] section

19.3.1 allow duplicate cert keys =

By default, the KDC will check to make sure that the keys encapsulated by certificates do not match any keys presented in a different certificate. This closes a security hole that could occur if a vendor were to re-use keys in a misguided effort to cut costs. If the value of this parameter is `true`, then the KDC no longer makes this check and will allow multiple MTAs to register with the same keys.

The default value is `false`.

This command is not required.

Example:

```
allow duplicate cert keys = true
```

19.3.2 allowed packets per minute =

If the KDC is configured to drop datagrams from clients that appear to be mounting a denial-of-service attack (see the `drop dos datagrams` command), this entry in the configuration file determines the level at which the KDC will conclude that an attack is mounted.

The default value is 4 (*i.e.*, a client sending five datagrams in the course of a single minute will be considered to be mounting an attack.)

This command is not required.

Example:

```
allowed packets per minute = 10
```

19.3.3 allow kdc cert principal name mismatch =

The KDC certificate is normally required to contain a principal name that is consistent with the configured realm of the KDC. (The security specification requires an MTA to reject a response in which the name in the certificate does not match the local realm name.) However, sometimes it is useful in a lab environment to violate this requirement. This command, when `true`, turns off the internal checking that the KDC normally performs at start-up to ensure that the principal name in the certificate is consistent with the configured realm.

The default value is `false`.

This command is not required.

Example:

```
allow kdc cert principal name mismatch = true
```

19.3.4 allow kdc cert realm mismatch =

The KDC certificate is normally required to contain a realm name that matches the configured realm of the KDC. (The security specification requires an MTA to reject a

response in which the realm in the certificate does not match the local realm name.) However, sometimes it is useful in a lab environment to violate this requirement. This command, when `true`, turns off the internal checking that the KDC normally performs at start-up to ensure that the realm name in the certificate matches the configured realm.

The default value is `false`.

This command is not required.

Example:

```
allow kdc cert realm mismatch = true
```

19.3.5 check ip address =

The security specification requires the KDC to check that the source IP address of incoming requests matches the contents of the `caddr` field in the request. Under some lab configurations, it may be difficult to arrange for the IP address of the MTA to match the address in the `caddr` field. Setting `check ip address` to `false` turns off this check at the KDC. Note that setting this to `false` will, by definition, result in non-compliant behaviour on the KDC. It is strongly recommended that you do not set this value to `false` unless your lab configuration requires it. This parameter does not affect other checks against the `caddr` field; it turns off only the check that ensures that the value of the field matches the source IP address of the incoming datagram.

The default value is `true`.

This command is not required.

Example:

```
check ip address = false
```

19.3.6 console debug level =

This controls the number of messages written to the console while the KDC is running. The KDC supports eight levels of debugging messages, numbered 0 through 7. Most messages are produced at level 5. Running at level 7 produces an extremely verbose log. In ordinary operation, it is probably reasonable to run at console debug level 4. See also the `log debug level` command.

The default value is `7`.

This command is not required.

Example:

```
console debug level = 4
```

19.3.7 delete old logs =

KDC logs are written to the directory `<KDC>/log`. This command, if `true`, instructs the KDC, at startup, to delete any old logs remaining in the `<KDC>/log` directory from prior runs. Note that the MSO multisite license cause usage files to be written to the log directory; these usage files are not affected by this command.

The default value is `false` (*i.e.*, old logs are not deleted).

This command is not required.

Example:

```
delete old logs = true
```

19.3.8 dh store size =

The KDC can pre-calculated sets of Diffie-Hellman parameters. On slow machines, calculation of Diffie-Hellman parameters can take a perceptible amount of time, and if the calculation is performed during the process of generating the AS-REP, this can markedly reduce the throughput of the KDC. By setting this parameter, the user can control the number of Diffie-Hellman parameters that are pre-calculated and stored for use during an AS-REQ/AS-REP exchange. The KDC monitors the remaining number of parameters in storage as parameters are removed (for use in the AS-REP messages), the KDC. When the number of stored parameters falls below a critical level, the KDC will perform background calculations to re-fill the store.

The default value is `100`. Under normal circumstances, and when running the KDC on a reasonably powerful machine, there should be no reason to change this value. A value of `0` disables this feature entirely, so that Diffie-Hellman parameters are always calculated at the time at which they are needed.

This command is not required.

Example:

```
dh store size = 1000
```

19.3.9 drop dos datagrams =

KDCs are good targets for denial-of-service (DoS) attacks. If this parameter is `true`, then the KDC will attempt to determine when a DoS attack is taking place, and if it determines that an MTA is attacking it, it will begin to drop packets from that MTA.

In a lab situation, it is usually best to set this value to `false`, otherwise the KDC may appear to be failing to respond to valid requests because it believes that a DoS attack is taking place.

The default value is `true`.

This command is not required.

Example:

```
drop dos datagrams = false
```

19.3.10 dynamic service keys =

By default, the KDC will operate with support for dynamic service keys as described in Section 19.13. This means that the KDC will ordinarily respond to requests for dynamic service keys from other network entities. Setting this command to `false` turns off this support.

The default value is `true`.

This command is not required.

Example:

```
dynamic service keys = false
```

19.3.11 element id =

Operators may wish to assign a unique numerical Element ID to a KDC. The value of the Element ID is set with this command.

The default value is `0`.

This command is not required.

Example:

```
element id = 12345
```

19.3.12 force error reply to as req =

This command will cause the KDC to respond to AS-REQ messages by sending a KRB-ERR.

This command is obeyed only when the KDC is in non-compliant mode.

The default value is `false`.

This command is not required.

Example:

```
force error reply to as req = true
```

19.3.13 force error reply to tgs req =

This command will cause the KDC to respond to TGS-REQ messages by sending a KRB-ERR.

This command is obeyed only when the KDC is in non-compliant mode.

The default value is `false`.

This command is not required.

Example:

```
force error reply to tgs req = true
```

19.3.14 FQDN =

This defines the fully qualified domain name for the KDC. By convention, domain names are rendered in lower case. The FQDN should not end with a concluding dot.

There is no default value for this command.

This command is required.

Example:

```
FQDN = kdc.ipfonix.com
```

19.3.15 interface address =

This command informs the KDC the value of the IP address (in dotted decimal notation) on which it should expect to receive Kerberos messaging.

There is no default value for this command.

This command is required.

Example:

```
interface address = 10.1.2.3
```

19.3.16 log debug level =

This controls the number of messages written to the log file while the KDC is running. The KDC supports eight levels of debugging messages, numbered 0 through 7. Most messages are produced at level 5. Running at level 7 produces an extremely verbose log; this level is recommended for vendors when debugging their own code against the KDC. If you need to send a logfile to IPfonix, Inc., we strongly recommend that you send one generated with the debug level set to 7. See also the `console debug level` command.

The default value is `7`.

This command is not required.

