BIND 9 Administrator Reference Manual

Release 9.16.28

Internet Systems Consortium

2022-04-11

CONTENTS

INTRODUCTION

The Internet Domain Name System (DNS) consists of the syntax to specify the names of entities in the Internet in a hierarchical manner, the rules used for delegating authority over names, and the system implementation that actually maps names to Internet addresses. DNS data is maintained in a group of distributed hierarchical databases.

1.1 Scope of Document

The Berkeley Internet Name Domain (BIND) implements a domain name server for a number of operating systems. This document provides basic information about the installation and care of the Internet Systems Consortium (ISC) BIND version 9 software package for system administrators.

This manual covers BIND version 9.16.28.

1.2 Organization of This Document

In this document, *Chapter 1* introduces the basic DNS and BIND concepts. *Chapter 2* describes resource requirements for running BIND in various environments. Information in *Chapter 3* is *task-oriented* in its presentation and is organized functionally, to aid in the process of installing the BIND 9 software. The task-oriented section is followed by *Chapter 4*, which is organized as a reference manual to aid in the ongoing maintenance of the software. *Chapter 5* contains more advanced concepts that the system administrator may need for implementing certain options. *Chapter 6* addresses security considerations, and *Chapter 7* contains troubleshooting help. The main body of the document is followed by several *appendices* which contain useful reference information, such as a *bibliography* and historic information related to BIND and the Domain Name System.

1.3 Conventions Used in This Document

In this document, we generally use `Fixed Width` text to indicate the following types of information:

- pathnames
- filenames
- URLs
- hostnames
- mailing list names
- new terms or concepts
- literal user input

- program output
- keywords
- variables

Text in "quotes," **bold**, or *italics* is also used for emphasis or clarity.

1.4 The Domain Name System (DNS)

This document explains the installation and upkeep of the BIND (Berkeley Internet Name Domain) software package. We begin by reviewing the fundamentals of the Domain Name System (DNS) as they relate to BIND.

1.4.1 DNS Fundamentals

The Domain Name System (DNS) is a hierarchical, distributed database. It stores information for mapping Internet host names to IP addresses and vice versa, mail routing information, and other data used by Internet applications.

Clients look up information in the DNS by calling a *resolver* library, which sends queries to one or more *name servers* and interprets the responses. The BIND 9 software distribution contains a name server, `named`, and a set of associated tools.

1.4.2 Domains and Domain Names

The data stored in the DNS is identified by *domain names* that are organized as a tree according to organizational or administrative boundaries. Each node of the tree, called a *domain*, is given a label. The domain name of the node is the concatenation of all the labels on the path from the node to the *root* node. This is represented in written form as a string of labels listed from right to left and separated by dots. A label need only be unique within its parent domain.

For example, a domain name for a host at the company *Example, Inc.* could be `ourhost.example.com`, where `com` is the top-level domain to which `ourhost.example.com` belongs, `example` is a subdomain of `com`, and `ourhost` is the name of the host.

For administrative purposes, the name space is partitioned into areas called *zones*, each starting at a node and extending down to the "leaf" nodes or to nodes where other zones start. The data for each zone is stored in a *name server*, which answers queries about the zone using the *DNS protocol*.

The data associated with each domain name is stored in the form of *resource records* (RRs). Some of the supported resource record types are described in *Types of Resource Records and When to Use Them*.

For more detailed information about the design of the DNS and the DNS protocol, please refer to the standards documents listed in *Requests for Comment (RFCs)*.

1.4.3 Zones

To properly operate a name server, it is important to understand the difference between a *zone* and a *domain*.

As stated previously, a zone is a point of delegation in the DNS tree. A zone consists of those contiguous parts of the domain tree for which a name server has complete information and over which it has authority. It contains all domain names from a certain point downward in the domain tree except those which are delegated to other zones. A delegation point is marked by one or more *NS records* in the parent zone, which should be matched by equivalent NS records at the root of the delegated zone.

For instance, consider the `example.com` domain, which includes names such as `host.aaa.example.com` and `host.bbb.example.com`, even though the `example.com` zone includes only delegations for the `aaa.example.com` and `bbb.example.com` zones. A zone can map exactly to a single domain, but could also include only part of a domain, the rest of which could be delegated to other name servers. Every name in the DNS tree is a *domain*, even if it is *terminal*, that is, has no *subdomains*. Every subdomain is a domain and every domain except the root is also a subdomain. The terminology is not intuitive and we suggest reading **RFC 1033**, **RFC 1034**, and **RFC 1035** to gain a complete understanding of this difficult and subtle topic.

Though BIND 9 is called a "domain name server," it deals primarily in terms of zones. The `primary` and `secondary` declarations in the `named.conf` file specify zones, not domains. When BIND asks some other site if it is willing to be a secondary server for a *domain*, it is actually asking for secondary service for some collection of *zones*.

1.4.4 Authoritative Name Servers

Each zone is served by at least one *authoritative name server*, which contains the complete data for the zone. To make the DNS tolerant of server and network failures, most zones have two or more authoritative servers, on different networks.

Responses from authoritative servers have the "authoritative answer" (AA) bit set in the response packets. This makes them easy to identify when debugging DNS configurations using tools like `dig` (*Diagnostic Tools*).

The Primary Server

The authoritative server, where the main copy of the zone data is maintained, is called the *primary* (formerly *master*) server, or simply the *primary*. Typically it loads the zone contents from some local file edited by humans or perhaps generated mechanically from some other local file which is edited by humans. This file is called the *zone file* or *master file*.

In some cases, however, the master file may not be edited by humans at all, but may instead be the result of *dynamic update* operations.

Secondary Servers

The other authoritative servers, the *secondary* servers (formerly known as *slave* servers) load the zone contents from another server using a replication process known as a *zone transfer*. Typically the data is transferred directly from the primary, but it is also possible to transfer it from another secondary. In other words, a secondary server may itself act as a primary to a subordinate secondary server.

Periodically, the secondary server must send a refresh query to determine whether the zone contents have been updated. This is done by sending a query for the zone's Start of Authority (SOA) record and checking whether the SERIAL field has been updated; if so, a new transfer request is initiated. The timing of these refresh queries is controlled by the SOA REFRESH and RETRY fields, but can be overridden with the `max-refresh-time`, `min-refresh-time`, `max-retry-time`, and `min-retry-time` options.

If the zone data cannot be updated within the time specified by the SOA EXPIRE option (up to a hard-coded maximum of 24 weeks), the secondary zone expires and no longer responds to queries.

Stealth Servers

Usually, all of the zone's authoritative servers are listed in NS records in the parent zone. These NS records constitute a *delegation* of the zone from the parent. The authoritative servers are also listed in the zone file itself, at the *top level* or *apex* of the zone. Servers that are not in the parent's NS delegation can be listed in the zone's top-level NS records, but servers that are not present at the zone's top level cannot be listed in the parent's delegation.

A *stealth server* is a server that is authoritative for a zone but is not listed in that zone's NS records. Stealth servers can be used for keeping a local copy of a zone, to speed up access to the zone's records or to make sure that the zone is available even if all the "official" servers for the zone are inaccessible.

A configuration where the primary server itself is a stealth server is often referred to as a "hidden primary" configuration. One use for this configuration is when the primary is behind a firewall and is therefore unable to communicate directly with the outside world.

1.4.5 Caching Name Servers

The resolver libraries provided by most operating systems are *stub resolvers*, meaning that they are not capable of performing the full DNS resolution process by themselves by talking directly to the authoritative servers. Instead, they rely on a local name server to perform the resolution on their behalf. Such a server is called a *recursive* name server; it performs *recursive lookups* for local clients.

To improve performance, recursive servers cache the results of the lookups they perform. Since the processes of recursion and caching are intimately connected, the terms *recursive server* and *caching server* are often used synonymously.

The length of time for which a record may be retained in the cache of a caching name server is controlled by the Time-To-Live (TTL) field associated with each resource record.

Forwarding

Even a caching name server does not necessarily perform the complete recursive lookup itself. Instead, it can *forward* some or all of the queries that it cannot satisfy from its cache to another caching name server, commonly referred to as a *forwarder*.

Forwarders are typically used when an administrator does not wish for all the servers at a given site to interact directly with the rest of the Internet. For example, a common scenario is when multiple internal DNS servers are behind an Internet firewall. Servers behind the firewall forward their requests to the server with external access, which queries Internet DNS servers on the internal servers' behalf.

Another scenario (largely now superseded by Response Policy Zones) is to send queries first to a custom server for RBL processing before forwarding them to the wider Internet.

There may be one or more forwarders in a given setup. The order in which the forwarders are listed in `named.conf` does not determine the sequence in which they are queried; rather, `named` uses the response times from previous queries to select the server that is likely to respond the most quickly. A server that has not yet been queried is given an initial small random response time to ensure that it is tried at least once. Dynamic adjustment of the recorded response times ensures that all forwarders are queried, even those with slower response times. This permits changes in behavior based on server responsiveness.

1.4.6 Name Servers in Multiple Roles

The BIND name server can simultaneously act as a primary for some zones, a secondary for other zones, and as a caching (recursive) server for a set of local clients.

However, since the functions of authoritative name service and caching/recursive name service are logically separate, it is often advantageous to run them on separate server machines. A server that only provides authoritative name service (an *authoritative-only* server) can run with recursion disabled, improving reliability and security. A server that is not authoritative for any zones and only provides recursive service to local clients (a *caching-only* server) does not need to be reachable from the Internet at large and can be placed inside a firewall.

BIND RESOURCE REQUIREMENTS

2.1 Hardware Requirements

DNS hardware requirements have traditionally been quite modest. For many installations, servers that have been retired from active duty have performed admirably as DNS servers.

However, the DNSSEC features of BIND 9 may be quite CPU-intensive, so organizations that make heavy use of these features may wish to consider larger systems for these applications. BIND 9 is fully multithreaded, allowing full utilization of multiprocessor systems for installations that need it.

2.2 CPU Requirements

CPU requirements for BIND 9 range from i386-class machines, for serving static zones without caching, to enterprise-class machines to process many dynamic updates and DNSSEC-signed zones, serving many thousands of queries per second.

2.3 Memory Requirements

Server memory must be sufficient to hold both the cache and the zones loaded from disk. The `max-cache-size` option can limit the amount of memory used by the cache, at the expense of reducing cache hit rates and causing more DNS traffic. It is still good practice to have enough memory to load all zone and cache data into memory; unfortunately, the best way to determine this for a given installation is to watch the name server in operation. After a few weeks, the server process should reach a relatively stable size where entries are expiring from the cache as fast as they are being inserted.

2.4 Name Server-Intensive Environment Issues

For name server-intensive environments, there are two configurations that may be used. The first is one where clients and any second-level internal name servers query the main name server, which has enough memory to build a large cache; this approach minimizes the bandwidth used by external name lookups. The second alternative is to set up second-level internal name servers to make queries independently. In this configuration, none of the individual machines need to have as much memory or CPU power as in the first alternative, but this has the disadvantage of making many more external queries, as none of the name servers share their cached data.

2.5 Supported Platforms

Current support status of various platforms and BIND 9 versions can be found in the ISC Knowledgebase:

https://kb.isc.org/docs/supported-platforms

In general, this version of BIND will build and run on any POSIX-compliant system with a C11-compliant C compiler, BSD-style sockets with RFC-compliant IPv6 support, POSIX-compliant threads, and the *required libraries*.

The following C11 features are used in BIND 9:

- Atomic operations support from the compiler is needed, either in the form of builtin operations, C11 atomics, or the `Interlocked` family of functions on Windows.

- Thread Local Storage support from the compiler is needed, either in the form of C11 `_Thread_local`/`thread_local`, the `__thread` GCC extension, or the `__declspec(thread)` MSVC extension on Windows.

ISC regularly tests BIND on many operating systems and architectures, but lacks the resources to test all of them. Consequently, ISC is only able to offer support on a "best effort" basis for some.

2.5.1 Regularly tested platforms

As of Dec 2021, BIND 9.16 is fully supported and regularly tested on the following systems:

- Debian 9, 10, 11
- Ubuntu LTS 18.04, 20.04
- Fedora 35
- Red Hat Enterprise Linux / CentOS / Oracle Linux 7, 8
- FreeBSD 12.3, 13.0
- OpenBSD 7.0
- Alpine Linux 3.15

The amd64, i386, armhf and arm64 CPU architectures are all fully supported.

2.5.2 Best effort

The following are platforms on which BIND is known to build and run. ISC makes every effort to fix bugs on these platforms, but may be unable to do so quickly due to lack of hardware, less familiarity on the part of engineering staff, and other constraints. With the exception of Windows Server 2016, none of these are tested regularly by ISC.

- Windows Server 2012 R2, 2016 / x64
- Windows 10 / x64
- macOS 10.12+
- Solaris 11
- NetBSD
- Other Linux distributions still supported by their vendors, such as:
    - Ubuntu 20.10+
    - Gentoo

– Arch Linux

- OpenWRT/LEDE 17.01+

- Other CPU architectures (mips, mipsel, sparc, …)

2.5.3 Community maintained

These systems may not all have the required dependencies for building BIND easily available, although it will be possible in many cases to compile those directly from source. The community and interested parties may wish to help with maintenance, and we welcome patch contributions, although we cannot guarantee that we will accept them. All contributions will be assessed against the risk of adverse effect on officially supported platforms.

- Platforms past or close to their respective EOL dates, such as:

  – Ubuntu 14.04, 16.04 (Ubuntu ESM releases are not supported)

  – CentOS 6

  – Debian Jessie

  – FreeBSD 10.x, 11.x

2.6 Unsupported Platforms

These are platforms on which BIND 9.16 is known *not* to build or run:

- Platforms without at least OpenSSL 1.0.2

- Windows 10 / x86

- Windows Server 2012 and older

- Solaris 10 and older

- Platforms that don't support IPv6 Advanced Socket API (RFC 3542)

- Platforms that don't support atomic operations (via compiler or library)

- Linux without NPTL (Native POSIX Thread Library)

- Platforms on which `libuv` cannot be compiled

2.7 Building BIND 9

To build on a Unix or Linux system, use:

```
$ ./configure
$ make
```

Several environment variables affect compilation, and they can be set before running `configure`. The most significant ones are:

Variable	Description
CC	The C compiler to use. `configure` tries to figure out the right one for supported systems.
CFLAGS	The C compiler flags. Defaults to include -g and/or -O2 as supported by the compiler. Please include `-g` if CFLAGS needs to be set.
STD_CINCLUDES	System header file directories. Can be used to specify where add-on thread or IPv6 support is, for example. Defaults to empty string.
STD_CDEFINES	Any additional preprocessor symbols you want defined. Defaults to empty string. For a list of possible settings, see the file OPTIONS.
LDFLAGS	The linker flags. Defaults to an empty string.
BUILD_CC	Needed when cross-compiling: the native C compiler to use when building for the target system.
BUILD_CFLAGS	CFLAGS for the target system during cross-compiling.
BUILD_CPPFLAGS	CPPFLAGS for the target system during cross-compiling.
BUILD_LDFLAGS	LDFLAGS for the target system during cross-compiling.
BUILD_LIBS	LIBS for the target system during cross-compiling.

Additional environment variables affecting the build are listed at the end of the `configure` help text, which can be obtained by running the command:

```
$ ./configure --help
```

If you're planning on making changes to the BIND 9 source, you should run `make depend`. If you're using Emacs, you might find `make tags` helpful.

2.7.1 Required Libraries

To build BIND 9, the following packages must be installed:

- `libcrypto`, `libssl`

- `libuv`

- `perl`

- `pkg-config` / `pkgconfig` / `pkgconf`

BIND 9.16 requires `libuv` 1.x or higher. On older systems, an updated `libuv` package needs to be installed from sources such as EPEL, PPA, or other native sources. The other option is to build and install `libuv` from source.

OpenSSL 1.0.2e or newer is required. If the OpenSSL library is installed in a nonstandard location, specify the prefix using `--with-openssl=<PREFIX>` on the `configure` command line.