Example:

```
log debug level = 7
```

19.3.17 log timer =

A timer fires inside the KDC once every thirty seconds. Ordinarily the timer fires silently. By setting this command to `true`, a DEBUG-level notification will be sent to the console and the log whenever the timer fires.

The default value is `false`.

This command is not required.

Example:

```
log timer = true
```

19.3.18 maximum client clock skew =

This defines the maximum allowed clock skew between the client and the KDC, in seconds. Requests that contain a clock skew greater than this value will be rejected with a clock skew error. For PacketCable-compliant operation, this value should be set to `300` (*i.e.*, five minutes). However, it is common in testing labs to set this value to a very large number (several days or weeks) so that statically-generated AS-REQ messages, or dynamic AS-REQ messages that contain static clock times, may be used without returning a clock skew error.

The default value is `300`.

This command is not required.

Example:

```
maximum client clock skew = 1000000
```

19.3.19 maximum log file size =

Ordinarily, the KDC generates a new log file as result of the UTC clock rolling over to a new day (*i.e.*, a logfile corresponds to a single day). By using the `maximum log file size` command, the KDC will instead create a new log file when the current file reaches the specified size (the size is specified in kB).

Normally, log files are named in a manner that encodes the time at which the logfile was generated; this is inappropriate when the log files are generated on the basis of size. Therefore, when this command is used, the current logfile (which still encodes the time at which the KDC was booted, so it follows the form `IPfonix*.log`) becomes `IPfonix*.log.1`; the file that was `IPfonix*.log.1` becomes `IPfonix*.log.2`, etc. The total number of log files can be limited with the `n saved log files` command.

There is no default value for this command.

This command is not required.

Example:

```
maximum log file size = 1000
```

19.3.20 maximum ps clock skew =

This defines the maximum allowed clock skew between the KDC and any provisioning servers, in seconds. According to the security specification, the value of this parameter must not exceed 3600 (*i.e.*, one hour). Any clock skew error returned by a provisioning server will be ignored if the skew exceeds this value (as demanded by the security specification).

The default value is `3600`.

This command is not required.

Example:

```
maximum ps clock skew = 1800
```

19.3.21 maximum ticket duration =

This defines the maximum duration for tickets generated by the KDC. The default unit is hours; by appending ‘m’ or ‘d’, the units can be changed to minutes or days respectively. This parameter is used to spread the lifetime of tickets so as to smooth out any “bump” in ticket requests caused by the fact that most deployments occur during working hours. The default value is 168 (*i.e.*, seven days), and it is recommended that this value not be

changed. Note that, in order for the KDC to meet the security specification, this parameter must not be set larger than 168.

The default value is `168`.

This command is not required.

Example:

```
maximum ticket duration = 6d
```

19.3.22 maximum ps backoff =

This defines the maximum time, in tenths of a second, that the KDC will wait after transmitting an MTA FQDN message before declaring that no response has been received. Unless your lab contains a rather slow provisioning server, this value should be left at its default value, which is 50 (*i.e.*, five seconds).

Note that this is the maximum value of a retry timer. Retries begin at the value given by the `minimum ps backoff` parameter (which defaults to 0.5 seconds) and increase with exponential delays until they reach the value specified by this parameter.

The default value is `50`.

This command is not required.

Example:

```
maximum ps backoff = 100
```

19.3.23 minimum ps backoff =

This defines the minimum time, in tenths of a second, that the KDC will wait after transmitting an MTA FQDN message before declaring that no response has been received. The default value is 5 (*i.e.*, 0.5 seconds).

Note that this is the minimum value of a retry timer. Retries begin at the value given by this parameter and increase with exponential delays until they reach the value specified by the `maximum ps backoff` parameter (which defaults to 5 seconds).

The default value is `5`.

This command is not required.

Example:

```
minimum ps backoff = 10
```

19.3.24 minimum ticket duration =

This defines the minimum duration (in hours) for tickets generated by the KDC. The default unit is hours; by appending ‘m’ or ‘d’, the units can be changed to minutes or days respectively. This parameter is used to spread the lifetime of tickets so as to smooth out any “bump” in ticket requests caused by the fact that most deployments occur during working hours. The default value is 144 (*i.e.*, six days), and it is recommended that this value not be changed.

The default value is `144`.

This command is not required.

Example:

```
minimum ticket duration = 90m
```

19.3.25 n saved log files =

This command defines the number of old log files (from the current run) that the KDC saves. These logfiles may be generated on the basis of date (*i.e.*, one logfile per day, which is the default behaviour) or size (see the `maximum log file size` command).

The default value is `7`.

This command is not required.

Example:

```
n saved log files = 10
```

19.3.26 pc20 =

This command instructs the KDC to operate in accordance with PacketCable 2.0. In particular, this enables IPv6 support and support for EUE messaging on the interface to the Provisioning Server.

The default value is `false`.

This command is not required.

Example:

```
pc20 = true
```

19.3.27 peer<n> =

The IPfonix, Inc. KDC can be networked in a set of up to ten peers that maintain a consistent global state across the peered devices. To define a set of peers, each KDC should identify its peers in the `[general]` section of the `kdc.ini` file. The syntax is easy: simply enter a command of the form:

```
peer<n> = IP address
```

for each peer, where `<n>` is a digit in the range 0 through 9. It is not necessary for the relationship between peer number and IP address to be consistent across the peered group. Neither is it necessary for `<n>` to increment sequentially in the list of peers.

In order for the peering function to be enabled, the command `peering = true` must be present in the `kdc.ini` file.

This command has no default value.

This command is not required.

Example:

Suppose that we wish to form a peered group of three KDCs located at the IP addresses 192.168.0.1, 192.168.0.2 and 192.168.0.3. Then the `kdc.ini` file for the device at 192.168.0.1 might have an entry that looks like this:

```
peer2 = 192.168.0.2
peer3 = 192.168.0.3
```

The entry for the KDC at 192.168.0.2 might look like this:

```
peer0 = 192.168.0.1
peer3 = 192.168.0.3
```

And the entry for the KDC at 192.168.0.3 could conceivably contain:

```
peer5 = 192.168.0.2
peer3 = 192.168.0.1
```

19.3.28 peering =

This controls the peering function. If `true`, then the peering function is enabled, if `false`, it is disabled.

The default value is `false`.

This command is not required.

Example:

```
peering = true
```

19.3.29 ping =

This controls the a ping function that allows a remote system to determine quickly whether the KDC is up and processing incoming messages. When this parameter is `true`, then the KDC will respond to an incoming UDP datagram on port 88 that comprises the ASCII string “ping” by sending a response datagram comprising the word “pong” to the source port on the device that sent the ping message.

The default value is `false`.

This command is not required.

Example:

```
ping = true
```

19.3.30 provisioning server is available =

This defines whether a PacketCable Provisioning Server is available to the KDC. PacketCable requires that, during the MTA provisioning process, the KDC contact a provisioning server to check whether the MTA that is requesting a ticket is in fact properly known to the operator. Setting the value of this command to `false` tells the KDC that in fact no such provisioning server is available. As a result, the MTA FQDN messaging is not performed, and the KDC acts as if such messaging has been performed and a positive response has been received from a provisioning server.

The default for this command is `true`.

This command is not required.

Example:

```
provisioning server is available = false
```

19.3.31 ps message cache size =

This defines the size of the FIFO cache that the KDC generates to store non-error responses from provisioning servers. Making this cache a non-zero size means that, for a normal provisioning boot sequence on an MTA, the KDC will cache the initial response from the provisioning server (*i.e.*, while processing the AS-REQ for the provisioning server ticket). When, a few seconds later, the MTA requests a ticket for a CMS, the provisioning server will not need to be re-contacted, since its prior response will be in the cache. Note that only non-error responses are stored in this cache.

The default value for this command is `0` (*i.e.*, the cache is turned off by default).

This command is not required.

Example:

```
ps message cache size = 25
```

19.3.32 realm =

This defines the realm name in which the KDC is operating. By convention, realm names are rendered in upper case. If this command is not present in the `kdc.ini` file, the KDC will obtain its realm name from the KDC certificate.

The default value for this command is the realm name contained in the KDC certificate.

This command is not required.

Example:

```
realm = IPFONIX.COM
```

19.3.33 replay cache size =

This defines the size of the FIFO cache that the KDC generates to store responses that have already been sent to MTAs. Making this cache a non-zero size means that, if an identical request arrives at the KDC, the previously-generated response will automatically be sent, so long as the request and response are found in the cache. Setting the value to zero will turn off caching of responses.

The default value for this command is `25`.

This command is not required.

Example:

```
replay cache size = 100
```

19.3.34 require critical keyusage =

An ambiguity in the security specification means that vendors of clients are technically permitted to include in certificates keyUsage extensions that are not critical. By default, the IPfonix, Inc. KDC conforms to the required behavior as defined by CableLabs for compliance: *i.e.*, it will accept such certificates.

However, operators may wish to permit acceptance only of certificates in which the keyUsage extension is marked as critical (which was the intent of the original specification, and is the practice of most vendors). The KDC can be configured to reject certificates that do not mark the keyUsage extension as critical by setting this command to `true`.

In order for this command to be executed, the KDC must be operated in non-compliant mode.

The default value for this command is `false`.

This command is not required.

Example:

```
require critical keyusage = true
```

19.3.35 require ip address in requests =

The security specification requires the presence of an IP address in incoming Kerberos requests. Since early versions of the security specification did not require the IP address to be present, the KDC allows the operator to operate in a non-compliant mode in which it does not require an IP address in inbound requests. To turn off this requirement, set this command to `false`.

The default value for this command is `true`.

This command is not required.

Example:

```
require ip address in requests = false
```

19.3.36 respond from port 88 =

Normally, the KDC will send its responses from a non-system UDP port (*i.e.*, a port with a value greater than 1023). Setting this value to `true` forces the KDC to respond from UDP port 88. This option is available because some Kerberos clients incorrectly assume that Kerberos messages from Kerberos servers must originate on port 88. Also, setting this parameter to `true` helps some packet sniffers to identify Kerberos messages coming from the KDC.

The default value for this command is `false`.

This command is not required.

Example:

```
respond from port 88 = true
```

19.3.37 send root certificate =

This parameter controls whether the Service Provider Root certificate is included in the AS-REP. The security specification defines the behaviour of an MTA when it receives a root certificate in the reply.

The default value for this command is `false`.

This command is not required.

Example:

```
send root certificate = true
```

19.3.38 support first oakley group =

The security specification requires support for the second Oakley group. It also says that KDCs MAY support the first Oakley group. By default the IPfonix KDC will reject MTA requests that use the first Oakley group (this makes sense, because, since a KDC is required to support only the second, MTAs should normally use the second Oakley group by default). Setting this parameter to `true` will cause the KDC to accept correctly formatted requests that use the first Oakley group. Note that, for this command to be processed correctly, `oakley` must be rendered entirely in lower case.

The default value for this command is `false`.

This command is not required.

Example:

```
support first oakley group = true
```

19.3.39 ticket extension =

Devices that accept PacketCable-compliant tickets must be prepared to deal with the posiibility that they contain an extension (because it is legal for a KDC to insert an extension; in some circumstances, in communication with some devices, it is mandatory that the KDC do so). Tickets issued to MTAs by the IPfonix, Inc. KDC do not ordinarily contain extensions. However by setting `ticket extension` to `true`, the KDC will include an extension in the ticket. This allows users to test whether the final recipient of the ticket can correctly process a ticket that contains an extension.

The default value of this parameter is `false`.

This command is not required.

Example:

```
ticket extension = true
```

19.3.40 worker threads =

The KDC uses a number of internal “worker” threads to process incoming requests. This command allows the user to change the number of such threads. The only time that this may improve performance noticeably is in the case where the KDC is being asked to process a rapid burst of incoming requests and the provisioning server is unable to

generate MTA FQDN reply messages sufficiently quickly. In this case, the KDC will be left waiting for replies from the provisioning server, and most or all of the threads will be idling, but none will be available to process additional incoming requests; this will result in the KDC dropping requests until the provisioning server has cleared the backlog.

The maximum value for this command is limited by the underlying operating system, but in general the value should be changed by relatively small increments, and values in excess of about 25 are unlikely to improve burst throughput.

The default value for this command is `10`.

This command is not required.

Example:

```
worker threads = 20
```

19.4 [jabber] section

The `[jabber]` section is used to control high-level messages that may be sent to a jabber client[7].

19.4.1 destination =

This identifies the jabber ID (JID) of the client to which messages will be sent.

There is no default value for this command.

This command is not required.

Example:

```
destination = jabclient@jabber.ipfonix.com
```

19.4.2 jid =

This identifies the jabber ID (JID) of the KDC. If no resource is included, then the resource "`/kdc`" will be automatically added to the JID.

There is no default value for this command.

This command is not required.

Example:

```
jid = kdc@jabber.ipfonix.com/kdc
```

19.4.3 password =

This gives the password that the KDC will use to log on to the jabber server.

There is no default value for this command.

This command is not required.

Example:

[7]Jabber support is available only in the Linux and Windows versions of the KDC.

```
password = kdc_password
```

19.5 [ocsp] section

The `[ocsp]` section is used to control the behavior of the OCSP client in the KDC.

19.5.1 check device certificates =

When running the OCSP client, the KDC will always check device manufacturer certificates. By default, it will also check individual device certificates. Setting this command to `false` will cause it not to check individual device certificates.

The default value is `true`.

This command is not required.

Example:

```
check device certificates = false
```

19.5.2 default =

This controls the assumed status of a certificate for which the KDC is unable to obtain a definitive response.. It may take the values `good` or `revoked`. Setting the value to `revoked` is more secure than `good` but may be impractical in practice, unless a high-availability OCSP responder is used.

The default value is `good`.

This command is not required.