Portions of BIND that are written in Python, including `dnssec-keymgr`, `dnssec-coverage`, `dnssec-checkds`, and some of the system tests, require the `argparse`, `ply` and `distutils.core` modules to be available. `argparse` is a standard module as of Python 2.7 and Python 3.2. `ply` is available from https://pypi.python.org/pypi/ply. `distutils.core` is required for installation.

2.7.2 Optional Features

To see a full list of configuration options, run `configure --help`.

To build shared libraries, specify `--with-libtool` on the `configure` command line.

To support the HTTP statistics channel, the server must be linked with at least one of the following libraries: `libxml2` (http://xmlsoft.org) or `json-c` (https://github.com/json-c/json-c). If these are installed at a nonstandard location, then:

- for `libxml2`, specify the prefix using `--with-libxml2=/prefix`,

- for `json-c`, adjust `PKG_CONFIG_PATH`.

To support compression on the HTTP statistics channel, the server must be linked against `zlib` (https://zlib.net/). If this is installed in a nonstandard location, specify the prefix using `--with-zlib=/prefix`.

To support storing configuration data for runtime-added zones in an LMDB database, the server must be linked with `liblmdb` (https://github.com/LMDB/lmdb). If this is installed in a nonstandard location, specify the prefix using `--with-lmdb=/prefix`.

To support MaxMind GeoIP2 location-based ACLs, the server must be linked with `libmaxminddb` (https://maxmind.github.io/libmaxminddb/). This is turned on by default if the library is found; if the library is installed in a nonstandard location, specify the prefix using `--with-maxminddb=/prefix`. GeoIP2 support can be switched off with `--disable-geoip`.

For DNSTAP packet logging, `libfstrm` (https://github.com/farsightsec/fstrm) and `libprotobuf-c` (https://developers.google.com/protocol-buffers) must be installed, and BIND must be configured with `--enable-dnstap`.

To support internationalized domain names in `dig`, `libidn2` (https://www.gnu.org/software/libidn/#libidn2) must be installed. If the library is installed in a nonstandard location, specify the prefix using `--with-libidn2=/prefix` or adjust `PKG_CONFIG_PATH`.

For line editing in `nsupdate` and `nslookup`, either the `readline` (https://tiswww.case.edu/php/chet/readline/rltop.html) or the `libedit` library (https://www.thrysoee.dk/editline/) must be installed. If these are installed at a nonstandard location, adjust `PKG_CONFIG_PATH`. `readline` is used by default, and `libedit` can be explicitly requested using `--with-readline=libedit`.

Certain compiled-in constants and default settings can be decreased to values better suited to small machines, e.g. OpenWRT boxes, by specifying `--with-tuning=small` on the `configure` command line. This decreases memory usage by using smaller structures, but degrades performance.

On Linux, process capabilities are managed in user space using the `libcap` library (https://git.kernel.org/pub/scm/libs/libcap/libcap.git/), which can be installed on most Linux systems via the `libcap-dev` or `libcap-devel` package. Process capability support can also be disabled by configuring with `--disable-linux-caps`.

On some platforms it is necessary to explicitly request large file support to handle files bigger than 2GB. This can be done by using `--enable-largefile` on the `configure` command line.

Support for the "fixed" RRset-order option can be enabled or disabled by specifying `--enable-fixed-rrset` or `--disable-fixed-rrset` on the `configure` command line. By default, fixed RRset-order is disabled to reduce memory footprint.

The `--enable-querytrace` option causes `named` to log every step while processing every query. This option should only be enabled when debugging because is has a significant negative impact on query performance.

`make install` installs `named` and the various BIND 9 libraries. By default, installation is into /usr/local, but this can be changed with the `--prefix` option when running `configure`.

The option `--sysconfdir` can be specified to set the directory where configuration files such as `named.conf` go by default; `--localstatedir` can be used to set the default parent directory of `run/named.pid`. `--sysconfdir` defaults to `$prefix/etc` and `--localstatedir` defaults to `$prefix/var`.

2.7.3 macOS

Building on macOS assumes that the "Command Tools for Xcode" are installed. These can be downloaded from https://developer.apple.com/download/more/ or, if Xcode is already installed, simply run `xcode-select --install`. (Note that an Apple ID may be required to access the download page.)

NAME SERVER CONFIGURATION

In this chapter we provide some suggested configurations, along with guidelines for their use. We suggest reasonable values for certain option settings.

3.1 Sample Configurations

3.1.1 A Caching-only Name Server

The following sample configuration is appropriate for a caching-only name server for use by clients internal to a corporation. All queries from outside clients are refused using the `allow-query` option. The same effect can be achieved using suitable firewall rules.

```
// Two corporate subnets we wish to allow queries from.
acl corpnets { 192.168.4.0/24; 192.168.7.0/24; };
options {
    // Working directory
    directory "/etc/namedb";

    allow-query { corpnets; };
};
// Provide a reverse mapping for the loopback
// address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
    type primary;
    file "localhost.rev";
    notify no;
};
```

3.1.2 An Authoritative-only Name Server

This sample configuration is for an authoritative-only server that is the primary server for `example.com` and a secondary server for the subdomain `eng.example.com`.

```
options {
    // Working directory
    directory "/etc/namedb";
    // Do not allow access to cache
    allow-query-cache { none; };
    // This is the default
    allow-query { any; };
```

(continues on next page)

```
    // Do not provide recursive service
    recursion no;
};

// Provide a reverse mapping for the loopback
// address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
    type primary;
    file "localhost.rev";
    notify no;
};
// We are the primary server for example.com
zone "example.com" {
    type primary;
    file "example.com.db";
    // IP addresses of secondary servers allowed to
    // transfer example.com
    allow-transfer {
     192.168.4.14;
     192.168.5.53;
    };
};
// We are a secondary server for eng.example.com
zone "eng.example.com" {
    type secondary;
    file "eng.example.com.bk";
    // IP address of eng.example.com primary server
    masters { 192.168.4.12; };
};
```

3.2 Load Balancing

A primitive form of load balancing can be achieved in the DNS by using multiple records (such as multiple A records) for one name.

For example, assuming three HTTP servers with network addresses of 10.0.0.1, 10.0.0.2, and 10.0.0.3, a set of records such as the following means that clients will connect to each machine one-third of the time:

Name	TTL	CLASS	TYPE	Resource Record (RR) Data
www	600	IN	A	10.0.0.1
	600	IN	A	10.0.0.2
	600	IN	A	10.0.0.3

When a resolver queries for these records, BIND rotates them and responds to the query with the records in a different order. In the example above, clients randomly receive records in the order 1, 2, 3; 2, 3, 1; and 3, 1, 2. Most clients use the first record returned and discard the rest.

For more detail on ordering responses, check the `rrset-order` sub-statement in the `options` statement; see *RRset Ordering*.

3.3 Name Server Operations

3.3.1 Tools for Use With the Name Server Daemon

This section describes several indispensable diagnostic, administrative, and monitoring tools available to the system administrator for controlling and debugging the name server daemon.

Diagnostic Tools

The `dig`, `host`, and `nslookup` programs are all command-line tools for manually querying name servers. They differ in style and output format.

dig `dig` is the most versatile and complete of these lookup tools. It has two modes: simple interactive mode for a single query, and batch mode, which executes a query for each in a list of several query lines. All query options are accessible from the command line.

For more information and a list of available commands and options, see *dig - DNS lookup utility*.

host The `host` utility emphasizes simplicity and ease of use. By default, it converts between host names and Internet addresses, but its functionality can be extended with the use of options.

For more information and a list of available commands and options, see *host - DNS lookup utility*.

nslookup `nslookup` has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains, or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

Due to its arcane user interface and frequently inconsistent behavior, we do not recommend the use of `nslookup`. Use `dig` instead.

Administrative Tools

Administrative tools play an integral part in the management of a server.

named-checkconf The `named-checkconf` program checks the syntax of a `named.conf` file.

For more information and a list of available commands and options, see *named-checkconf - named configuration file syntax checking tool*.

named-checkzone The `named-checkzone` program checks a zone file for syntax and consistency.

For more information and a list of available commands and options, see *named-checkzone - zone file validation tool*.

named-compilezone This tool is similar to `named-checkzone` but it always dumps the zone content to a specified file (typically in a different format).

For more information and a list of available commands and options, see *named-compilezone - zone file converting tool*.

rndc The remote name daemon control (`rndc`) program allows the system administrator to control the operation of a name server.

See *rndc - name server control utility* for details of the available `rndc` commands.

`rndc` requires a configuration file, since all communication with the server is authenticated with digital signatures that rely on a shared secret, and there is no way to provide that secret other than with a configuration file. The default location for the `rndc` configuration file is `/etc/rndc.conf`, but an alternate location can be specified with the `-c` option. If the configuration file is not found, `rndc` also looks in `/etc/rndc.key` (or whatever

sysconfdir was defined when the BIND build was configured). The rndc.key file is generated by running rndc-confgen -a as described in *controls Statement Definition and Usage*.

The format of the configuration file is similar to that of named.conf, but is limited to only four statements: the options, key, server, and include statements. These statements are what associate the secret keys to the servers with which they are meant to be shared. The order of statements is not significant.

The options statement has three clauses: default-server, default-key, and default-port. default-server takes a host name or address argument and represents the server that is contacted if no -s option is provided on the command line. default-key takes the name of a key as its argument, as defined by a key statement. default-port specifies the port to which rndc should connect if no port is given on the command line or in a server statement.

The key statement defines a key to be used by rndc when authenticating with named. Its syntax is identical to the key statement in named.conf. The keyword key is followed by a key name, which must be a valid domain name, though it need not actually be hierarchical; thus, a string like rndc_key is a valid name. The key statement has two clauses: algorithm and secret. While the configuration parser accepts any string as the argument to algorithm, currently only the strings hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, and hmac-sha512 have any meaning. The secret is a Base64-encoded string as specified in **RFC 3548**.

The server statement associates a key defined using the key statement with a server. The keyword server is followed by a host name or address. The server statement has two clauses: key and port. The key clause specifies the name of the key to be used when communicating with this server, and the port clause can be used to specify the port rndc should connect to on the server.

A sample minimal configuration file is as follows:

```
key rndc_key {
      algorithm "hmac-sha256";
      secret
        "c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K";
};
options {
      default-server 127.0.0.1;
      default-key    rndc_key;
};
```

This file, if installed as /etc/rndc.conf, allows the command:

```
$ rndc reload
```

to connect to 127.0.0.1 port 953 and causes the name server to reload, if a name server on the local machine is running with the following controls statements:

```
controls {
    inet 127.0.0.1
        allow { localhost; } keys { rndc_key; };
};
```

and it has an identical key statement for rndc_key.

Running the rndc-confgen program conveniently creates an rndc.conf file, and also displays the corresponding controls statement needed to add to named.conf. Alternatively, it is possible to run rndc-confgen -a to set up an rndc.key file and not modify named.conf at all.

3.3.2 Signals

Certain Unix signals cause the name server to take specific actions, as described in the following table. These signals can be sent using the `kill` command.

SIGHUP	Causes the server to read `named.conf` and reload the database.
SIGTERM	Causes the server to clean up and exit.
SIGINT	Causes the server to clean up and exit.

3.4 Plugins

Plugins are a mechanism to extend the functionality of `named` using dynamically loadable libraries. By using plugins, core server functionality can be kept simple for the majority of users; more complex code implementing optional features need only be installed by users that need those features.

The plugin interface is a work in progress, and is expected to evolve as more plugins are added. Currently, only "query plugins" are supported; these modify the name server query logic. Other plugin types may be added in the future.

The only plugin currently included in BIND is `filter-aaaa.so`, which replaces the `filter-aaaa` feature that previously existed natively as part of `named`. The code for this feature has been removed from `named` and can no longer be configured using standard `named.conf` syntax, but linking in the `filter-aaaa.so` plugin provides identical functionality.

3.4.1 Configuring Plugins

A plugin is configured with the `plugin` statement in `named.conf`:

```
plugin query "library.so" {
    parameters
};
```

In this example, file `library.so` is the plugin library. `query` indicates that this is a query plugin.

Multiple `plugin` statements can be specified, to load different plugins or multiple instances of the same plugin.

`parameters` are passed as an opaque string to the plugin's initialization routine. Configuration syntax differs depending on the module.

3.4.2 Developing Plugins

Each plugin implements four functions:

- `plugin_register` to allocate memory, configure a plugin instance, and attach to hook points within `named`,
- `plugin_destroy` to tear down the plugin instance and free memory,
- `plugin_version` to check that the plugin is compatible with the current version of the plugin API,
- `plugin_check` to test syntactic correctness of the plugin parameters.

At various locations within the `named` source code, there are "hook points" at which a plugin may register itself. When a hook point is reached while `named` is running, it is checked to see whether any plugins have registered themselves there; if so, the associated "hook action" - a function within the plugin library - is called. Hook actions may examine the runtime state and make changes: for example, modifying the answers to be sent back to a client or forcing a query to be aborted. More details can be found in the file `lib/ns/include/ns/hooks.h`.

BIND 9 CONFIGURATION REFERENCE

4.1 Configuration File Elements

Following is a list of elements used throughout the BIND configuration file documentation:

acl_name The name of an `address_match_list` as defined by the `acl` statement.

address_match_list A list of one or more `ip_addr`, `ip_prefix`, `key_id`, or `acl_name` elements; see *Address Match Lists*.

remoteserver_list A named list of one or more `ip_addr` with optional `key_id` and/or `ip_port`. A remoteserver_list may include other `remoteserver_list`.

domain_name A quoted string which is used as a DNS name; for example. `my.test.domain`.

namelist A list of one or more `domain_name` elements.

dotted_decimal One to four integers valued 0 through 255 separated by dots (`.`), such as `123.45.67` or `89.123.45.67`.

ip4_addr An IPv4 address with exactly four elements in `dotted_decimal` notation.

ip6_addr An IPv6 address, such as `2001:db8::1234`. IPv6-scoped addresses that have ambiguity on their scope zones must be disambiguated by an appropriate zone ID with the percent character (`%`) as a delimiter. It is strongly recommended to use string zone names rather than numeric identifiers, to be robust against system configuration changes. However, since there is no standard mapping for such names and identifier values, only interface names as link identifiers are supported, assuming one-to-one mapping between interfaces and links. For example, a link-local address `fe80::1` on the link attached to the interface `ne0` can be specified as `fe80::1%ne0`. Note that on most systems link-local addresses always have ambiguity and need to be disambiguated.

ip_addr An `ip4_addr` or `ip6_addr`.

ip_dscp A `number` between 0 and 63, used to select a differentiated services code point (DSCP) value for use with outgoing traffic on operating systems that support DSCP.

ip_port An IP port `number`. The `number` is limited to 0 through 65535, with values below 1024 typically restricted to use by processes running as root. In some cases, an asterisk (`*`) character can be used as a placeholder to select a random high-numbered port.

ip_prefix An IP network specified as an `ip_addr`, followed by a slash (`/`) and then the number of bits in the netmask. Trailing zeros in an``ip_addr`` may be omitted. For example, `127/8` is the network `127.0.0.0``with netmask  ``255.0.0.0` and `1.2.3.0/28` is network `1.2.3.0` with netmask `255.255.255.240`. When specifying a prefix involving a IPv6-scoped address, the scope may be omitted. In that case, the prefix matches packets from any scope.

key_id A `domain_name` representing the name of a shared key, to be used for transaction security.

key_list A list of one or more `key_id`, separated by semicolons and ending with a semicolon.

number A non-negative 32-bit integer (i.e., a number between 0 and 4294967295, inclusive). Its acceptable value might be further limited by the context in which it is used.

fixedpoint A non-negative real number that can be specified to the nearest one-hundredth. Up to five digits can be specified before a decimal point, and up to two digits after, so the maximum value is 99999.99. Acceptable values might be further limited by the contexts in which they are used.

path_name A quoted string which is used as a pathname, such as `zones/master/my.test.domain`.

port_list A list of an `ip_port` or a port range. A port range is specified in the form of `range` followed by two `ip_port``s, ``port_low` and `port_high`, which represents port numbers from `port_low` through `port_high`, inclusive. `port_low` must not be larger than `port_high`. For example, `range 1024 65535` represents ports from 1024 through 65535. In either case an asterisk (`*`) character is not allowed as a valid `ip_port`.

size_spec A 64-bit unsigned integer, or the keywords `unlimited` or `default`. Integers may take values 0 <= value <= 18446744073709551615, though certain parameters (such as `max-journal-size`) may use a more limited range within these extremes. In most cases, setting a value to 0 does not literally mean zero; it means "undefined" or "as big as possible," depending on the context. See the explanations of particular parameters that use `size_spec` for details on how they interpret its use. Numeric values can optionally be followed by a scaling factor: `K` or `k` for kilobytes, `M` or `m` for megabytes, and `G` or `g` for gigabytes, which scale by 1024, 1024*1024, and 1024*1024*1024 respectively. `unlimited` generally means "as big as possible," and is usually the best way to safely set a very large number. `default` uses the limit that was in force when the server was started.

size_or_percent A `size_spec` or integer value followed by `%` to represent percent. The behavior is exactly the same as `size_spec`, but `size_or_percent` also allows specifying a positive integer value followed by the `%` sign to represent percent.

yes_or_no Either `yes` or `no`. The words `true` and `false` are also accepted, as are the numbers `1` and `0`.

dialup_option One of `yes`, `no`, `notify`, `notify-passive`, `refresh`, or `passive`. When used in a zone, `notify-passive`, `refresh`, and `passive` are restricted to secondary and stub zones.