Example:

```
default = revoked
```

19.5.3 interface address =

This command informs the KDC the value of the IP address (in dotted decimal notation) on which it should send messages to the OCSP responder.

The default value is the same as the interface address for the Kerberos messages (set in the `[general]` section.

This command is not required.

Example:

```
interface address = 127.0.0.1
```

19.5.4 max skew =

This determines the maximum amount of allowed clock skew (in seconds) between the OCSP client and the OCSP responder.

The default value is `300`.

This command is not required.

Example:

```
max skew = 60
```

19.5.5 ocsp =

This determines whether the OCSP client is turned on. It may take the values `true` or `false`.

The default value is `false`.

This command is not required.

Example:

```
ocsp = true
```

19.5.6 port =

This identifies the destination TCP port on the OCSP responder to which OCSP requests will be sent.

The default value is `80`.

This command is not required.

Example:

```
port = 8008
```

19.5.7 responder address =

The address to which the OCSP client should send OCSP requests.

The default value is `127.0.0.1`.

This command is not required.

Example:

```
responder address = 10.168.0.1
```

19.5.8 timeout =

This controls the length of the period (in seconds) for which the KDC will wait for a response from an OCSP responder.

The default value is `10`.

This command is not required.

Example:

```
timeout = 20
```

19.6 [pkcross] section

The `[pkcross]` section is used to control values related to the PKCROSS protocol.

19.6.1 lifetime =

This defines the duration for PKCROSS tickets generated by the KDC. The default unit is hours; by appending ‘m’ or ‘d’, the units can be changed to minutes or days respectively. Note that, in order for the KDC to meet the security specification, this parameter must not be set larger than 168 (which corresponds to seven days).

The default value is `24` (*i.e.*, one day).

This command is not required.

Example:

```
lifetime = 2d
```

19.6.2 maximum backoff =

This defines how long, in tenths of a second, the KDC will wait, in the worst case, for the reply from a KDC in another realm.

The default value is `50` (*i.e.*, five seconds).

This command is not required.

Example:

```
maximum backoff = 100
```

19.6.3 maximum skew =

Defines the maximum allowed clock skew, in seconds, during a PKCROSS exchange.

The default value is `300` (*i.e.*, five minutes).

This command is not required.

Example:

```
maximum skew = 3600
```

19.6.4 nameserver =

This is the IP address, in dotted decimal notation, of the nameserver that the KDC should use when attempting to perform the SRV lookup of a remote KDC.

The default value is `127.0.0.1` (*i.e.*, the local KDC itself).

This command is not required.

Example:

```
nameserver = 192.168.0.25
```

19.7 [pseudo ps] section

The `[pseudo ps]` section is used to control the PacketCable pseudo-provserver code (see section 14).

19.7.1 allow non-standard source port =

According to the security specification, requests to a provisioning server must occur from port number 1293. Setting this option to `true` causes the KDC to accept requests from any client port.

The default value is `false`.

This command is not required.

Example:

```
allow non-standard source port = true
```

19.7.2 allow wakeup =

Normally, the pseudo ps will not allow a value of zero for the server nonce in AP Request messages. If this parameter is set to `true`, the KDC will permit the nonce to contain zero.

The default value is `false`

This command is not required.

Example:

```
allow wakeup = true
```

19.7.3 append to leak keys file =

When leaking keys to a file with the `leak keys file` command, the KDC by default will overwrite any existing file. Setting `append to leak keys file` to `true` causes the KDC to append keys to a pre-existing file instead of deleting any pre-existing contents.

The default value is `false`

This command is not required.

Example:

```
append to leak keys file = true
```

19.7.4 ap-rep subkey =

Normally, a new subkey is generated every time that the pseudo-ps sends an AP-REP. Using this command allows the user to pre-define the subkey that will be used in all AP-REP messages, which can be useful for testing and debugging purposes. The subkey is a hex value encoding 46 octets, of the form *0x01ab23cd...*.

There is no default value.

This command is not required.

Example (should be all on one line in the `kdc.ini` file):

```
ap-rep subkey =
0xa0b26591bc05f1e60a9b5c7d2e6fa0b244e629754f10f3002991f43f3
f8209dc867f32a0ca6f779f108abc76390fff28
```

19.7.5 ciphersuites =

This command lists the acceptable ciphersuites that may be submitted by the MTA in the AP-REQ. Each acceptable ciphersuite is in the form `xx/yy` where `xx` is the hexadecimal value of the authentication algorithm and `yy` is the hexadecimal value of the encryption algorithm. Acceptable ciphersuites are separated by spaces.

The default value is the single ciphersuite: `21/20`.

This command is not required.

Example:

```
ciphersuites = 21/21 21/20
```

19.7.6 delay wakeup keys =

This command allows the operator to control the time at which keys are leaked. Normally, they are leaked as soon as they are sent to the client; however, this raises the possibility that a key will be leaked that is never acknowledged or used by the client. By setting this parameter to `true`, the operator can delay the leakage until the client has acknowledged the new key.

The default value is `false`.

This command is not required.

Example:

```
delay wakeup keys = true
```

19.7.7 eue config file hash =

This provides the hash of the configuration file that is to be returned to an EUE client in provisioning step MTA-19. The format is a string of hexadecimal characters. The total length of the string is either 32 octets (for MD5) or 40 octets (for SHA-1).

There is no default value for this command.

This command is required if the pseudo provserver function is turned on.

Example (all on one line in the actual `kdc.ini` file):

```
eue config file hash =
1234567890123456789012345678 90aabbccddee
```

19.7.8 eue config file key =

This provides the encryption key of the configuration file that is to be returned to an EUE client in provisioning step MTA-19. The format is a string of hexadecimal characters.

If this string is empty, then the configuration file is assumed to be unencrypted. The total length of a non-empty string must be 16 octets.

There is no default value for this command.

This command is not required.

Example:

```
eue config file key = 1234567890aabbcc
```

19.7.9 eue config url =

This defines the URL of the configuration file that is to be returned to an EUE client in provisioning step MTA-19.

There is no default value for this command.

This command is required if the pseudo provserver function is turned on.

Example (all on one line in the actual `kdc.ini` file):

```
eue config url =
tftp://tftp.ipfonix.com/default_config_file.txt
```

19.7.10 force error reply to ap req =

This command will cause the KDC to respond to AP-REQ messages by sending a KRB-ERR.

The default value is `false`.

This command is not required.

Example:

```
force error reply to ap req = true
```

19.7.11 interface address =

This command informs the KDC the value of the IP address (in dotted decimal notation) on which it should expect to receive messages for the pseudo provserver.

The default value is the same as the interface address for the Kerberos messages (set in the `[general]` section.

This command is not required.

Example:

```
interface address = 10.1.2.4
```

19.7.12 leak keys file =

This provides a file into which SNMP keying information will be leaked. The format of the entries in this file is defined in section 14.1; note that the entries are unencrypted, so this command should be used with care.

There is no default value for this command.

This command is not required.

Example:

```
leak keys file = mtakeys.txt
```

19.7.13 leak keys named pipe =

This provides a named pipe into which SNMP keying information will be leaked. The format of the entries in this pipe is defined in section 14.1; note that the entries are unencrypted, so this command should be used with care.

There is no default value for this command.

This command is not required.

Example:

```
leak keys name pipe = fifo.buffer
```

19.7.14 mta config file hash =

This provides the hash of the configuration file that is to be returned to the client in provisioning step MTA-19. The format is a string of hexadecimal characters. The total length of the string is either 32 octets (for MD5) or 40 octets (for SHA-1).

There is no default value for this command.

This command is required if the pseudo provserver function is turned on.

Example (all on one line in the actual `kdc.ini` file):

```
mta config file hash =
12345678901234567890123456789 0aabbccddee
```

19.7.15 mta config file key =

This provides the encryption key of the configuration file that is to be returned to the client in provisioning step MTA-19. The format is a string of hexadecimal characters.

If this string is empty, then the configuration file is assumed to be unencrypted. The total length of a non-empty string must be 16 octets.

There is no default value for this command.

This command is not required.

Example:

```
mta config file key = 1234567890aabbcc
```

19.7.16 mta config url =

This defines the URL of the configuration file that is to be returned to the client in provisioning step MTA-19.

There is no default value for this command.

This command is required if the pseudo provserver function is turned on.

Example (all on one line in the actual `kdc.ini` file):

```
mta config url =
tftp://tftp.ipfonix.com/default_config_file.txt
```

19.7.17 pc 1.5 =

There is an inconsistency between the requirements in PacketCable 1.0 and PacketCable 1.5. PacketCable 1.5 requires that pktcMtaDevProvConfigKey not be sent to an MTA if the configuration file is not encrypted. However, PacketCable 1.0 requires that it be sent. This parameter is used to control whether the pseudo ps conforms to PacketCable 1.0 or PacketCable 1.5.

The default value is `false`. (That is: by default, the pseudo ps conforms to PacketCable 1.0.)

This command is not required.

Example:

```
pc 1.5 = true
```

19.7.18 ping =

This controls the a ping function that allows a remote system to determine quickly whether the KDC is up and processing incoming messages. When this parameter is `true`, then the KDC will respond to an incoming UDP datagram on the psudo-ps port that comprises the ASCII string “ping” by sending a response datagram comprising the word “pong” to the source port on the device that sent the ping message. Internally, the incoming message is handed to a worker thread (if one is free) and processed by that thread, just as if it were a Kerberos message.

The default value is `false`.

This command is not required.

Example:

```
ping = true
```

19.7.19 pseudo ps =

This controls whether the pseudo-provserver code is enabled.

The default value for this command is `false`.

This command is not required.

Example:

```
pseudo ps = true
```

19.7.20 pseudo ps engine id =

This defines the SNMP engine ID for the pseudo-provserver. If the value is preceded with `0x`, then it is interpreted as a hex string. Otherwise, the value is interpreted as a literal string.

The default value for this command is `IPfonix pseudo ps engine`.

This command is required if the pseudo provserver function is turned on.

Example 1:

```
pseudo ps engine id = 0x112233abccfebd
```

Example 2:

```
pseudo ps engine id = Operator engine ID
```

19.7.21 pseudo ps fqdn =

This defines the Fully Qualified Domain Name for the pseudo-provserver.

There is no default value for this command.

This command is required if the pseudo provserver function is turned on.

Example:

```
pseudo ps fqdn = pseudops.ipfonix.com
```

19.7.22 pseudo ps prov filename =

In addition to the commands `pseudo ps config file hash`, `pseudo ps config file key` and `pseudo ps config url`, the pseudo-ps can read a file to obtain values that will be returned when appropriate in MTA-19.

Using the `pseudo ps prov filename` command allows the KDC to load multiple values of {hash, key, URL}, so that different MTAs (*e.g.*, MTAs from different manufacturers) can be provided with different files.

When the pseudo-ps processes an incoming MTA-15 INFORM, it examines the INFORM to identify the value of pktcMtaDevTypeIdentifier. If there is a match with the contents of the file identified by the `pseudo ps prov filename` command, the values from that file are used; if here is no match, then the values from the `kdc.ini` file are used in the MTA-19 response.

There is no default value for this command.

The format of the contents of the `pseudo ps prov filename` file follows that of a standard `.ini` file:

```
[value of pktcMtaDevTypeIdentifier]
url = <URL of config file>
hash = <hash of config file as hex characters>
key = <key of config file as hex characters>
```

The “key” line is optional.

So a complete example would be:

- in the `kdc.ini` file:

```
pseudo ps prov filename = prov_info.ini
```

- in the `prov_info.ini` file:

```
[pktc1.0:05190101000201020B0501050608090C01010D01010F0101100107]
url = http://kdc.ipfonix.com/config.txt
hash = 12345678901234567890123456789012345678 90
key =

[pktc1.0:05190101000201020B0501050608090C01010D01010F0101100108]
url = http://kdc.ipfonix.com/config.1.txt
hash = aabbccddeeffeeddccbbaabbccddeeffeeddccbb
key = 1234567890aabbcc
```

19.7.23 pseudo ps snmp port =

This identifies the port on which the pseudo-provserver will listen for incoming SNMP messages.

The default value for this command is `162`.

This command is not required.

Example:

```
pseudo ps snmp port = 10162
```

19.7.24 snmp enable =

Enables processing of SNMP messages from MTAs. If this is `false`, then such messages are ignored by the pseudo-provserver.

The default value for this command is `true`.

This command is not required.

Example:

```
snmp enable = false
```

19.7.25 worker threads =

The pseudo-provserver is multi-threaded. This command may be used to change the number of threads used by the pseudo-provserver. Under normal circumstances, it is unlikely that the user should change this value.

The default value for this command is 10

This command is not required.

Example:

```
worker threads = 15
```

19.8 [testing] section

The `[testing]` section contains commands that cause output to be written to disk in a more convenient form than the ordinary log file. Apart from the `directory` command, all commands are of the form:

`<message type or internal variable> = <true or false>`

If the value is set to `true`, then any instances of that particular message type (or the value of the internal variable at the time it is used) are written, in real time, to a pair of files in the directory named by the `directory` command. One file has the extension `.bin`, and contains the raw binary of the message; the other file has the extension `.hex` and contains a hex dump of the same information. The filename also encodes the type of the message or the name of the variable and the IP address of the client, so that it is easy to associate a particular file with a particular client. Any incoming message of the same type and from the same client will overwrite any pre-existing file of that type from the same client.

So, for example, the file `192.168.0.1.asreq.bin` contains the most recent AS-REQ message exactly as received from the client 192.168.0.1. The file `192.168.0.1.asreq.hex` contains the same information, but in hex format.

By default, dumps for all message types are set to `false`. The default dump directory is `<KDC>/testing/`.