4.1.1 Address Match Lists

Syntax

```
address_match_list = address_match_list_element ; ...

address_match_list_element = [ ! ] ( ip_address | ip_prefix |
    key key_id | acl_name | { address_match_list } )
```

Definition and Usage

Address match lists are primarily used to determine access control for various server operations. They are also used in the `listen-on` and `sortlist` statements. The elements which constitute an address match list can be any of the following:

- an IP address (IPv4 or IPv6)

- an IP prefix (in / notation)

- a key ID, as defined by the `key` statement

- the name of an address match list defined with the `acl` statement

- a nested address match list enclosed in braces

Elements can be negated with a leading exclamation mark (`!`), and the match list names "any", "none", "localhost", and "localnets" are predefined. More information on those names can be found in the description of the `acl` statement.

The addition of the key clause made the name of this syntactic element something of a misnomer, since security keys can be used to validate access without regard to a host or network address. Nonetheless, the term "address match list" is still used throughout the documentation.

When a given IP address or prefix is compared to an address match list, the comparison takes place in approximately O(1) time. However, key comparisons require that the list of keys be traversed until a matching key is found, and therefore may be somewhat slower.

The interpretation of a match depends on whether the list is being used for access control, defining `listen-on` ports, or in a `sortlist`, and whether the element was negated.

When used as an access control list, a non-negated match allows access and a negated match denies access. If there is no match, access is denied. The clauses `allow-notify`, `allow-recursion`, `allow-recursion-on`, `allow-query`, `allow-query-on`, `allow-query-cache`, `allow-query-cache-on`, `allow-transfer`, `allow-update`, `allow-update-forwarding`, `blackhole`, and `keep-response-order` all use address match lists. Similarly, the `listen-on` option causes the server to refuse queries on any of the machine's addresses which do not match the list.

Order of insertion is significant. If more than one element in an ACL is found to match a given IP address or prefix, preference is given to the one that came *first* in the ACL definition. Because of this first-match behavior, an element that defines a subset of another element in the list should come before the broader element, regardless of whether either is negated. For example, in `1.2.3/24; ! 1.2.3.13;` the 1.2.3.13 element is completely useless because the algorithm matches any lookup for 1.2.3.13 to the 1.2.3/24 element. Using `! 1.2.3.13; 1.2.3/24` fixes that problem by blocking 1.2.3.13 via the negation, but all other 1.2.3.* hosts pass through.

4.1.2 Comment Syntax

The BIND 9 comment syntax allows comments to appear anywhere that whitespace may appear in a BIND configuration file. To appeal to programmers of all kinds, they can be written in the C, C++, or shell/perl style.

Syntax

```
/* This is a BIND comment as in C */
```

```
// This is a BIND comment as in C++
```

```
# This is a BIND comment as in common Unix shells
# and perl
```

Definition and Usage

Comments may appear anywhere that whitespace may appear in a BIND configuration file.

C-style comments start with the two characters /* (slash, star) and end with */ (star, slash). Because they are completely delimited with these characters, they can be used to comment only a portion of a line or to span multiple lines.

C-style comments cannot be nested. For example, the following is not valid because the entire comment ends with the first */:

```
/* This is the start of a comment.
   This is still part of the comment.
/* This is an incorrect attempt at nesting a comment. */
   This is no longer in any comment. */
```

C++-style comments start with the two characters // (slash, slash) and continue to the end of the physical line. They cannot be continued across multiple physical lines; to have one logical comment span multiple lines, each line must use the // pair. For example:

```
// This is the start of a comment.  The next line
// is a new comment, even though it is logically
// part of the previous comment.
```

Shell-style (or perl-style) comments start with the character # (number sign) and continue to the end of the physical line, as in C++ comments. For example:

```
# This is the start of a comment.  The next line
# is a new comment, even though it is logically
# part of the previous comment.
```

> **Warning:** The semicolon (`;`) character cannot start a comment, unlike in a zone file. The semicolon indicates the end of a configuration statement.

4.2 Configuration File Grammar

A BIND 9 configuration consists of statements and comments. Statements end with a semicolon; statements and comments are the only elements that can appear without enclosing braces. Many statements contain a block of sub-statements, which are also terminated with a semicolon.

The following statements are supported:

acl Defines a named IP address matching list, for access control and other uses.

controls Declares control channels to be used by the `rndc` utility.

dnssec-policy Describes a DNSSEC key and signing policy for zones. See *dnssec-policy Grammar* for details.

include Includes a file.

key Specifies key information for use in authentication and authorization using TSIG.

logging Specifies what information the server logs and where the log messages are sent.

masters Synonym for `primaries`.

options Controls global server configuration options and sets defaults for other statements.

parental-agents Defines a named list of servers for inclusion in primary and secondary zones' `parental-agents` lists.

primaries Defines a named list of servers for inclusion in stub and secondary zones' `primaries` or `also-notify` lists. (Note: this is a synonym for the original keyword `masters`, which can still be used, but is no longer the preferred terminology.)

server Sets certain configuration options on a per-server basis.

statistics-channels Declares communication channels to get access to `named` statistics.

trust-anchors Defines DNSSEC trust anchors: if used with the `initial-key` or `initial-ds` keyword, trust anchors are kept up-to-date using **RFC 5011** trust anchor maintenance; if used with `static-key` or `static-ds`, keys are permanent.

managed-keys Is identical to `trust-anchors`; this option is deprecated in favor of `trust-anchors` with the `initial-key` keyword, and may be removed in a future release.

trusted-keys Defines permanent trusted DNSSEC keys; this option is deprecated in favor of `trust-anchors` with the `static-key` keyword, and may be removed in a future release.

view Defines a view.

zone Defines a zone.

The `logging` and `options` statements may only occur once per configuration.

4.2.1 `acl` Statement Grammar

```
acl <string> { <address_match_element>; ... };
```

4.2.2 `acl` Statement Definition and Usage

The `acl` statement assigns a symbolic name to an address match list. It gets its name from one of the primary uses of address match lists: Access Control Lists (ACLs).

The following ACLs are built-in:

any Matches all hosts.

none Matches no hosts.

localhost Matches the IPv4 and IPv6 addresses of all network interfaces on the system. When addresses are added or removed, the `localhost` ACL element is updated to reflect the changes.

localnets Matches any host on an IPv4 or IPv6 network for which the system has an interface. When addresses are added or removed, the `localnets` ACL element is updated to reflect the changes. Some systems do not provide a way to determine the prefix lengths of local IPv6 addresses; in such cases, `localnets` only matches the local IPv6 addresses, just like `localhost`.

4.2.3 `controls` Statement Grammar

```
controls {
    inet ( <ipv4_address> | <ipv6_address> |
        * ) [ port ( <integer> | * ) ] allow
        { <address_match_element>; ... } [
        keys { <string>; ... } ] [ read-only
        <boolean> ];
    unix <quoted_string> perm <integer>
        owner <integer> group <integer> [
        keys { <string>; ... } ] [ read-only
        <boolean> ];
};
```

4.2.4 `controls` Statement Definition and Usage

The `controls` statement declares control channels to be used by system administrators to manage the operation of the name server. These control channels are used by the `rndc` utility to send commands to and retrieve non-DNS results from a name server.

An `inet` control channel is a TCP socket listening at the specified `ip_port` on the specified `ip_addr`, which can be an IPv4 or IPv6 address. An `ip_addr` of `*` (asterisk) is interpreted as the IPv4 wildcard address; connections are accepted on any of the system's IPv4 addresses. To listen on the IPv6 wildcard address, use an `ip_addr` of `::`. If `rndc` is only used on the local host, using the loopback address (`127.0.0.1` or `::1`) is recommended for maximum security.

If no port is specified, port 953 is used. The asterisk `*` cannot be used for `ip_port`.

The ability to issue commands over the control channel is restricted by the `allow` and `keys` clauses. Connections to the control channel are permitted based on the `address_match_list`. This is for simple IP address-based filtering only; any `key_id` elements of the `address_match_list` are ignored.

A `unix` control channel is a Unix domain socket listening at the specified path in the file system. Access to the socket is specified by the `perm`, `owner`, and `group` clauses. Note that on some platforms (SunOS and Solaris), the permissions (`perm`) are applied to the parent directory as the permissions on the socket itself are ignored.

The primary authorization mechanism of the command channel is the `key_list`, which contains a list of `key_id``s`. Each `` ``key_id`` in the `key_list` is authorized to execute commands over the control channel. See *Administrative Tools* for information about configuring keys in `rndc`.

If the `read-only` clause is enabled, the control channel is limited to the following set of read-only commands: `nta -dump`, `null`, `status`, `showzone`, `testgen`, and `zonestatus`. By default, `read-only` is not enabled and the control channel allows read-write access.

If no `controls` statement is present, `named` sets up a default control channel listening on the loopback address 127.0.0.1 and its IPv6 counterpart, ::1. In this case, and also when the `controls` statement is present but does not have a `keys` clause, `named` attempts to load the command channel key from the file `rndc.key` in `/etc` (or whatever `sysconfdir` was specified when BIND was built). To create an `rndc.key` file, run `rndc-confgen -a`.

To disable the command channel, use an empty `controls` statement: `controls { };`.

4.2.5 `include` Statement Grammar

```
include filename;
```

4.2.6 `include` Statement Definition and Usage

The `include` statement inserts the specified file (or files if a valid glob expression is detected) at the point where the `include` statement is encountered. The `include` statement facilitates the administration of configuration files by permitting the reading or writing of some things but not others. For example, the statement could include private keys that are readable only by the name server.

4.2.7 `key` Statement Grammar

```
key <string> {
     algorithm <string>;
     secret <string>;
};
```

4.2.8 `key` Statement Definition and Usage

The `key` statement defines a shared secret key for use with TSIG (see *TSIG*) or the command channel (see *controls Statement Definition and Usage*).

The `key` statement can occur at the top level of the configuration file or inside a `view` statement. Keys defined in top-level `key` statements can be used in all views. Keys intended for use in a `controls` statement (see *controls Statement Definition and Usage*) must be defined at the top level.

The `key_id`, also known as the key name, is a domain name that uniquely identifies the key. It can be used in a `server` statement to cause requests sent to that server to be signed with this key, or in address match lists to verify that incoming requests have been signed with a key matching this name, algorithm, and secret.

The `algorithm_id` is a string that specifies a security/authentication algorithm. The `named` server supports `hmac-md5`, `hmac-sha1`, `hmac-sha224`, `hmac-sha256`, `hmac-sha384`, and `hmac-sha512` TSIG authentication. Truncated hashes are supported by appending the minimum number of required bits preceded by a dash, e.g., `hmac-sha1-80`. The `secret_string` is the secret to be used by the algorithm, and is treated as a Base64-encoded string.

4.2.9 `logging` Statement Grammar

```
logging {
     category <string> { <string>; ... };
     channel <string> {
             buffered <boolean>;
             file <quoted_string> [ versions ( unlimited | <integer> ) ]
                 [ size <size> ] [ suffix ( increment | timestamp ) ];
             null;
             print-category <boolean>;
             print-severity <boolean>;
             print-time ( iso8601 | iso8601-utc | local | <boolean> );
             severity <log_severity>;
             stderr;
             syslog [ <syslog_facility> ];
     };
};
```

4.2.10 `logging` Statement Definition and Usage

The `logging` statement configures a wide variety of logging options for the name server. Its `channel` phrase associates output methods, format options, and severity levels with a name that can then be used with the `category` phrase to select how various classes of messages are logged.

Only one `logging` statement is used to define as many channels and categories as desired. If there is no `logging` statement, the logging configuration is:

```
logging {
    category default { default_syslog; default_debug; };
    category unmatched { null; };
};
```

If `named` is started with the `-L` option, it logs to the specified file at startup, instead of using syslog. In this case the logging configuration is:

```
logging {
    category default { default_logfile; default_debug; };
    category unmatched { null; };
};
```

The logging configuration is only established when the entire configuration file has been parsed. When the server starts up, all logging messages regarding syntax errors in the configuration file go to the default channels, or to standard error if the `-g` option was specified.

The `channel` Phrase

All log output goes to one or more `channels`; there is no limit to the number of channels that can be created.

Every channel definition must include a destination clause that says whether messages selected for the channel go to a file, go to a particular syslog facility, go to the standard error stream, or are discarded. The definition can optionally also limit the message severity level that is accepted by the channel (the default is `info`), and whether to include a `named`-generated time stamp, the category name, and/or the severity level (the default is not to include any).

The `null` destination clause causes all messages sent to the channel to be discarded; in that case, other options for the channel are meaningless.

The `file` destination clause directs the channel to a disk file. It can include additional arguments to specify how large the file is allowed to become before it is rolled to a backup file (`size`), how many backup versions of the file are saved each time this happens (`versions`), and the format to use for naming backup versions (`suffix`).

The `size` option is used to limit log file growth. If the file ever exceeds the specified size, then `named` stops writing to the file unless it has a `versions` option associated with it. If backup versions are kept, the files are rolled as described below. If there is no `versions` option, no more data is written to the log until some out-of-band mechanism removes or truncates the log to less than the maximum size. The default behavior is not to limit the size of the file.

File rolling only occurs when the file exceeds the size specified with the `size` option. No backup versions are kept by default; any existing log file is simply appended. The `versions` option specifies how many backup versions of the file should be kept. If set to `unlimited`, there is no limit.

The `suffix` option can be set to either `increment` or `timestamp`. If set to `timestamp`, then when a log file is rolled, it is saved with the current timestamp as a file suffix. If set to `increment`, then backup files are saved with incrementing numbers as suffixes; older files are renamed when rolling. For example, if `versions` is set to 3 and `suffix` to `increment`, then when `filename.log` reaches the size specified by `size`, `filename.log.1` is renamed to `filename.log.2`, `filename.log.0` is renamed to `filename.log.1`, and `filename.log` is renamed to `filename.log.0`, whereupon a new `filename.log` is opened.

Here is an example using the `size`, `versions`, and `suffix` options:

```
channel an_example_channel {
    file "example.log" versions 3 size 20m suffix increment;
    print-time yes;
    print-category yes;
};
```

The `syslog` destination clause directs the channel to the system log. Its argument is a syslog facility as described in the `syslog` man page. Known facilities are `kern`, `user`, `mail`, `daemon`, `auth`, `syslog`, `lpr`, `news`, `uucp`, `cron`, `authpriv`, `ftp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, and `local7`; however, not all facilities are supported on all operating systems. How `syslog` handles messages sent to this facility is described in the `syslog.conf` man page. On a system which uses a very old version of `syslog`, which only uses two arguments to the `openlog()` function, this clause is silently ignored.

On Windows machines, syslog messages are directed to the EventViewer.

The `severity` clause works like `syslog`'s "priorities," except that they can also be used when writing straight to a file rather than using `syslog`. Messages which are not at least of the severity level given are not selected for the channel; messages of higher severity levels are accepted.

When using `syslog`, the `syslog.conf` priorities also determine what eventually passes through. For example, defining a channel facility and severity as `daemon` and `debug`, but only logging `daemon.warning` via `syslog.conf`, causes messages of severity `info` and `notice` to be dropped. If the situation were reversed, with `named` writing messages of only `warning` or higher, then `syslogd` would print all messages it received from the channel.

The `stderr` destination clause directs the channel to the server's standard error stream. This is intended for use when the server is running as a foreground process, as when debugging a configuration, for example.

The server can supply extensive debugging information when it is in debugging mode. If the server's global debug level is greater than zero, debugging mode is active. The global debug level is set either by starting the `named` server with the `-d` flag followed by a positive integer, or by running `rndc trace`. The global debug level can be set to zero, and debugging mode turned off, by running `rndc notrace`. All debugging messages in the server have a debug level; higher debug levels give more detailed output. Channels that specify a specific debug severity, for example:

```
channel specific_debug_level {
    file "foo";
    severity debug 3;
};
```

get debugging output of level 3 or less any time the server is in debugging mode, regardless of the global debugging level. Channels with `dynamic` severity use the server's global debug level to determine what messages to print.

`print-time` can be set to `yes`, `no`, or a time format specifier, which may be one of `local`, `iso8601`, or `iso8601-utc`. If set to `no`, the date and time are not logged. If set to `yes` or `local`, the date and time are logged in a human-readable format, using the local time zone. If set to `iso8601`, the local time is logged in ISO 8601 format. If set to `iso8601-utc`, the date and time are logged in ISO 8601 format, with time zone set to UTC. The default is `no`.

`print-time` may be specified for a `syslog` channel, but it is usually pointless since `syslog` also logs the date and time.

If `print-category` is requested, then the category of the message is logged as well. Finally, if `print-severity` is on, then the severity level of the message is logged. The `print-` options may be used in any combination, and are always printed in the following order: time, category, severity. Here is an example where all three `print-` options are on:

```
28-Feb-2000 15:05:32.863 general: notice: running
```

If `buffered` has been turned on, the output to files is not flushed after each log entry. By default all log messages are flushed.

There are four predefined channels that are used for `named`'s default logging, as follows. If `named` is started with the `-L` option, then a fifth channel, `default_logfile`, is added. How they are used is described in *The category Phrase*.

```
channel default_syslog {
    // send to syslog's daemon facility
    syslog daemon;
    // only send priority info and higher
    severity info;
};

channel default_debug {
    // write to named.run in the working directory
    // Note: stderr is used instead of "named.run" if
    // the server is started with the '-g' option.
    file "named.run";
    // log at the server's current debug level
    severity dynamic;
};

channel default_stderr {
    // writes to stderr
    stderr;
    // only send priority info and higher
    severity info;
};

channel null {
  // toss anything sent to this channel
  null;
};

channel default_logfile {
    // this channel is only present if named is
    // started with the -L option, whose argument
    // provides the file name
    file "...";
    // log at the server's current debug level
    severity dynamic;
};
```

The `default_debug` channel has the special property that it only produces output when the server's debug level is non-zero. It normally writes to a file called `named.run` in the server's working directory.

For security reasons, when the `-u` command-line option is used, the `named.run` file is created only after `named` has changed to the new UID, and any debug output generated while `named` is starting - and still running as root - is discarded. To capture this output, run the server with the `-L` option to specify a default logfile, or the `-g` option to log to standard error which can be redirected to a file.

Once a channel is defined, it cannot be redefined. The built-in channels cannot be altered directly, but the default logging can be modified by pointing categories at defined channels.

The `category` Phrase

There are many categories, so desired logs can be sent anywhere while unwanted logs are ignored. If a list of channels is not specified for a category, log messages in that category are sent to the `default` category instead. If no default category is specified, the following "default default" is used:

```
category default { default_syslog; default_debug; };
```

If `named` is started with the `-L` option, the default category is:

```
category default { default_logfile; default_debug; };
```

As an example, let's say a user wants to log security events to a file, but also wants to keep the default logging behavior. They would specify the following:

```
channel my_security_channel {
    file "my_security_file";
    severity info;
};
category security {
    my_security_channel;
    default_syslog;
    default_debug;
};
```

To discard all messages in a category, specify the `null` channel:

```
category xfer-out { null; };
category notify { null; };
```

The following are the available categories and brief descriptions of the types of log information they contain. More categories may be added in future BIND releases.

`client` Processing of client requests.

`cname` Name servers that are skipped for being a CNAME rather than A/AAAA records.

`config` Configuration file parsing and processing.

`database` Messages relating to the databases used internally by the name server to store zone and cache data.

`default` Logging options for those categories where no specific configuration has been defined.

`delegation-only` Queries that have been forced to NXDOMAIN as the result of a delegation-only zone or a `delegation-only` in a forward, hint, or stub zone declaration.

`dispatch` Dispatching of incoming packets to the server modules where they are to be processed.

`dnssec` DNSSEC and TSIG protocol processing.

`dnstap` The "dnstap" DNS traffic capture system.

`edns-disabled` Log queries that have been forced to use plain DNS due to timeouts. This is often due to the remote servers not being **RFC 1034**-compliant (not always returning FORMERR or similar to EDNS queries and other extensions to the DNS when they are not understood). In other words, this is targeted at servers that fail to respond to DNS queries that they don't understand.

Note: the log message can also be due to packet loss. Before reporting servers for non-**RFC 1034** compliance they should be re-tested to determine the nature of the non-compliance. This testing should prevent or reduce the number of false-positive reports.

Note: eventually `named` will have to stop treating such timeouts as due to [RFC 1034](#) non-compliance and start treating it as plain packet loss. Falsely classifying packet loss as due to [RFC 1034](#) non-compliance impacts DNSSEC validation, which requires EDNS for the DNSSEC records to be returned.

general A catch-all for many things that still are not classified into categories.

lame-servers Misconfigurations in remote servers, discovered by BIND 9 when trying to query those servers during resolution.

network Network operations.

notify The NOTIFY protocol.

nsid NSID options received from upstream servers.

queries A location where queries should be logged.

At startup, specifying the category `queries` also enables query logging unless the `querylog` option has been specified.

The query log entry first reports a client object identifier in @0x<hexadecimal-number> format. Next, it reports the client's IP address and port number, and the query name, class, and type. Next, it reports whether the Recursion Desired flag was set (+ if set, - if not set), whether the query was signed (S), whether EDNS was in use along with the EDNS version number (E(#)), whether TCP was used (T), whether DO (DNSSEC Ok) was set (D), whether CD (Checking Disabled) was set (C), whether a valid DNS Server COOKIE was received (V), and whether a DNS COOKIE option without a valid Server COOKIE was present (K). After this, the destination address the query was sent to is reported. Finally, if any CLIENT-SUBNET option was present in the client query, it is included in square brackets in the format [ECS address/source/scope].

```
client 127.0.0.1#62536 (www.example.com): query:  www.example.com IN AAAA
+SE client ::1#62537 (www.example.net): query: www.example.net IN AAAA -SE
```

The first part of this log message, showing the client address/port number and query name, is repeated in all subsequent log messages related to the same query.

query-errors Information about queries that resulted in some failure.

rate-limit Start, periodic, and final notices of the rate limiting of a stream of responses that are logged at `info` severity in this category. These messages include a hash value of the domain name of the response and the name itself, except when there is insufficient memory to record the name for the final notice. The final notice is normally delayed until about one minute after rate limiting stops. A lack of memory can hurry the final notice, which is indicated by an initial asterisk (*). Various internal events are logged at debug level 1 and higher.

Rate limiting of individual requests is logged in the `query-errors` category.

resolver DNS resolution, such as the recursive lookups performed on behalf of clients by a caching name server.

rpz Information about errors in response policy zone files, rewritten responses, and, at the highest `debug` levels, mere rewriting attempts.

security Approval and denial of requests.

serve-stale Indication of whether a stale answer is used following a resolver failure.

spill Queries that have been terminated, either by dropping or responding with SERVFAIL, as a result of a fetchlimit quota being exceeded.

trust-anchor-telemetry Trust-anchor-telemetry requests received by `named`.

unmatched Messages that `named` was unable to determine the class of, or for which there was no matching `view`. A one-line summary is also logged to the `client` category. This category is best sent to a file or stderr; by default it is sent to the `null` channel.

update Dynamic updates.

update-security Approval and denial of update requests.

xfer-in Zone transfers the server is receiving.

xfer-out Zone transfers the server is sending.

zoneload Loading of zones and creation of automatic empty zones.

The `query-errors` Category

The `query-errors` category is used to indicate why and how specific queries resulted in responses which indicate an error. Normally, these messages are logged at `debug` logging levels; note, however, that if query logging is active, some are logged at `info`. The logging levels are described below:

At `debug` level 1 or higher - or at `info` when query logging is active - each response with the rcode of SERVFAIL is logged as follows:

```
client 127.0.0.1#61502: query failed (SERVFAIL) for www.example.com/IN/AAAA at
query.c:3880
```

This means an error resulting in SERVFAIL was detected at line 3880 of source file `query.c`. Log messages of this level are particularly helpful in identifying the cause of SERVFAIL for an authoritative server.

At `debug` level 2 or higher, detailed context information about recursive resolutions that resulted in SERVFAIL is logged. The log message looks like this:

```
fetch completed at resolver.c:2970 for www.example.com/A
in 10.000183: timed out/success [domain:example.com,
referral:2,restart:7,qrysent:8,timeout:5,lame:0,quota:0,neterr:0,
badresp:1,adberr:0,findfail:0,valfail:0]
```

The first part before the colon shows that a recursive resolution for AAAA records of www.example.com completed in 10.000183 seconds, and the final result that led to the SERVFAIL was determined at line 2970 of source file `resolver.c`.

The next part shows the detected final result and the latest result of DNSSEC validation. The latter is always "success" when no validation attempt was made. In this example, this query probably resulted in SERVFAIL because all name servers are down or unreachable, leading to a timeout in 10 seconds. DNSSEC validation was probably not attempted.

The last part, enclosed in square brackets, shows statistics collected for this particular resolution attempt. The `domain` field shows the deepest zone that the resolver reached; it is the zone where the error was finally detected. The meaning of the other fields is summarized in the following list.

referral The number of referrals the resolver received throughout the resolution process. In the above `example.com` there are two.

restart The number of cycles that the resolver tried remote servers at the `domain` zone. In each cycle, the resolver sends one query (possibly resending it, depending on the response) to each known name server of the `domain` zone.

qrysent The number of queries the resolver sent at the `domain` zone.

timeout The number of timeouts the resolver received since the last response.

lame The number of lame servers the resolver detected at the `domain` zone. A server is detected to be lame either by an invalid response or as a result of lookup in BIND 9's address database (ADB), where lame servers are cached.

quota The number of times the resolver was unable to send a query because it had exceeded the permissible fetch quota for a server.

neterr The number of erroneous results that the resolver encountered in sending queries at the `domain` zone. One common case is when the remote server is unreachable and the resolver receives an "ICMP unreachable" error message.

badresp The number of unexpected responses (other than `lame`) to queries sent by the resolver at the `domain` zone.

adberr Failures in finding remote server addresses of the``domain`` zone in the ADB. One common case of this is that the remote server's name does not have any address records.

findfail Failures to resolve remote server addresses. This is a total number of failures throughout the resolution process.

valfail Failures of DNSSEC validation. Validation failures are counted throughout the resolution process (not limited to the `domain` zone), but should only happen in `domain`.

At `debug` level 3 or higher, the same messages as those at `debug` level 1 are logged for errors other than SERVFAIL. Note that negative responses such as NXDOMAIN are not errors, and are not logged at this debug level.

At `debug` level 4 or higher, the detailed context information logged at `debug` level 2 is logged for errors other than SERVFAIL and for negative responses such as NXDOMAIN.

4.2.11 `parental-agents` Statement Grammar

```
parental-agents <string> [ port <integer> ] [
    dscp <integer> ] { ( <remote-servers> |
    <ipv4_address> [ port <integer> ] |
    <ipv6_address> [ port <integer> ] ) [ key
    <string> ]; ... };
```

4.2.12 `parental-agents` Statement Definition and Usage

`parental-agents` lists allow for a common set of parental agents to be easily used by multiple primary and secondary zones in their `parental-agents` lists. A parental agent is the entity that the zone has a relationship with to change its delegation information (defined in **RFC 7344**).

4.2.13 `primaries` Statement Grammar

```
primaries <string> [ port <integer> ] [ dscp
    <integer> ] { ( <remote-servers> |
    <ipv4_address> [ port <integer> ] |
    <ipv6_address> [ port <integer> ] ) [ key
    <string> ]; ... };
```

4.2.14 `primaries` Statement Definition and Usage

`primaries` lists allow for a common set of primary servers to be easily used by multiple stub and secondary zones in their `primaries` or `also-notify` lists. (Note: `primaries` is a synonym for the original keyword `masters`, which can still be used, but is no longer the preferred terminology.)