19.8.1 as req =

Controls writing of files that contain AS-REQ messages.

19.8.2 as rep =

Controls writing of files that contain AS-REP messages.

19.8.3 dh private value =

Controls writing of files that contain the value of the Diffie-Hellman private value.

19.8.4 dh public value =

Controls writing of files that contain the value of the Diffie-Hellman public value.

19.8.5 dh shared secret =

Controls writing of files that contain the value of the calculated Diffie-Hellman shared secret.

19.8.6 directory =

Defines the name of the directory (relative to `<KDC>`) into which the files are to be placed. This directory must exist.

19.8.7 krb error =

Controls writing of files that contain KRB-ERROR messages.

19.8.8 local system cert =

Controls writing of files that contain local system certificates received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.9 manufacturer cert =

This is used to control writing of files that contain Manufacturer certificates received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.10 mta device cert =

Controls writing of files that contain MTA Device certificates received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.11 mta fqdn req =

Controls writing of files that contain MTA FQDN Request messages.

19.8.12 mta fqdn rep =

Controls writing of files that contain MTA FQDN Reply messages.

19.8.13 mta fqdn session key =

Controls writing of files that contain the session key for MTA FQDN messages.

19.8.14 mta manufacturer cert =

Controls writing of files that contain MTA manufacturer certificates received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.15 packetcable server cert =

Controls writing of files that contain PacketCable Server certificates (as defined in Table 43 of the PacketCable security specification) received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.16 ps element cert =

This is used to control writing of files that contain PS Element certificates received in client messages. If the certificate causes a fatal parsing error, the KDC may not execute this command.

19.8.17 received datagram =

Controls writing of files that contain datagrams exactly as they are received on port 88 from clients, prior to any processing to determine the message type.

19.8.18 service key =

Controls writing of files that contain the service key used to encrypt a ticket.

19.8.19 session key =

Controls writing of files that contain the session key returned in tickets.

19.8.20 tgs req =

Controls writing of files that contain TGS-REQ messages.

19.8.21 tgs rep =

Controls writing of files that contain TGS-REP messages.

19.8.22 ticket =

Controls writing of files that contain tickets. The tickets are exactly as sent to the MTA (*i.e.*, they are encrypted with the corresponding service key).

19.9 [checks] section

The `[checks]` section should only be used if you need to set the KDC so that it does not perform checks that are required by the security specification. By default, all these checks are set to `true`. Setting any of these checks to `false` results in a non-compliant KDC. More importantly, the KDC was not designed to run without these checks. The options in the `[checks]` section have been added at the request of various MTA vendors in order to help them perform test partial interoperability when the non-KDC device is not yet completely compliant to the security specification.

Whenever a test is skipped, there is a possibility that the KDC will behave in an unexpected manner, since the rest of the KDC code will assume that the test has been applied and passed. In other words, skipping tests may result in contracts in the internal KDC code being violated. If you see unexpected KDC behaviour, please turn all the checks back on and make sure that the unexpected behaviour persists before reporting a problem.

19.9.1 check authority key identifier =

Controls checking of the authorityKeyIdentifier extension in certificates received in an AS-REQ.

19.9.2 check basic constraints =

Controls checking of the basicConstraints extension in certificates received in an AS-REQ.

19.9.3 check kdc cert =

Controls checking of the validity of the KDC certificate at start-up time.

19.9.4 check key usage =

Controls checking of the keyUsage extension in certificates received in an AS-REQ.

19.9.5 check local system cert =

Controls checking of the validity of the Local System certificate at start-up time.

19.9.6 check manufacturer cert =

This is used to control all checking of the Manufacturer certificate received in an AS-REQ.

19.9.7 check manufacturer certificate signature =

This is used to control checking of the signature in the Manufacturer certificate.

19.9.8 check mta device cert =

Controls all checking of the MTA device certificate received in an AS-REQ.

19.9.9 check mta device certificate signature =

Controls checking of the signature in the MTA device certificate.

19.9.10 check mta manufacturer cert =

Controls all checking of the MTA manufacturer certificate received in an AS-REQ.

19.9.11 check mta manufacturer certificate signature =

Controls checking of the signature in the MTA manufacturer certificate.

19.9.12 check pkauth checksum =

Controls whether the pkAuth checksum field in the AS-REQ is verified.

19.9.13 check ps element cert =

This is used to control all checking of the PS Element certificate received in an AS-REQ.

19.9.14 check ps element certificate signature =

This is used to control checking of the signature in the PS Element certificate.

19.9.15 check service provider cert =

Controls checking of the validity of the Service Provider certificate at start-up time.

19.10 [violations] section

The `[violations]` section is used to control most of the negative-testing capability of the KDC. Prudent security design requires that devices that receive messages should check every field of a received message to ensure that it matches the expected contents of that field[8]. Not only is this prudent, it is required by the security specification. Therefore, if a device that receives messages from a KDC (either directly or indirectly – for example, in the form of a ticket) is to be properly tested for conformance, the KDC must be capable of creating messages that are purposefully non-compliant in a controlled manner. This allows users and systems integrators to ensure that error conditions are handled correctly on the recipient of the KDC messages. The `[violations]` section of the `kdc.ini` file is designed to facilitate the process of having the KDC produce messages that *do not comply* with the detailed requirements of the security specification.

19.10.1 Numbered violations

Most entries in the `[violations]` section take the form:

`r<nnnn> = <boolean>`

where `<nnnn>` is a requirement number extracted from the PacketCable Security Compliance Test Plan and `<boolean>` is either `true` or `false`. (Actually, if the value of `<boolean>` is not precisely equal to `true`, it will be construed as being `false`.) By default all violations are assumed to be `false`.

The KDC supports the values for `<nnnn>` listed in Table 2.

Table 2

<nnnn>	Comment
1928	encryption type not des3-cbc-md5
1929.1	digest algorithm not SHA-1 in AS-REP
1929.2	incorrect nonce (setting the value to "nonzero" forces the nonce in the response to be nonzero but incorrect; setting the value to "true" sets it to zero)
1929.3	no KDC certificate
1929.5	no Service Provider certificate
1929.9	extra signerInfo in signerInfos
1929.13	digest algorithm not SHA-1 in signerInfo
1929.14	signature algorithm not RSA in signerInfo
1940.1	typed-data is not REQ-NONCE in KRB-ERROR
1940.2	add checksum even when have no shared key
1943.1	client time not filled out in skew error
1948.1	incorrect pre-auth in reply
1951	forbidden fields added to AS-REP
1953	incorrect calculation of encryptedData
1954	forbidden options added to ticket (in AS-REP only)
1955	encryption type in ticket not set to des3-cbc-md5

[8]This is directly opposed to the oft-cited Postel Robustness Principle ("be conservative in what you send, but liberal in what you accept"). The Robustness Principle, while having a wide range of applicability, should in general *not* be applied to security systems.