4.2.15 `options` Statement Grammar

This is the grammar of the `options` statement in the `named.conf` file:

```
options {
     allow-new-zones <boolean>;
     allow-notify { <address_match_element>; ... };
     allow-query { <address_match_element>; ... };
     allow-query-cache { <address_match_element>; ... };
     allow-query-cache-on { <address_match_element>; ... };
     allow-query-on { <address_match_element>; ... };
     allow-recursion { <address_match_element>; ... };
     allow-recursion-on { <address_match_element>; ... };
     allow-transfer { <address_match_element>; ... };
     allow-update { <address_match_element>; ... };
     allow-update-forwarding { <address_match_element>; ... };
     also-notify [ port <integer> ] [ dscp <integer> ] { (
         <remote-servers> | <ipv4_address> [ port <integer> ] |
         <ipv6_address> [ port <integer> ] ) [ key <string> ]; ... };
     alt-transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * )
         ] [ dscp <integer> ];
     alt-transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> |
         * ) ] [ dscp <integer> ];
     answer-cookie <boolean>;
     attach-cache <string>;
     auth-nxdomain <boolean>; // default changed
     auto-dnssec ( allow | maintain | off );
     automatic-interface-scan <boolean>;
     avoid-v4-udp-ports { <portrange>; ... };
     avoid-v6-udp-ports { <portrange>; ... };
     bindkeys-file <quoted_string>;
     blackhole { <address_match_element>; ... };
     cache-file <quoted_string>; // deprecated
     catalog-zones { zone <string> [ default-masters [ port <integer> ]
         [ dscp <integer> ] { ( <remote-servers> | <ipv4_address> [ port
         <integer> ] | <ipv6_address> [ port <integer> ] ) [ key
         <string> ]; ... } ] [ zone-directory <quoted_string> ] [
         in-memory <boolean> ] [ min-update-interval <duration> ]; ... };
     check-dup-records ( fail | warn | ignore );
     check-integrity <boolean>;
     check-mx ( fail | warn | ignore );
     check-mx-cname ( fail | warn | ignore );
     check-names ( primary | master |
         secondary | slave | response ) (
         fail | warn | ignore );
     check-sibling <boolean>;
     check-spf ( warn | ignore );
     check-srv-cname ( fail | warn | ignore );
     check-wildcard <boolean>;
     clients-per-query <integer>;
     cookie-algorithm ( aes | siphash24 );
     cookie-secret <string>;
     coresize ( default | unlimited | <sizeval> );
     datasize ( default | unlimited | <sizeval> );
     deny-answer-addresses { <address_match_element>; ... } [
         except-from { <string>; ... } ];
     deny-answer-aliases { <string>; ... } [ except-from { <string>; ...
         } ];
```

```
      dialup ( notify | notify-passive | passive | refresh | <boolean> );
      directory <quoted_string>;
      disable-algorithms <string> { <string>;
          ... };
      disable-ds-digests <string> { <string>;
          ... };
      disable-empty-zone <string>;
      dns64 <netprefix> {
              break-dnssec <boolean>;
              clients { <address_match_element>; ... };
              exclude { <address_match_element>; ... };
              mapped { <address_match_element>; ... };
              recursive-only <boolean>;
              suffix <ipv6_address>;
      };
      dns64-contact <string>;
      dns64-server <string>;
      dnskey-sig-validity <integer>;
      dnsrps-enable <boolean>;
      dnsrps-options { <unspecified-text> };
      dnssec-accept-expired <boolean>;
      dnssec-dnskey-kskonly <boolean>;
      dnssec-loadkeys-interval <integer>;
      dnssec-must-be-secure <string> <boolean>;
      dnssec-policy <string>;
      dnssec-secure-to-insecure <boolean>;
      dnssec-update-mode ( maintain | no-resign );
      dnssec-validation ( yes | no | auto );
      dnstap { ( all | auth | client | forwarder | resolver | update ) [
          ( query | response ) ]; ... };
      dnstap-identity ( <quoted_string> | none | hostname );
      dnstap-output ( file | unix ) <quoted_string> [ size ( unlimited |
          <size> ) ] [ versions ( unlimited | <integer> ) ] [ suffix (
          increment | timestamp ) ];
      dnstap-version ( <quoted_string> | none );
      dscp <integer>;
      dual-stack-servers [ port <integer> ] { ( <quoted_string> [ port
          <integer> ] [ dscp <integer> ] | <ipv4_address> [ port
          <integer> ] [ dscp <integer> ] | <ipv6_address> [ port
          <integer> ] [ dscp <integer> ] ); ... };
      dump-file <quoted_string>;
      edns-udp-size <integer>;
      empty-contact <string>;
      empty-server <string>;
      empty-zones-enable <boolean>;
      fetch-quota-params <integer> <fixedpoint> <fixedpoint> <fixedpoint>;
      fetches-per-server <integer> [ ( drop | fail ) ];
      fetches-per-zone <integer> [ ( drop | fail ) ];
      files ( default | unlimited | <sizeval> );
      flush-zones-on-shutdown <boolean>;
      forward ( first | only );
      forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address>
          | <ipv6_address> ) [ port <integer> ] [ dscp <integer> ]; ... };
      fstrm-set-buffer-hint <integer>;
      fstrm-set-flush-timeout <integer>;
      fstrm-set-input-queue-size <integer>;
      fstrm-set-output-notify-threshold <integer>;
```

```
fstrm-set-output-queue-model ( mpsc | spsc );
fstrm-set-output-queue-size <integer>;
fstrm-set-reopen-interval <duration>;
geoip-directory ( <quoted_string> | none );
glue-cache <boolean>;
heartbeat-interval <integer>;
hostname ( <quoted_string> | none );
interface-interval <duration>;
ixfr-from-differences ( primary | master | secondary | slave |
    <boolean> );
keep-response-order { <address_match_element>; ... };
key-directory <quoted_string>;
lame-ttl <duration>;
listen-on [ port <integer> ] [ dscp
    <integer> ] {
    <address_match_element>; ... };
listen-on-v6 [ port <integer> ] [ dscp
    <integer> ] {
    <address_match_element>; ... };
lmdb-mapsize <sizeval>;
lock-file ( <quoted_string> | none );
managed-keys-directory <quoted_string>;
masterfile-format ( map | raw | text );
masterfile-style ( full | relative );
match-mapped-addresses <boolean>;
max-cache-size ( default | unlimited | <sizeval> | <percentage> );
max-cache-ttl <duration>;
max-clients-per-query <integer>;
max-ixfr-ratio ( unlimited | <percentage> );
max-journal-size ( default | unlimited | <sizeval> );
max-ncache-ttl <duration>;
max-records <integer>;
max-recursion-depth <integer>;
max-recursion-queries <integer>;
max-refresh-time <integer>;
max-retry-time <integer>;
max-rsa-exponent-size <integer>;
max-stale-ttl <duration>;
max-transfer-idle-in <integer>;
max-transfer-idle-out <integer>;
max-transfer-time-in <integer>;
max-transfer-time-out <integer>;
max-udp-size <integer>;
max-zone-ttl ( unlimited | <duration> );
memstatistics <boolean>;
memstatistics-file <quoted_string>;
message-compression <boolean>;
min-cache-ttl <duration>;
min-ncache-ttl <duration>;
min-refresh-time <integer>;
min-retry-time <integer>;
minimal-any <boolean>;
minimal-responses ( no-auth | no-auth-recursive | <boolean> );
multi-master <boolean>;
new-zones-directory <quoted_string>;
no-case-compress { <address_match_element>; ... };
nocookie-udp-size <integer>;
```

```
    notify ( explicit | master-only | primary-only | <boolean> );
    notify-delay <integer>;
    notify-rate <integer>;
    notify-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [
        dscp <integer> ];
    notify-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ]
        [ dscp <integer> ];
    notify-to-soa <boolean>;
    nta-lifetime <duration>;
    nta-recheck <duration>;
    nxdomain-redirect <string>;
    parental-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [
        dscp <integer> ];
    parental-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * )
        ] [ dscp <integer> ];
    pid-file ( <quoted_string> | none );
    port <integer>;
    preferred-glue <string>;
    prefetch <integer> [ <integer> ];
    provide-ixfr <boolean>;
    qname-minimization ( strict | relaxed | disabled | off );
    query-source ( ( [ address ] ( <ipv4_address> | * ) [ port (
        <integer> | * ) ] ) | ( [ [ address ] ( <ipv4_address> | * ) ]
        port ( <integer> | * ) ) ) [ dscp <integer> ];
    query-source-v6 ( ( [ address ] ( <ipv6_address> | * ) [ port (
        <integer> | * ) ] ) | ( [ [ address ] ( <ipv6_address> | * ) ]
        port ( <integer> | * ) ) ) [ dscp <integer> ];
    querylog <boolean>;
    random-device ( <quoted_string> | none );
    rate-limit {
            all-per-second <integer>;
            errors-per-second <integer>;
            exempt-clients { <address_match_element>; ... };
            ipv4-prefix-length <integer>;
            ipv6-prefix-length <integer>;
            log-only <boolean>;
            max-table-size <integer>;
            min-table-size <integer>;
            nodata-per-second <integer>;
            nxdomains-per-second <integer>;
            qps-scale <integer>;
            referrals-per-second <integer>;
            responses-per-second <integer>;
            slip <integer>;
            window <integer>;
    };
    recursing-file <quoted_string>;
    recursion <boolean>;
    recursive-clients <integer>;
    request-expire <boolean>;
    request-ixfr <boolean>;
    request-nsid <boolean>;
    require-server-cookie <boolean>;
    reserved-sockets <integer>;
    resolver-nonbackoff-tries <integer>;
    resolver-query-timeout <integer>;
    resolver-retry-interval <integer>;
```

```
response-padding { <address_match_element>; ... } block-size
    <integer>;
response-policy { zone <string> [ add-soa <boolean> ] [ log
    <boolean> ] [ max-policy-ttl <duration> ] [ min-update-interval
    <duration> ] [ policy ( cname | disabled | drop | given | no-op
    | nodata | nxdomain | passthru | tcp-only <quoted_string> ) ] [
    recursive-only <boolean> ] [ nsip-enable <boolean> ] [
    nsdname-enable <boolean> ]; ... } [ add-soa <boolean> ] [
    break-dnssec <boolean> ] [ max-policy-ttl <duration> ] [
    min-update-interval <duration> ] [ min-ns-dots <integer> ] [
    nsip-wait-recurse <boolean> ] [ qname-wait-recurse <boolean> ]
    [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [
    nsdname-enable <boolean> ] [ dnsrps-enable <boolean> ] [
    dnsrps-options { <unspecified-text> } ];
reuseport <boolean>;
root-delegation-only [ exclude { <string>; ... } ];
root-key-sentinel <boolean>;
rrset-order { [ class <string> ] [ type <string> ] [ name
    <quoted_string> ] <string> <string>; ... };
secroots-file <quoted_string>;
send-cookie <boolean>;
serial-query-rate <integer>;
serial-update-method ( date | increment | unixtime );
server-id ( <quoted_string> | none | hostname );
servfail-ttl <duration>;
session-keyalg <string>;
session-keyfile ( <quoted_string> | none );
session-keyname <string>;
sig-signing-nodes <integer>;
sig-signing-signatures <integer>;
sig-signing-type <integer>;
sig-validity-interval <integer> [ <integer> ];
sortlist { <address_match_element>; ... };
stacksize ( default | unlimited | <sizeval> );
stale-answer-client-timeout ( disabled | off | <integer> );
stale-answer-enable <boolean>;
stale-answer-ttl <duration>;
stale-cache-enable <boolean>;
stale-refresh-time <duration>;
startup-notify-rate <integer>;
statistics-file <quoted_string>;
synth-from-dnssec <boolean>;
tcp-advertised-timeout <integer>;
tcp-clients <integer>;
tcp-idle-timeout <integer>;
tcp-initial-timeout <integer>;
tcp-keepalive-timeout <integer>;
tcp-listen-queue <integer>;
tkey-dhkey <quoted_string> <integer>;
tkey-domain <quoted_string>;
tkey-gssapi-credential <quoted_string>;
tkey-gssapi-keytab <quoted_string>;
transfer-format ( many-answers | one-answer );
transfer-message-size <integer>;
transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [
    dscp <integer> ];
transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * )
```

```
        ] [ dscp <integer> ];
    transfers-in <integer>;
    transfers-out <integer>;
    transfers-per-ns <integer>;
    trust-anchor-telemetry <boolean>; // experimental
    try-tcp-refresh <boolean>;
    update-check-ksk <boolean>;
    use-alt-transfer-source <boolean>;
    use-v4-udp-ports { <portrange>; ... };
    use-v6-udp-ports { <portrange>; ... };
    v6-bias <integer>;
    validate-except { <string>; ... };
    version ( <quoted_string> | none );
    zero-no-soa-ttl <boolean>;
    zero-no-soa-ttl-cache <boolean>;
    zone-statistics ( full | terse | none | <boolean> );
};
```

4.2.16 `options` Statement Definition and Usage

The `options` statement sets up global options to be used by BIND. This statement may appear only once in a configuration file. If there is no `options` statement, an options block with each option set to its default is used.

attach-cache This option allows multiple views to share a single cache database. Each view has its own cache database by default, but if multiple views have the same operational policy for name resolution and caching, those views can share a single cache to save memory, and possibly improve resolution efficiency, by using this option.

The `attach-cache` option may also be specified in `view` statements, in which case it overrides the global `attach-cache` option.

The `cache_name` specifies the cache to be shared. When the `named` server configures views which are supposed to share a cache, it creates a cache with the specified name for the first view of these sharing views. The rest of the views simply refer to the already-created cache.

One common configuration to share a cache is to allow all views to share a single cache. This can be done by specifying `attach-cache` as a global option with an arbitrary name.

Another possible operation is to allow a subset of all views to share a cache while the others retain their own caches. For example, if there are three views A, B, and C, and only A and B should share a cache, specify the `attach-cache` option as a view of A (or B)'s option, referring to the other view name:

```
view "A" {
  // this view has its own cache
  ...
};
view "B" {
  // this view refers to A's cache
  attach-cache "A";
};
view "C" {
  // this view has its own cache
  ...
};
```

Views that share a cache must have the same policy on configurable parameters that may affect caching. The current implementation requires the following configurable options be consistent among

---

these views: `check-names`, `dnssec-accept-expired`, `dnssec-validation`, `max-cache-ttl`, `max-ncache-ttl`, `max-stale-ttl`, `max-cache-size`, `min-cache-ttl`, `min-ncache-ttl`, and `zero-no-soa-ttl`.

Note that there may be other parameters that may cause confusion if they are inconsistent for different views that share a single cache. For example, if these views define different sets of forwarders that can return different answers for the same question, sharing the answer does not make sense or could even be harmful. It is the administrator's responsibility to ensure that configuration differences in different views do not cause disruption with a shared cache.

directory This sets the working directory of the server. Any non-absolute pathnames in the configuration file are taken as relative to this directory. The default location for most server output files (e.g., `named.run`) is this directory. If a directory is not specified, the working directory defaults to `"."`, the directory from which the server was started. The directory specified should be an absolute path, and *must* be writable by the effective user ID of the `named` process.

The option takes effect only at the time that the configuration option is parsed; if other files are being included before or after specifying the new `directory`, the `directory` option must be listed before any other directive (like `include`) that can work with relative files. The safest way to include files is to use absolute file names.

dnstap `dnstap` is a fast, flexible method for capturing and logging DNS traffic. Developed by Robert Edmonds at Farsight Security, Inc., and supported by multiple DNS implementations, `dnstap` uses `libfstrm` (a lightweight high-speed framing library; see https://github.com/farsightsec/fstrm) to send event payloads which are encoded using Protocol Buffers (`libprotobuf-c`, a mechanism for serializing structured data developed by Google, Inc.; see https://developers.google.com/protocol-buffers/).

To enable `dnstap` at compile time, the `fstrm` and `protobuf-c` libraries must be available, and BIND must be configured with `--enable-dnstap`.

The `dnstap` option is a bracketed list of message types to be logged. These may be set differently for each view. Supported types are `client`, `auth`, `resolver`, `forwarder`, and `update`. Specifying type `all` causes all `dnstap` messages to be logged, regardless of type.

Each type may take an additional argument to indicate whether to log `query` messages or `response` messages; if not specified, both queries and responses are logged.

Example: To log all authoritative queries and responses, recursive client responses, and upstream queries sent by the resolver, use:

```
dnstap {
  auth;
  client response;
  resolver query;
};
```

Logged `dnstap` messages can be parsed using the `dnstap-read` utility (see *dnstap-read - print dnstap data in human-readable form* for details).

For more information on `dnstap`, see http://dnstap.info.

The fstrm library has a number of tunables that are exposed in `named.conf`, and can be modified if necessary to improve performance or prevent loss of data. These are:

- `fstrm-set-buffer-hint`: The threshold number of bytes to accumulate in the output buffer before forcing a buffer flush. The minimum is 1024, the maximum is 65536, and the default is 8192.

- `fstrm-set-flush-timeout`: The number of seconds to allow unflushed data to remain in the output buffer. The minimum is 1 second, the maximum is 600 seconds (10 minutes), and the default is 1 second.

- `fstrm-set-output-notify-threshold`: The number of outstanding queue entries to allow on an input queue before waking the I/O thread. The minimum is 1 and the default is 32.

- `fstrm-set-output-queue-model`: The queuing semantics to use for queue objects. The default is `mpsc` (multiple producer, single consumer); the other option is `spsc` (single producer, single consumer).

- `fstrm-set-input-queue-size`: The number of queue entries to allocate for each input queue. This value must be a power of 2. The minimum is 2, the maximum is 16384, and the default is 512.

- `fstrm-set-output-queue-size`: The number of queue entries to allocate for each output queue. The minimum is 2, the maximum is system-dependent and based on `IOV_MAX`, and the default is 64.

- `fstrm-set-reopen-interval`: The number of seconds to wait between attempts to reopen a closed output stream. The minimum is 1 second, the maximum is 600 seconds (10 minutes), and the default is 5 seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value.

Note that all of the above minimum, maximum, and default values are set by the `libfstrm` library, and may be subject to change in future versions of the library. See the `libfstrm` documentation for more information.

dnstap-output This configures the path to which the `dnstap` frame stream is sent if `dnstap` is enabled at compile time and active.

The first argument is either `file` or `unix`, indicating whether the destination is a file or a Unix domain socket. The second argument is the path of the file or socket. (Note: when using a socket, `dnstap` messages are only sent if another process such as `fstrm_capture` (provided with `libfstrm`) is listening on the socket.)

If the first argument is `file`, then up to three additional options can be added: `size` indicates the size to which a `dnstap` log file can grow before being rolled to a new file; `versions` specifies the number of rolled log files to retain; and `suffix` indicates whether to retain rolled log files with an incrementing counter as the suffix (`increment`) or with the current timestamp (`timestamp`). These are similar to the `size`, `versions`, and `suffix` options in a `logging` channel. The default is to allow `dnstap` log files to grow to any size without rolling.

`dnstap-output` can only be set globally in `options`. Currently, it can only be set once while `named` is running; once set, it cannot be changed by `rndc reload` or `rndc reconfig`.

dnstap-identity This specifies an `identity` string to send in `dnstap` messages. If set to `hostname`, which is the default, the server's hostname is sent. If set to `none`, no identity string is sent.

dnstap-version This specifies a `version` string to send in `dnstap` messages. The default is the version number of the BIND release. If set to `none`, no version string is sent.

geoip-directory When `named` is compiled using the MaxMind GeoIP2 geolocation API, this specifies the directory containing GeoIP database files. By default, the option is set based on the prefix used to build the `libmaxminddb` module; for example, if the library is installed in `/usr/local/lib`, then the default `geoip-directory` is `/usr/local/share/GeoIP`. On Windows, the default is the `named` working directory. See *acl Statement Definition and Usage* for details about `geoip` ACLs.

key-directory This is the directory where the public and private DNSSEC key files should be found when performing a dynamic update of secure zones, if different than the current working directory. (Note that this option has no effect on the paths for files containing non-DNSSEC keys such as `bind.keys`, `rndc.key`, or `session.key`.)

lmdb-mapsize When `named` is built with liblmdb, this option sets a maximum size for the memory map of the new-zone database (NZD) in LMDB database format. This database is used to store configuration information for zones added using `rndc addzone`. Note that this is not the NZD database file size, but the largest size that the database may grow to.

Because the database file is memory-mapped, its size is limited by the address space of the `named` process. The default of 32 megabytes was chosen to be usable with 32-bit `named` builds. The largest permitted value is 1 terabyte. Given typical zone configurations without elaborate ACLs, a 32 MB NZD file ought to be able to hold configurations of about 100,000 zones.

managed-keys-directory This specifies the directory in which to store the files that track managed DNSSEC keys (i.e., those configured using the `initial-key` or `initial-ds` keywords in a `trust-anchors` statement).

By default, this is the working directory. The directory *must* be writable by the effective user ID of the `named` process.

If `named` is not configured to use views, managed keys for the server are tracked in a single file called `managed-keys.bind`. Otherwise, managed keys are tracked in separate files, one file per view; each file name is the view name (or, if it contains characters that are incompatible with use as a file name, the SHA256 hash of the view name), followed by the extension `.mkeys`.

(Note: in earlier releases, file names for views always used the SHA256 hash of the view name. To ensure compatibility after upgrading, if a file using the old name format is found to exist, it is used instead of the new format.)

max-ixfr-ratio This sets the size threshold (expressed as a percentage of the size of the full zone) beyond which `named` chooses to use an AXFR response rather than IXFR when answering zone transfer requests. See *Incremental Zone Transfers (IXFR)*.

The minimum value is `1%`. The keyword `unlimited` disables ratio checking and allows IXFRs of any size. The default is `unlimited`.

new-zones-directory This specifies the directory in which to store the configuration parameters for zones added via `rndc addzone`. By default, this is the working directory. If set to a relative path, it is relative to the working directory. The directory *must* be writable by the effective user ID of the `named` process.

qname-minimization This option controls QNAME minimization behavior in the BIND resolver. When set to `strict`, BIND follows the QNAME minimization algorithm to the letter, as specified in **RFC 7816**. Setting this option to `relaxed` causes BIND to fall back to normal (non-minimized) query mode when it receives either NXDOMAIN or other unexpected responses (e.g., SERVFAIL, improper zone cut, REFUSED) to a minimized query. `disabled` disables QNAME minimization completely. The current default is `relaxed`, but it may be changed to `strict` in a future release.

tkey-gssapi-keytab This is the KRB5 keytab file to use for GSS-TSIG updates. If this option is set and tkey-gssapi-credential is not set, updates are allowed with any key matching a principal in the specified keytab.

tkey-gssapi-credential This is the security credential with which the server should authenticate keys requested by the GSS-TSIG protocol. Currently only Kerberos 5 authentication is available; the credential is a Kerberos principal which the server can acquire through the default system key file, normally `/etc/krb5.keytab`. The location of the keytab file can be overridden using the `tkey-gssapi-keytab` option. Normally this principal is of the form `DNS/server.domain`. To use GSS-TSIG, `tkey-domain` must also be set if a specific keytab is not set with `tkey-gssapi-keytab`.

tkey-domain This domain is appended to the names of all shared keys generated with `TKEY`. When a client requests a `TKEY` exchange, it may or may not specify the desired name for the key. If present, the name of the shared key is `client-specified part` + `tkey-domain`. Otherwise, the name of the shared key is `random hex digits` + `tkey-domain`. In most cases, the `domainname` should be the server's domain name, or an otherwise nonexistent subdomain like `_tkey.domainname`. If using GSS-TSIG, this variable must be defined, unless a specific keytab is specified using `tkey-gssapi-keytab`.

tkey-dhkey This is the Diffie-Hellman key used by the server to generate shared keys with clients using the Diffie-Hellman mode of `TKEY`. The server must be able to load the public and private keys from files in the working directory. In most cases, the `key_name` should be the server's host name.

cache-file This is for testing only. Do not use.

dump-file This is the pathname of the file the server dumps the database to, when instructed to do so with `rndc dumpdb`. If not specified, the default is `named_dump.db`.

memstatistics-file This is the pathname of the file the server writes memory usage statistics to on exit. If not specified, the default is `named.memstats`.

lock-file This is the pathname of a file on which `named` attempts to acquire a file lock when starting for the first time; if unsuccessful, the server terminates, under the assumption that another server is already running. If not specified, the default is `none`.

Specifying `lock-file none` disables the use of a lock file. `lock-file` is ignored if `named` was run using the `-X` option, which overrides it. Changes to `lock-file` are ignored if `named` is being reloaded or reconfigured; it is only effective when the server is first started.

pid-file This is the pathname of the file the server writes its process ID in. If not specified, the default is `/var/run/named/named.pid`. The PID file is used by programs that send signals to the running name server. Specifying `pid-file none` disables the use of a PID file; no file is written and any existing one is removed. Note that `none` is a keyword, not a filename, and therefore is not enclosed in double quotes.

recursing-file This is the pathname of the file where the server dumps the queries that are currently recursing, when instructed to do so with `rndc recursing`. If not specified, the default is `named.recursing`.

statistics-file This is the pathname of the file the server appends statistics to, when instructed to do so using `rndc stats`. If not specified, the default is `named.stats` in the server's current directory. The format of the file is described in *The Statistics File*.

bindkeys-file This is the pathname of a file to override the built-in trusted keys provided by `named`. See the discussion of `dnssec-validation` for details. If not specified, the default is `/etc/bind.keys`.

secroots-file This is the pathname of the file the server dumps security roots to, when instructed to do so with `rndc secroots`. If not specified, the default is `named.secroots`.

session-keyfile This is the pathname of the file into which to write a TSIG session key generated by `named` for use by `nsupdate -l`. If not specified, the default is `/var/run/named/session.key`. (See *Dynamic Update Policies*, and in particular the discussion of the `update-policy` statement's `local` option, for more information about this feature.)

session-keyname This is the key name to use for the TSIG session key. If not specified, the default is `local-ddns`.

session-keyalg This is the algorithm to use for the TSIG session key. Valid values are hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, hmac-sha512, and hmac-md5. If not specified, the default is hmac-sha256.

port This is the UDP/TCP port number the server uses to receive and send DNS protocol traffic. The default is 53. This option is mainly intended for server testing; a server using a port other than 53 is not able to communicate with the global DNS.

dscp This is the global Differentiated Services Code Point (DSCP) value to classify outgoing DNS traffic, on operating systems that support DSCP. Valid values are 0 through 63. It is not configured by default.

random-device This specifies a source of entropy to be used by the server; it is a device or file from which to read entropy. If it is a file, operations requiring entropy will fail when the file has been exhausted.

Entropy is needed for cryptographic operations such as TKEY transactions, dynamic update of signed zones, and generation of TSIG session keys. It is also used for seeding and stirring the pseudo-random number generator which is used for less critical functions requiring randomness, such as generation of DNS message transaction IDs.

If `random-device` is not specified, or if it is set to `none`, entropy is read from the random number generation function supplied by the cryptographic library with which BIND was linked (i.e. OpenSSL or a PKCS#11 provider).

The `random-device` option takes effect during the initial configuration load at server startup time and is ignored on subsequent reloads.

preferred-glue If specified, the listed type (A or AAAA) is emitted before other glue in the additional section of a query response. The default is to prefer A records when responding to queries that arrived via IPv4 and AAAA when responding to queries that arrived via IPv6.

root-delegation-only This turns on enforcement of delegation-only in TLDs (top-level domains) and root zones with an optional exclude list.

DS queries are expected to be made to and be answered by delegation-only zones. Such queries and responses are treated as an exception to delegation-only processing and are not converted to NXDOMAIN responses, provided a

BIND 9 Administrator Reference Manual, Release 9.16.28

CNAME is not discovered at the query name.

If a delegation-only zone server also serves a child zone, it is not always possible to determine whether an answer comes from the delegation-only zone or the child zone. SOA NS and DNSKEY records are apex-only records and a matching response that contains these records or DS is treated as coming from a child zone. RRSIG records are also examined to see whether they are signed by a child zone, and the authority section is examined to see if there is evidence that the answer is from the child zone. Answers that are determined to be from a child zone are not converted to NXDOMAIN responses. Despite all these checks, there is still a possibility of false negatives when a child zone is being served.

Similarly, false positives can arise from empty nodes (no records at the name) in the delegation-only zone when the query type is not `ANY`.

Note that some TLDs are not delegation-only; e.g., "DE", "LV", "US", and "MUSEUM". This list is not exhaustive.

```
options {
    root-delegation-only exclude { "de"; "lv"; "us"; "museum"; };
};
```

disable-algorithms This disables the specified DNSSEC algorithms at and below the specified name. Multiple `disable-algorithms` statements are allowed. Only the best-match `disable-algorithms` clause is used to determine the algorithms.

If all supported algorithms are disabled, the zones covered by the `disable-algorithms` setting are treated as insecure.

Configured trust anchors in `trust-anchors` (or `managed-keys` or `trusted-keys`) that match a disabled algorithm are ignored and treated as if they were not configured.

disable-ds-digests This disables the specified DS digest types at and below the specified name. Multiple `disable-ds-digests` statements are allowed. Only the best-match `disable-ds-digests` clause is used to determine the digest types.

If all supported digest types are disabled, the zones covered by `disable-ds-digests` are treated as insecure.

dnssec-must-be-secure This specifies hierarchies which must be or may not be secure (signed and validated). If `yes`, then `named` only accepts answers if they are secure. If `no`, then normal DNSSEC validation applies, allowing insecure answers to be accepted. The specified domain must be defined as a trust anchor, for instance in a `trust-anchors` statement, or `dnssec-validation auto` must be active.

dns64 This directive instructs `named` to return mapped IPv4 addresses to AAAA queries when there are no AAAA records. It is intended to be used in conjunction with a NAT64. Each `dns64` defines one DNS64 prefix. Multiple DNS64 prefixes can be defined.

Compatible IPv6 prefixes have lengths of 32, 40, 48, 56, 64, and 96, per **RFC 6052**. Bits 64..71 inclusive must be zero, with the most significant bit of the prefix in position 0.

In addition, a reverse IP6.ARPA zone is created for the prefix to provide a mapping from the IP6.ARPA names to the corresponding IN-ADDR.ARPA names using synthesized CNAMEs. `dns64-server` and `dns64-contact` can be used to specify the name of the server and contact for the zones. These can be set at the view/options level but not on a per-prefix basis.

Each `dns64` supports an optional `clients` ACL that determines which clients are affected by this directive. If not defined, it defaults to `any;`.

Each `dns64` supports an optional `mapped` ACL that selects which IPv4 addresses are to be mapped in the corresponding A RRset. If not defined, it defaults to `any;`.

Normally, DNS64 does not apply to a domain name that owns one or more AAAA records; these records are simply returned. The optional `exclude` ACL allows specification of a list of IPv6 addresses that are ignored if

4.2. Configuration File Grammar 43

they appear in a domain name's AAAA records; DNS64 is applied to any A records the domain name owns. If not defined, `exclude` defaults to ::ffff:0.0.0.0/96.

An optional `suffix` can also be defined to set the bits trailing the mapped IPv4 address bits. By default these bits are set to `::`. The bits matching the prefix and mapped IPv4 address must be zero.

If `recursive-only` is set to `yes`, the DNS64 synthesis only happens for recursive queries. The default is `no`.

If `break-dnssec` is set to `yes`, the DNS64 synthesis happens even if the result, if validated, would cause a DNSSEC validation failure. If this option is set to `no` (the default), the DO is set on the incoming query, and there are RRSIGs on the applicable records, then synthesis does not happen.

```
acl rfc1918 { 10/8; 192.168/16; 172.16/12; };

dns64 64:FF9B::/96 {
    clients { any; };
    mapped { !rfc1918; any; };
    exclude { 64:FF9B::/96; ::ffff:0000:0000/96; };
    suffix ::;
};
```

dnssec-loadkeys-interval When a zone is configured with `auto-dnssec maintain;`, its key repository must be checked periodically to see if any new keys have been added or any existing keys' timing metadata has been updated (see *dnssec-keygen: DNSSEC key generation tool* and *dnssec-settime: set the key timing metadata for a DNSSEC key*). The `dnssec-loadkeys-interval` option sets the frequency of automatic repository checks, in minutes. The default is `60` (1 hour), the minimum is `1` (1 minute), and the maximum is `1440` (24 hours); any higher value is silently reduced.

dnssec-policy This specifies which key and signing policy (KASP) should be used for this zone. This is a string referring to a `dnssec-policy` statement. There are three built-in policies: `default`, which uses the default policy, `insecure`, to be used when you want to gracefully unsign your zone, and `none`, which means no DNSSEC policy. The default is `none`. See *dnssec-policy Grammar* for more details.

dnssec-update-mode If this option is set to its default value of `maintain` in a zone of type `primary` which is DNSSEC-signed and configured to allow dynamic updates (see *Dynamic Update Policies*), and if `named` has access to the private signing key(s) for the zone, then `named` automatically signs all new or changed records and maintains signatures for the zone by regenerating RRSIG records whenever they approach their expiration date.

If the option is changed to `no-resign`, then `named` signs all new or changed records, but scheduled maintenance of signatures is disabled.

With either of these settings, `named` rejects updates to a DNSSEC-signed zone when the signing keys are inactive or unavailable to `named`. (A planned third option, `external`, will disable all automatic signing and allow DNSSEC data to be submitted into a zone via dynamic update; this is not yet implemented.)

nta-lifetime This specifies the default lifetime, in seconds, for negative trust anchors added via `rndc nta`.

A negative trust anchor selectively disables DNSSEC validation for zones that are known to be failing because of misconfiguration, rather than an attack. When data to be validated is at or below an active NTA (and above any other configured trust anchors), `named` aborts the DNSSEC validation process and treats the data as insecure rather than bogus. This continues until the NTA's lifetime has elapsed. NTAs persist across `named` restarts.

For convenience, TTL-style time-unit suffixes can be used to specify the NTA lifetime in seconds, minutes, or hours. It also accepts ISO 8601 duration formats.

`nta-lifetime` defaults to one hour; it cannot exceed one week.

nta-recheck This specifies how often to check whether negative trust anchors added via `rndc nta` are still necessary.

A negative trust anchor is normally used when a domain has stopped validating due to operator error; it temporarily disables DNSSEC validation for that domain. In the interest of ensuring that DNSSEC validation is turned back on as soon as possible, `named` periodically sends a query to the domain, ignoring negative trust anchors, to find out whether it can now be validated. If so, the negative trust anchor is allowed to expire early.