<nnnn>	Comment
1974	forbidden fields added to ticket in TGS-REP
1975	encryption type in TGS-REP not set to des3-cbc-md5
1976.1	no checksum included
1976.2	bad checksum type in KRB-ERROR return to TGS-REQ
1976.3	no REQ-NONCE in KRB-ERROR return to TGS-REQ
1976.4	incorrect value of checksum returned in KRB-ERROR to TGS-REQ
1963	REQ-NONCE not included in KRB-ERROR
1963.1	checksum not included in KRB-ERROR
2343	no boot-time check for cert version
2345	no check for whether cert is RSA
2350	allow no keyUsage extension
2351	allow no basicConstraints extension
2352	allow illegal values in basicConstraints
2353	forbidden signature algorithm
2874	wrong key type for session key in AS-REP
3012	incorrect checksum in encrypted portion of AS-REP
3014	key in ticket is not des3-cbc-md5
3016	incorrect value of checksum
3018.1	incorrect nonce in TGS-REP
3018.2	incorrect checksum in TGS-REP
3020	invalid client principal name
3023	server name type is not KRB_NT_SRV_HST
3023.1	server name has incorrect first component
3023.2	server name has incorrect second component
3025.1	MUTUAL-AUTHENTICATION flag is not set
3025.2	USE-SESSION-KEY flag is set
3025.3.1	no sequence number in authenticator
3025.3.2	disallowed OPTIONAL field is present in authenticator
3025.3.4	incorrect encryption type used in authenticator
3026.1	inconsistent seq-number used in KRB-SAFE
3026.2	disallowed OPTIONAL field present in KRB-SAFE
3026.3	incorrect checksum type in KRB-SAFE

For example, if the KDC is to be configured to violate requirements 1929.1 and 1951, then one conceivable, and particularly disorganised, version of the `[violations]` section that would ensure that these violations (and no others) occur looks like this:

```
[violations]
r1928 = false
r1929.1 = true
r1929.9 = false
r1976.4 = false
r1951 = true
r1929.5 = false
```

```
;r1948.1 = true
```

A more organised `kdc.ini` file that accomplishes the same result might contain:

```
[violations]
r1929.1 = true
r1951 = true
```

19.10.2 Unnumbered violations

In addition to the violations described in the previous section, a few entries take a different form, to cover the cases where there is no explicit requirement number in the security specification.

19.10.2.1 ap error checksum =

This controls whether the checksum in an error reply to an AP-REQ is valid.

19.10.2.2 ap error sequence number =

This controls whether the sequence number in an error reply to an AP-REQ is valid.

19.10.2.3 as rep signature =

This controls whether the signature in the AS-REP verifies correctly.

19.10.2.4 as rep nonce =

This controls violations in the nonce in the AS-REP. Allowed values are `true` (a zero nonce is returned), `false` (the nonce is the expected value) and `nonzero` (an invalid, nonzero nonce is returned).

19.10.2.5 checksum =

This controls whether the checksum in the ticket in the AS-REP is valid.

19.11 [ipv6 section]

The IPfonix, Inc. KDC includes support for dual-stack operation, in which it send and receive IPv6 messages in addition to the normal IPv4 messages[9]. The differences between IPv4 and IPv6 operation are described in section Error: Reference source not found.

This section describes the configuration parameters that may be placed in the `kdc.ini` file in order to control the operation of the IPv6 portion of the KDC.

19.11.1 enable =

This command is used to globally enable IPv6 support. In order to support *any* IPv6 feature, this parameter must be set to `true`. If this parameter is `true`, then the various IPv6 sub-features may be enabled. If this parameter is `false`, then any commands that enable any IPv6 sub-features are ignored.

The default value of this command is `false`.

This command is not required.

Example:

[9]Linux and Windows only.

```
enable = true
```

19.11.2 fqdn =

This command contains either an FQDN that resolves to an IPv6 address, or an IPv6 address.

There is no default value for this command.

This command is required when operating with IPv6 enabled..

Example:

```
fqdn = ::1
```

19.11.3 ipv6 in mta fqdn =

This command controls whether the KDC will accept IPv6 addresses in MTA FQDN Reply messages.

The default value of this command is `false`.

This command is not required.

Example:

```
ipv6 in mta fqdn = true
```

19.11.4 kerberos enable =

This command controls whether the KDC will listen for incoming Kerberos messages (AS-REQ, *etc*.) from clients using IPv6.

The default value of this command is `false`.

This command is not required.

Example:

```
kerberos enable = true
```

19.11.5 provserver enable =

This command controls whether the KDC will transmit to the PacketCable provisioning server over IPv6 (and similarly listen for the reply on IPv6).

The default value of this command is `false`.

This command is not required.

Example:

```
provserver enable = true
```

19.12 [statistics] section

The KDC can be configured to output ongoing statistical information about its operation. This information is sent to a named pipe (in Linux and Solaris only). This capability is controlled by a command in the `[statistics]` section.

19.12.1 named pipe =

This command gives the name of the named pipe to which statistics are to be sent. If this command is not present, then statistics are not output. The named pipe can be read by any user.

There is no default value for this command.

Thids command is not required.

Example:

```
named pipe = statistics-pipe
```

19.13 Minimal configuration

Because an IPfonix, Inc. KDC has so many possible configuration parameters, it can be daunting to configure the software for a particular network. However, most environments actually require very few of entries in the `kdc.ini` file. In particular, when operating the KDC in a fully compliant mode, the following is a complete example `kdc.ini` file:

```
[general]
interface address = 10.1.2.3
FQDN = kdc.ipfonix.com
```

Nothing else is required.

Users wishing to create a more complicated `kdc.ini` file may want to download the free program `buildini` from our website: (see http://www.ipfonix.com/buildini/buildini.html).

20 Dynamic Service Keys

Although the security specification implies that dynamic service keys should be supported, it currently does not place any requirements as to what messages might be used to implement this. Any mechanism used to support dynamic service keys should at a minimum, have the following properties:

1. It should allow previously unknown core network devices to register with the KDC and obtain a service key
2. It should associate an expiration time and version number with a key
3. It should (obviously) be as secure as reasonably possible. In particular, it should exhibit Perfect Forward Secrecy.

4. Preferably, it should re-use as many as possible of the messages and formats already required by the specification.

The IPfonix, Inc. KDC includes an implementation of a dynamic keying mechanism that is designed to meet these goals. We describe it here so that implementors of other devices on the network can use the provided mechanism to obtain service keys from the KDC without the need to pre-provision static keys into their devices.

As far as possible, the operations involved in obtaining a dynamic service key mirror those used by the MTA to establish shared secrets between itself and application servers.