Validity checks can be disabled for an individual NTA by using `rndc nta -f`, or for all NTAs by setting `nta-recheck` to zero.

For convenience, TTL-style time-unit suffixes can be used to specify the NTA recheck interval in seconds, minutes, or hours. It also accepts ISO 8601 duration formats.

The default is five minutes. It cannot be longer than `nta-lifetime`, which cannot be longer than a week.

max-zone-ttl This specifies a maximum permissible TTL value in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the maximum value. When loading a zone file using a `masterfile-format` of `text` or `raw`, any record encountered with a TTL higher than `max-zone-ttl` causes the zone to be rejected.

This is useful in DNSSEC-signed zones because when rolling to a new DNSKEY, the old key needs to remain available until RRSIG records have expired from caches. The `max-zone-ttl` option guarantees that the largest TTL in the zone is no higher than the set value.

(Note: because `map`-format files load directly into memory, this option cannot be used with them.)

The default value is `unlimited`. A `max-zone-ttl` of zero is treated as `unlimited`.

stale-answer-ttl This specifies the TTL to be returned on stale answers. The default is 30 seconds. The minimum allowed is 1 second; a value of 0 is updated silently to 1 second.

For stale answers to be returned, they must be enabled, either in the configuration file using `stale-answer-enable` or via `rndc serve-stale on`.

serial-update-method Zones configured for dynamic DNS may use this option to set the update method to be used for the zone serial number in the SOA record.

With the default setting of `serial-update-method increment;`, the SOA serial number is incremented by one each time the zone is updated.

When set to `serial-update-method unixtime;`, the SOA serial number is set to the number of seconds since the Unix epoch, unless the serial number is already greater than or equal to that value, in which case it is simply incremented by one.

When set to `serial-update-method date;`, the new SOA serial number is the current date in the form "YYYYMMDD", followed by two zeroes, unless the existing serial number is already greater than or equal to that value, in which case it is incremented by one.

zone-statistics If `full`, the server collects statistical data on all zones, unless specifically turned off on a per-zone basis by specifying `zone-statistics terse` or `zone-statistics none` in the `zone` statement. The statistical data includes, for example, DNSSEC signing operations and the number of authoritative answers per query type. The default is `terse`, providing minimal statistics on zones (including name and current serial number, but not query type counters).

These statistics may be accessed via the `statistics-channel` or using `rndc stats`, which dumps them to the file listed in the `statistics-file`. See also *The Statistics File*.

For backward compatibility with earlier versions of BIND 9, the `zone-statistics` option can also accept `yes` or `no`; `yes` has the same meaning as `full`. As of BIND 9.10, `no` has the same meaning as `none`; previously, it was the same as `terse`.

Boolean Options

automatic-interface-scan If `yes` and supported by the operating system, this automatically rescans network interfaces when the interface addresses are added or removed. The default is `yes`. This configuration option does not affect the time-based `interface-interval` option; it is recommended to set the time-based `interface-interval` to 0 when the operator confirms that automatic interface scanning is supported by the operating system.

The `automatic-interface-scan` implementation uses routing sockets for the network interface discovery; therefore, the operating system must support the routing sockets for this feature to work.

allow-new-zones If `yes`, then zones can be added at runtime via `rndc addzone`. The default is `no`.

Newly added zones' configuration parameters are stored so that they can persist after the server is restarted. The configuration information is saved in a file called `viewname.nzf` (or, if `named` is compiled with liblmdb, in an LMDB database file called `viewname.nzd`). "viewname" is the name of the view, unless the view name contains characters that are incompatible with use as a file name, in which case a cryptographic hash of the view name is used instead.

Configurations for zones added at runtime are stored either in a new-zone file (NZF) or a new-zone database (NZD), depending on whether `named` was linked with liblmdb at compile time. See *rndc - name server control utility* for further details about `rndc addzone`.

auth-nxdomain If `yes`, then the `AA` bit is always set on NXDOMAIN responses, even if the server is not actually authoritative. The default is `no`.

memstatistics This writes memory statistics to the file specified by `memstatistics-file` at exit. The default is `no` unless `-m record` is specified on the command line, in which case it is `yes`.

dialup If `yes`, then the server treats all zones as if they are doing zone transfers across a dial-on-demand dialup link, which can be brought up by traffic originating from this server. Although this setting has different effects according to zone type, it concentrates the zone maintenance so that everything happens quickly, once every `heartbeat-interval`, ideally during a single call. It also suppresses some normal zone maintenance traffic. The default is `no`.

If specified in the `view` and `zone` statements, the `dialup` option overrides the global `dialup` option.

If the zone is a primary zone, the server sends out a NOTIFY request to all the secondaries (default). This should trigger the zone serial number check in the secondary (providing it supports NOTIFY), allowing the secondary to verify the zone while the connection is active. The set of servers to which NOTIFY is sent can be controlled by `notify` and `also-notify`.

If the zone is a secondary or stub zone, the server suppresses the regular "zone up to date" (refresh) queries and only performs them when the `heartbeat-interval` expires, in addition to sending NOTIFY requests.

Finer control can be achieved by using `notify`, which only sends NOTIFY messages; `notify-passive`, which sends NOTIFY messages and suppresses the normal refresh queries; `refresh`, which suppresses normal refresh processing and sends refresh queries when the `heartbeat-interval` expires; and `passive`, which disables normal refresh processing.

dialup mode	normal refresh	heart-beat refresh	heart-beat notify
`no` (default)	yes	no	no
`yes`	no	yes	yes
`notify`	yes	no	yes
`refresh`	no	yes	no
`passive`	no	no	no
`notify-passive`	no	no	yes

Note that normal NOTIFY processing is not affected by `dialup`.

flush-zones-on-shutdown When the name server exits upon receiving SIGTERM, flush or do not flush any pending zone writes. The default is `flush-zones-on-shutdown no`.

geoip-use-ecs This option was part of an experimental implementation of the EDNS CLIENT-SUBNET for authoritative servers, but is now obsolete.

root-key-sentinel If `yes`, respond to root key sentinel probes as described in draft-ietf-dnsop-kskroll-sentinel-08. The default is `yes`.

reuseport This option enables kernel load-balancing of sockets on systems which support it, including Linux (SO_REUSEPORT) and FreeBSD (SO_REUSEPORT_LB). This instructs the kernel to distribute incoming socket connections among the networking threads based on a hashing scheme. For more information, see the receive network flow classification options (`rx-flow-hash`) section in the `ethtool` manual page. The default is `yes`.

Enabling `reuseport` significantly increases general throughput when incoming traffic is distributed uniformly onto the threads by the operating system. However, in cases where a worker thread is busy with a long-lasting operation, such as processing a Response Policy Zone (RPZ) or Catalog Zone update or an unusually large zone transfer, incoming traffic that hashes onto that thread may be delayed. On servers where these events occur frequently, it may be preferable to disable socket load-balancing so that other threads can pick up the traffic that would have been sent to the busy thread.

Note: this option can only be set when `named` first starts. Changes will not take effect during reconfiguration; the server must be restarted.

message-compression If `yes`, DNS name compression is used in responses to regular queries (not including AXFR or IXFR, which always use compression). Setting this option to `no` reduces CPU usage on servers and may improve throughput. However, it increases response size, which may cause more queries to be processed using TCP; a server with compression disabled is out of compliance with **RFC 1123** Section 6.1.3.2. The default is `yes`.

minimal-responses This option controls the addition of records to the authority and additional sections of responses. Such records may be included in responses to be helpful to clients; for example, NS or MX records may have associated address records included in the additional section, obviating the need for a separate address lookup. However, adding these records to responses is not mandatory and requires additional database lookups, causing extra latency when marshalling responses. `minimal-responses` takes one of four values:

- `no`: the server is as complete as possible when generating responses.

- `yes`: the server only adds records to the authority and additional sections when such records are required by the DNS protocol (for example, when returning delegations or negative responses). This provides the best server performance but may result in more client queries.

- `no-auth`: the server omits records from the authority section except when they are required, but it may still add records to the additional section.

- `no-auth-recursive`: the same as `no-auth` when recursion is requested in the query (RD=1), or the same as `no` if recursion is not requested.

`no-auth` and `no-auth-recursive` are useful when answering stub clients, which usually ignore the authority section. `no-auth-recursive` is meant for use in mixed-mode servers that handle both authoritative and recursive queries.

The default is `no-auth-recursive`.

glue-cache When set to `yes`, a cache is used to improve query performance when adding address-type (A and AAAA) glue records to the additional section of DNS response messages that delegate to a child zone.

The glue cache uses memory proportional to the number of delegations in the zone. The default setting is `yes`, which improves performance at the cost of increased memory usage for the zone. To avoid this, set it to `no`.

minimal-any If set to `yes`, the server replies with only one of the RRsets for the query name, and its covering RRSIGs if any, when generating a positive response to a query of type ANY over UDP, instead of replying with all known RRsets for the name. Similarly, a query for type RRSIG is answered with the RRSIG records covering only

---

one type. This can reduce the impact of some kinds of attack traffic, without harming legitimate clients. (Note, however, that the RRset returned is the first one found in the database; it is not necessarily the smallest available RRset.) Additionally, `minimal-responses` is turned on for these queries, so no unnecessary records are added to the authority or additional sections. The default is `no`.

notify If set to `yes` (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes; see *Notify*. The messages are sent to the servers listed in the zone's NS records (except the primary server identified in the SOA MNAME field), and to any servers listed in the `also-notify` option.

If set to `primary-only` (or the older keyword `master-only`), notifies are only sent for primary zones. If set to `explicit`, notifies are sent only to servers explicitly listed using `also-notify`. If set to `no`, no notifies are sent.

The `notify` option may also be specified in the `zone` statement, in which case it overrides the `options notify` statement. It would only be necessary to turn off this option if it caused secondary zones to crash.

notify-to-soa If `yes`, do not check the name servers in the NS RRset against the SOA MNAME. Normally a NOTIFY message is not sent to the SOA MNAME (SOA ORIGIN), as it is supposed to contain the name of the ultimate primary server. Sometimes, however, a secondary server is listed as the SOA MNAME in hidden primary configurations; in that case, the ultimate primary should be set to still send NOTIFY messages to all the name servers listed in the NS RRset.

recursion If `yes`, and a DNS query requests recursion, then the server attempts to do all the work required to answer the query. If recursion is off and the server does not already know the answer, it returns a referral response. The default is `yes`. Note that setting `recursion no` does not prevent clients from getting data from the server's cache; it only prevents new data from being cached as an effect of client queries. Caching may still occur as an effect of the server's internal operation, such as NOTIFY address lookups.

request-nsid If `yes`, then an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution. If the authoritative server returns an NSID option in its response, then its contents are logged in the `nsid` category at level `info`. The default is `no`.

request-sit This experimental option is obsolete.

require-server-cookie If `yes`, require a valid server cookie before sending a full response to a UDP request from a cookie-aware client. BADCOOKIE is sent if there is a bad or nonexistent server cookie.

The default is `no`.

Users wishing to test that DNS COOKIE clients correctly handle BADCOOKIE, or who are getting a lot of forged DNS requests with DNS COOKIES present, should set this to `yes`. Setting this to `yes` results in a reduced amplification effect in a reflection attack, as the BADCOOKIE response is smaller than a full response, while also requiring a legitimate client to follow up with a second query with the new, valid, cookie.

answer-cookie When set to the default value of `yes`, COOKIE EDNS options are sent when applicable in replies to client queries. If set to `no`, COOKIE EDNS options are not sent in replies. This can only be set at the global options level, not per-view.

`answer-cookie no` is intended as a temporary measure, for use when `named` shares an IP address with other servers that do not yet support DNS COOKIE. A mismatch between servers on the same address is not expected to cause operational problems, but the option to disable COOKIE responses so that all servers have the same behavior is provided out of an abundance of caution. DNS COOKIE is an important security mechanism, and should not be disabled unless absolutely necessary.

send-cookie If `yes`, then a COOKIE EDNS option is sent along with the query. If the resolver has previously communicated with the server, the COOKIE returned in the previous transaction is sent. This is used by the server to determine whether the resolver has talked to it before. A resolver sending the correct COOKIE is assumed not to be an off-path attacker sending a spoofed-source query; the query is therefore unlikely to be part of a reflection/amplification attack, so resolvers sending a correct COOKIE option are not subject to response rate limiting

(RRL). Resolvers which do not send a correct COOKIE option may be limited to receiving smaller responses via the `nocookie-udp-size` option.

The default is `yes`.

stale-answer-enable If `yes`, enable the returning of "stale" cached answers when the name servers for a zone are not answering and the `stale-cache-enable` option is also enabled. The default is not to return stale answers.

Stale answers can also be enabled or disabled at runtime via `rndc serve-stale on` or `rndc serve-stale off`; these override the configured setting. `rndc serve-stale reset` restores the setting to the one specified in `named.conf`. Note that if stale answers have been disabled by `rndc`, they cannot be re-enabled by reloading or reconfiguring `named`; they must be re-enabled with `rndc serve-stale on`, or the server must be restarted.

Information about stale answers is logged under the `serve-stale` log category.

stale-answer-client-timeout This option defines the amount of time (in milliseconds) that `named` waits before attempting to answer the query with a stale RRset from cache. If a stale answer is found, `named` continues the ongoing fetches, attempting to refresh the RRset in cache until the `resolver-query-timeout` interval is reached.

This option is off by default, which is equivalent to setting it to `off` or `disabled`. It also has no effect if `stale-answer-enable` is disabled.

The maximum value for this option is `resolver-query-timeout` minus one second. The minimum value, `0`, causes a cached (stale) RRset to be immediately returned if it is available while still attempting to refresh the data in cache. **RFC 8767** recommends a value of `1800` (milliseconds).

stale-cache-enable If `yes`, enable the retaining of "stale" cached answers. Default `yes`.

stale-refresh-time If the name servers for a given zone are not answering, this sets the time window for which `named` will promptly return "stale" cached answers for that RRSet being requested before a new attempt in contacting the servers is made. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The default `stale-refresh-time` is 30 seconds, as **RFC 8767** recommends that attempts to refresh to be done no more frequently than every 30 seconds. A value of zero disables the feature, meaning that normal resolution will take place first, if that fails only then `named` will return "stale" cached answers.

nocookie-udp-size This sets the maximum size of UDP responses that are sent to queries without a valid server COOKIE. A value below 128 is silently raised to 128. The default value is 4096, but the `max-udp-size` option may further limit the response size as the default for `max-udp-size` is 4096.

sit-secret This experimental option is obsolete.

cookie-algorithm This sets the algorithm to be used when generating the server cookie; the options are "aes" or "siphash24". The default is "siphash24". The "aes" option remains for legacy purposes.

cookie-secret If set, this is a shared secret used for generating and verifying EDNS COOKIE options within an anycast cluster. If not set, the system generates a random secret at startup. The shared secret is encoded as a hex string and needs to be 128 bits for either "siphash24" or "aes".

If there are multiple secrets specified, the first one listed in `named.conf` is used to generate new server cookies. The others are only used to verify returned cookies.

response-padding The EDNS Padding option is intended to improve confidentiality when DNS queries are sent over an encrypted channel, by reducing the variability in packet sizes. If a query:

1. contains an EDNS Padding option,

2. includes a valid server cookie or uses TCP,

3. is not signed using TSIG or SIG(0), and

4. is from a client whose address matches the specified ACL,

then the response is padded with an EDNS Padding option to a multiple of `block-size` bytes. If these conditions are not met, the response is not padded.

If `block-size` is 0 or the ACL is `none;`, this feature is disabled and no padding occurs; this is the default. If `block-size` is greater than 512, a warning is logged and the value is truncated to 512. Block sizes are ordinarily expected to be powers of two (for instance, 128), but this is not mandatory.

trust-anchor-telemetry This causes `named` to send specially formed queries once per day to domains for which trust anchors have been configured via, e.g., `trust-anchors` or `dnssec-validation auto`.