In order to obtain a dynamic service key, the server should perform the following steps:

1. Perform a PKINIT exchange with the KDC. The requirements on this exchange exactly match the requirements on the PKINIT exchange already described in the specification, with the exception that in the AS-REQ the server must provide a sequence of certificates rooted in the Service Provider Root certificate rather than the MTA Root certificate. The precise requirements for these certificates are already provided in the security specification.
2. The PKINIT exchange should request a ticket for a service named "createkey" running on the KDC[10]. The principal identifier for this service is:

   `createkey/<KDC FQDN>@REALM`

3. Once it has obtained a ticket for the createkey service, the server then submits an AP-REQ to the KDC. This is exactly as described in section 6.5.2 of the security specification, with the following changes:
   - The DOI is (arbitrarily) set to 250 (decimal)
   - The Application Specific Data is an ASN.1 encoded value defined as:

```
CREATE_KEY_APPLICATION_SPECIFIC_DATA =
      SEQUENCE { duration [0] INTEGER OPTIONAL – duration in seconds
   }
```

   where the duration is the requested life of the service key in seconds.

Only the following ciphersuite is recognized:

Authentication Algorithm must have the value 0x00, corresponding to no authentication

Encryption Transform ID must have the value 0x01, corresponding to 3DES (because only 3DES service keys are supported by PacketCable)

4. The AP-REP is identical to the AP-REP already described in the security specification, with the following changes:

- The DOI is (arbitrarily) set to 250 (decimal)
- The kerberos subkey field contains a 24-octet 3DES session key, `ks`.

[10] In order to meet the requirement of Perfect Forward Secrecy, tickets for the createkey service can be obtained only via an AS-REQ. They cannot be obtained via a TGS-REQ.

- The Application Specific Data contains an ASN.1 OCTET STRING. The contents of the OCTET STRING are encoded object, padded as necessary to a multiple of eight octets, and encrypted by 3DES CBC encryption with a null IV and using the key `ks`.
- Once decrypted, the OCTET STRING in the Application Specific Data contains a SERVICE_KEY object:

```
SERVICE_KEY ::= SEQUENCE {
 PrincipalIdentifier, -- principal identifier associated with the key
 INTEGER,      -- key version number
 OCTET STRING,  -- key value
 UTCTime OPTIONAL  -- key expiration
 }
```

The `PrincipalIdentifier` type is defined as:

```
PrincipalIdentifier ::= SEQUENCE {
      PrincipalName,
      Realm
}
```

The contents of the key value OCTET STRING are the desired 24-octet 3DES service key.

21 IPv6 Support

Officially (*i.e*, according to the published specifications) PacketCable 1.x does not support IPv6. However, some operators are experimenting with IPv6 networks, and some MTA vendors are producing experimental equipment that includes support for IPv6.

The IPfonix, Inc. KDC includes support for IPv6[11]. Under IPv6, operation is almost identical to the published PacketCable specifications for IPv6, with just a few minor alterations, as follows:

1. Under normal circumstances, the MTA must place its IPv4 address in the AS-REQ. When the KDC is operated with IPv6 support, the MTA can instead use an IPv6 address in this field.
2. During the MTA FQDN messaging, the Provisioning Server normally must include the IPv4 address of the MTA in the MTA FQDN reply. When operating with IPv6 support, the Provisioning server may return either an IPv4 address (length 4 octets) as usual, or it may return any number of IPv4 and IPv6 addresses in the format specified by PacketCable 2.0.

There is no way to disable IPv4 support on the KDC. When IPv6 is enabled, the KDC operates in a dual-stack mode, supporting both IPv4 (as usual) and IPv6.

[11] Linux and Windows only

22 Troubleshooting

This section should help with the most common issues related to using the KDC software. If your question is not addressed here, please *carefully read this User Guide in its entirety*. Almost all the e-mails we receive asking for help could have been avoided by reading this document. We are always happy to provide help, but it is usually faster for you, the customer, to read the relevant section of this document and find the answer yourself than it is to contact us with copies of your `kdc.ini` file and relevant log files, and then wait while we examine those to determine the cause of the problem. The vast majority of the problems we see are caused by misconfiguration, so please look at the KDC log file carefully, since that usually contains a description of any configuration errors that the KDC can detect.

22.1 Errors during startup

The KDC performs a large number of internal consistency checks during start-up. Any detected inconsistency will result in an error message being sent to the console and to the log, after which the KDC will close itself down.

Below are listed the most common reasons we see for the KDC refusing to start. This is, however, by no means an exhaustive list. Many other errors in configuration are possible. The error message on the console and in the log file should be sufficiently clear to enable you to determine exactly what the problem is. If you are unable to interpret the error message, please e-mail it to us and we will explain exactly what it means.

22.1.1 The KDC says that there is an “unexpected number of chains” in a certificate directory.

The KDC expects to see exactly two chains of certificates in each directory containing certificates. One chain consists of a complete service provider hierarchy (certificates for: the service provider root, the service provider, an optional local system, and the KDC). The second chain consists of a single certificate: the root certificate for the client device. If the KDC finds more (or fewer) certificates than these in the directory, it will complain that there is an unexpected number of chains, and then abort. The solution, then, is to provide exactly the required certificates: no more, and no less. (With fewer certificates than the required number, the KDC would be unable to generate the correct chains needed to operate correctly; with more certificates than the required number, the KDC may be unable to determine which certificates the operator wishes to use.)

22.2 Errors during operation

The KDC is very careful to perform all the checks required by the security specification, in addition to common-sense checks (such as signature verifications) that are technically not required by the security specification. As a result, there is a large number of errors that may be triggered when a KDC processes incoming messaging. Most of these cause a KRB-ERR message to be sent to the client. The KRB-ERR message should contain a reasonable textual description of the error; a more detailed explanation should be written to the logfile. We here describe errors that may not be self-evident.

22.2.1 The KDC processes an AS-REQ and produces an AS-REP, but the MTA will not accept it.

Almost certainly, this is caused by a certificate mismatch. The AS-REP contains several certificates that are rooted in the Service Provider Root certificate that is provisioned on to the KDC. The MTA verifies the hierarchy by checking that they are rooted in the Service Provider Root certificate that is on the MTA (typically, this certificate is placed into the MTA firmware during manufacture). Consequently, if the Service Provider Root certificate on the MTA is not identical to the one on the KDC, a mismatch occurs and the MTA rejects the response.

Unfortunately, MTAs are often not provided with a good debugging interface; sometimes an MTA will explain that there is a certificate mismatch; more likely it will complain vaguely about an "incorrect certificate found in reply", or, more misleadingly, "certificate missing". Often an MTA has no debugging interface at all, and there is no externally-accessible method to determine why the MTA has rejected the response: all the user sees is that the MTA retries a few times and eventually gives up or reboots, just to cycle through the sequence again.

If the KDC produces an AS-REP, you should have a very high degree of confidence that the KDC is happy; there are no known ways for a KDC to produce an invalid AS-REP (assuming, of course, that one is not creating a message that is invalid by virtue of settings in the `[violations]` section of the `kdc.ini` file).

So the solution for this problem is to check that the Service Provider Root certificate on the MTA is identical to the one on the KDC. How one does this varies from MTA to MTA, and often can only be done by directly contacting the MTA manufacturer and asking for a copy of the Service Provider Root certificate(s) supported by the MTA.