The query name used for these queries has the form `_ta-xxxx(-xxxx)(...).<domain>`, where each "xxxx" is a group of four hexadecimal digits representing the key ID of a trusted DNSSEC key. The key IDs for each domain are sorted smallest to largest prior to encoding. The query type is NULL.

By monitoring these queries, zone operators are able to see which resolvers have been updated to trust a new key; this may help them decide when it is safe to remove an old one.

The default is `yes`.

use-ixfr *This option is obsolete*. To disable IXFR to a particular server or servers, see the information on the `provide-ixfr` option in *server Statement Definition and Usage*. See also *Incremental Zone Transfers (IXFR)*.

provide-ixfr See the description of `provide-ixfr` in *server Statement Definition and Usage*.

request-ixfr See the description of `request-ixfr` in *server Statement Definition and Usage*.

request-expire See the description of `request-expire` in *server Statement Definition and Usage*.

match-mapped-addresses If `yes`, then an IPv4-mapped IPv6 address matches any address-match list entries that match the corresponding IPv4 address.

This option was introduced to work around a kernel quirk in some operating systems that causes IPv4 TCP connections, such as zone transfers, to be accepted on an IPv6 socket using mapped addresses. This caused address-match lists designed for IPv4 to fail to match. However, `named` now solves this problem internally. The use of this option is discouraged.

ixfr-from-differences When `yes` and the server loads a new version of a primary zone from its zone file or receives a new version of a secondary file via zone transfer, it compares the new version to the previous one and calculates a set of differences. The differences are then logged in the zone's journal file so that the changes can be transmitted to downstream secondaries as an incremental zone transfer.

By allowing incremental zone transfers to be used for non-dynamic zones, this option saves bandwidth at the expense of increased CPU and memory consumption at the primary server. In particular, if the new version of a zone is completely different from the previous one, the set of differences is of a size comparable to the combined size of the old and new zone versions, and the server needs to temporarily allocate memory to hold this complete difference set.

`ixfr-from-differences` also accepts `primary` and `secondary` at the view and options levels, which causes `ixfr-from-differences` to be enabled for all primary or secondary zones, respectively. It is off for all zones by default.

Note: if inline signing is enabled for a zone, the user-provided `ixfr-from-differences` setting is ignored for that zone.

multi-master This should be set when there are multiple primary servers for a zone and the addresses refer to different machines. If `yes`, `named` does not log when the serial number on the primary is less than what `named` currently has. The default is `no`.

auto-dnssec Zones configured for dynamic DNS may use this option to allow varying levels of automatic DNSSEC key management. There are three possible settings:

`auto-dnssec allow;` permits keys to be updated and the zone fully re-signed whenever the user issues the command `rndc sign zonename`.

`auto-dnssec maintain;` includes the above, but also automatically adjusts the zone's DNSSEC keys on a schedule, according to the keys' timing metadata (see *dnssec-keygen: DNSSEC key generation tool* and *dnssec-settime: set the key timing metadata for a DNSSEC key*). The command `rndc sign zonename` causes `named` to load keys from the key repository and sign the zone with all keys that are active. `rndc loadkeys zonename` causes `named` to load keys from the key repository and schedule key maintenance events to occur in the future, but it does not sign the full zone immediately. Note: once keys have been loaded for a zone the first time, the repository is searched for changes periodically, regardless of whether `rndc loadkeys` is used. The recheck interval is defined by `dnssec-loadkeys-interval`.

`auto-dnssec off;` does not allow for DNSSEC key management. This is the default setting.

This option may only be activated at the zone level; if configured at the view or options level, it must be set to `off`.

dnssec-enable This option is obsolete and has no effect.

dnssec-validation This option enables DNSSEC validation in `named`.

If set to `auto`, DNSSEC validation is enabled and a default trust anchor for the DNS root zone is used.

If set to `yes`, DNSSEC validation is enabled, but a trust anchor must be manually configured using a `trust-anchors` statement (or the `managed-keys` or `trusted-keys` statements, both deprecated). If there is no configured trust anchor, validation does not take place.

If set to `no`, DNSSEC validation is disabled.

The default is `auto`, unless BIND is built with `configure --disable-auto-validation`, in which case the default is `yes`.

The default root trust anchor is stored in the file `bind.keys`. `named` loads that key at startup if `dnssec-validation` is set to `auto`. A copy of the file is installed along with BIND 9, and is current as of the release date. If the root key expires, a new copy of `bind.keys` can be downloaded from https://www.isc.org/bind-keys.

(To prevent problems if `bind.keys` is not found, the current trust anchor is also compiled in `named`. Relying on this is not recommended, however, as it requires `named` to be recompiled with a new key when the root key expires.)

Note: `named` loads *only* the root key from `bind.keys`. The file cannot be used to store keys for other zones. The root key in `bind.keys` is ignored if `dnssec-validation auto` is not in use.

Whenever the resolver sends out queries to an EDNS-compliant server, it always sets the DO bit indicating it can support DNSSEC responses, even if `dnssec-validation` is off.

---

validate-except This specifies a list of domain names at and beneath which DNSSEC validation should *not* be performed, regardless of the presence of a trust anchor at or above those names. This may be used, for example, when configuring a top-level domain intended only for local use, so that the lack of a secure delegation for that domain in the root zone does not cause validation failures. (This is similar to setting a negative trust anchor except that it is a permanent configuration, whereas negative trust anchors expire and are removed after a set period of time.)

dnssec-accept-expired This accepts expired signatures when verifying DNSSEC signatures. The default is `no`. Setting this option to `yes` leaves `named` vulnerable to replay attacks.

querylog Query logging provides a complete log of all incoming queries and all query errors. This provides more insight into the server's activity, but with a cost to performance which may be significant on heavily loaded servers.

The `querylog` option specifies whether query logging should be active when `named` first starts. If `querylog` is not specified, then query logging is determined by the presence of the logging category `queries`. Query logging can also be activated at runtime using the command `rndc querylog on`, or deactivated with `rndc querylog off`.

check-names This option is used to restrict the character set and syntax of certain domain names in primary files and/or DNS responses received from the network. The default varies according to usage area. For `primary` zones the default is `fail`. For `secondary` zones the default is `warn`. For answers received from the network (`response`), the default is `ignore`.

The rules for legal hostnames and mail domains are derived from **RFC 952** and **RFC 821** as modified by **RFC 1123**.

`check-names` applies to the owner names of A, AAAA, and MX records. It also applies to the domain names in the RDATA of NS, SOA, MX, and SRV records. It further applies to the RDATA of PTR records where the owner name indicates that it is a reverse lookup of a hostname (the owner name ends in IN-ADDR.ARPA, IP6.ARPA, or IP6.INT).

check-dup-records This checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS. The default is to `warn`. Other possible values are `fail` and `ignore`.

check-mx This checks whether the MX record appears to refer to an IP address. The default is to `warn`. Other possible values are `fail` and `ignore`.

check-wildcard This option is used to check for non-terminal wildcards. The use of non-terminal wildcards is almost always as a result of a lack of understanding of the wildcard matching algorithm (**RFC 1034**). This option affects primary zones. The default (`yes`) is to check for non-terminal wildcards and issue a warning.

check-integrity This performs post-load zone integrity checks on primary zones. It checks that MX and SRV records refer to address (A or AAAA) records and that glue address records exist for delegated zones. For MX and SRV records, only in-zone hostnames are checked (for out-of-zone hostnames, use `named-checkzone`). For NS records, only names below top-of-zone are checked (for out-of-zone names and glue consistency checks, use `named-checkzone`). The default is `yes`.

The use of the SPF record to publish Sender Policy Framework is deprecated, as the migration from using TXT records to SPF records was abandoned. Enabling this option also checks that a TXT Sender Policy Framework record exists (starts with "v=spf1") if there is an SPF record. Warnings are emitted if the TXT record does not exist; they can be suppressed with `check-spf`.

check-mx-cname If `check-integrity` is set, then fail, warn, or ignore MX records that refer to CNAMES. The default is to `warn`.

check-srv-cname If `check-integrity` is set, then fail, warn, or ignore SRV records that refer to CNAMES. The default is to `warn`.

check-sibling When performing integrity checks, also check that sibling glue exists. The default is `yes`.

check-spf If `check-integrity` is set, check that there is a TXT Sender Policy Framework record present (starts with "v=spf1") if there is an SPF record present. The default is `warn`.

zero-no-soa-ttl If `yes`, when returning authoritative negative responses to SOA queries, set the TTL of the SOA record returned in the authority section to zero. The default is `yes`.

zero-no-soa-ttl-cache If `yes`, when caching a negative response to an SOA query set the TTL to zero. The default is `no`.

update-check-ksk When set to the default value of `yes`, check the KSK bit in each key to determine how the key should be used when generating RRSIGs for a secure zone.

Ordinarily, zone-signing keys (that is, keys without the KSK bit set) are used to sign the entire zone, while key-signing keys (keys with the KSK bit set) are only used to sign the DNSKEY RRset at the zone apex. However, if

this option is set to `no`, then the KSK bit is ignored; KSKs are treated as if they were ZSKs and are used to sign the entire zone. This is similar to the `dnssec-signzone -z` command-line option.

When this option is set to `yes`, there must be at least two active keys for every algorithm represented in the DNSKEY RRset: at least one KSK and one ZSK per algorithm. If there is any algorithm for which this requirement is not met, this option is ignored for that algorithm.

dnssec-dnskey-kskonly When this option and `update-check-ksk` are both set to `yes`, only key-signing keys (that is, keys with the KSK bit set) are used to sign the DNSKEY, CDNSKEY, and CDS RRsets at the zone apex. Zone-signing keys (keys without the KSK bit set) are used to sign the remainder of the zone, but not the DNSKEY RRset. This is similar to the `dnssec-signzone -x` command-line option.

The default is `no`. If `update-check-ksk` is set to `no`, this option is ignored.

try-tcp-refresh If `yes`, try to refresh the zone using TCP if UDP queries fail. The default is `yes`.

dnssec-secure-to-insecure This allows a dynamic zone to transition from secure to insecure (i.e., signed to unsigned) by deleting all of the DNSKEY records. The default is `no`. If set to `yes`, and if the DNSKEY RRset at the zone apex is deleted, all RRSIG and NSEC records are removed from the zone as well.

If the zone uses NSEC3, it is also necessary to delete the NSEC3PARAM RRset from the zone apex; this causes the removal of all corresponding NSEC3 records. (It is expected that this requirement will be eliminated in a future release.)

Note that if a zone has been configured with `auto-dnssec maintain` and the private keys remain accessible in the key repository, the zone will be automatically signed again the next time `named` is started.

synth-from-dnssec This option synthesizes answers from cached NSEC, NSEC3, and other RRsets that have been proved to be correct using DNSSEC. The default is `no`, but it will become `yes` again in future releases.

Note: DNSSEC validation must be enabled for this option to be effective. This initial implementation only covers synthesis of answers from NSEC records; synthesis from NSEC3 is planned for the future. This will also be controlled by `synth-from-dnssec`.

Forwarding

The forwarding facility can be used to create a large site-wide cache on a few servers, reducing traffic over links to external name servers. It can also be used to allow queries by servers that do not have direct access to the Internet, but wish to look up exterior names anyway. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.

forward This option is only meaningful if the forwarders list is not empty. A value of `first` is the default and causes the server to query the forwarders first; if that does not answer the question, the server then looks for the answer itself. If `only` is specified, the server only queries the forwarders.

forwarders This specifies a list of IP addresses to which queries are forwarded. The default is the empty list (no forwarding). Each address in the list can be associated with an optional port number and/or DSCP value, and a default port number and DSCP value can be set for the entire list.

Forwarding can also be configured on a per-domain basis, allowing for the global forwarding options to be overridden in a variety of ways. Particular domains can be set to use different forwarders, or have a different `forward only/first` behavior, or not forward at all; see *zone Statement Grammar*.

Dual-stack Servers

Dual-stack servers are used as servers of last resort, to work around problems in reachability due to the lack of support for either IPv4 or IPv6 on the host machine.

dual-stack-servers This specifies host names or addresses of machines with access to both IPv4 and IPv6 transports. If a hostname is used, the server must be able to resolve the name using only the transport it has. If the machine is dual-stacked, the `dual-stack-servers` parameter has no effect unless access to a transport has been disabled on the command line (e.g., `named -4`).

Access Control

Access to the server can be restricted based on the IP address of the requesting system. See *Address Match Lists* for details on how to specify IP address lists.

allow-notify This ACL specifies which hosts may send NOTIFY messages to inform this server of changes to zones for which it is acting as a secondary server. This is only applicable for secondary zones (i.e., type `secondary` or `slave`).

If this option is set in `view` or `options`, it is globally applied to all secondary zones. If set in the `zone` statement, the global value is overridden.

If not specified, the default is to process NOTIFY messages only from the configured `primaries` for the zone. `allow-notify` can be used to expand the list of permitted hosts, not to reduce it.

allow-query This specifies which hosts are allowed to ask ordinary DNS questions. `allow-query` may also be specified in the `zone` statement, in which case it overrides the `options allow-query` statement. If not specified, the default is to allow queries from all hosts.

Note: `allow-query-cache` is used to specify access to the cache.

---

allow-query-on This specifies which local addresses can accept ordinary DNS questions. This makes it possible, for instance, to allow queries on internal-facing interfaces but disallow them on external-facing ones, without necessarily knowing the internal network's addresses.

Note that `allow-query-on` is only checked for queries that are permitted by `allow-query`. A query must be allowed by both ACLs, or it is refused.

`allow-query-on` may also be specified in the `zone` statement, in which case it overrides the `options allow-query-on` statement.

If not specified, the default is to allow queries on all addresses.

Note: `allow-query-cache` is used to specify access to the cache.

---

allow-query-cache This specifies which hosts are allowed to get answers from the cache. If `allow-recursion` is not set, BIND checks to see if the following parameters are set, in order: `allow-query-cache` and `allow-query` (unless `recursion no;` is set). If neither of those parameters is set, the default (localnets; localhost;) is used.

allow-query-cache-on This specifies which local addresses can send answers from the cache. If `allow-query-cache-on` is not set, then `allow-recursion-on` is used if set. Otherwise, the default is to allow cache responses to be sent from any address. Note: both `allow-query-cache` and `allow-query-cache-on` must be satisfied before a cache response can be sent; a client that is blocked by one cannot be allowed by the other.

---

allow-recursion This specifies which hosts are allowed to make recursive queries through this server. BIND checks to see if the following parameters are set, in order: `allow-query-cache` and `allow-query`. If neither of those parameters is set, the default (localnets; localhost;) is used.

allow-recursion-on This specifies which local addresses can accept recursive queries. If `allow-recursion-on` is not set, then `allow-query-cache-on` is used if set; otherwise, the default is to allow recursive queries on all addresses. Any client permitted to send recursive queries can send them to any address on which `named` is listening. Note: both `allow-recursion` and `allow-recursion-on` must be satisfied before recursion is allowed; a client that is blocked by one cannot be allowed by the other.

allow-update When set in the `zone` statement for a primary zone, this specifies which hosts are allowed to submit Dynamic DNS updates to that zone. The default is to deny updates from all hosts.

Note that allowing updates based on the requestor's IP address is insecure; see *Dynamic Update Security* for details.

In general, this option should only be set at the `zone` level. While a default value can be set at the `options` or `view` level and inherited by zones, this could lead to some zones unintentionally allowing updates.

allow-update-forwarding When set in the `zone` statement for a secondary zone, this specifies which hosts are allowed to submit Dynamic DNS updates and have them be forwarded to the primary. The default is `{ none; }`, which means that no update forwarding is performed.

To enable update forwarding, specify `allow-update-forwarding { any; };` in the `zone` statement. Specifying values other than `{ none; }` or `{ any; }` is usually counterproductive; the responsibility for update access control should rest with the primary server, not the secondary.

Note that enabling the update forwarding feature on a secondary server may expose primary servers to attacks if they rely on insecure IP-address-based access control; see *Dynamic Update Security* for more details.

In general this option should only be set at the `zone` level. While a default value can be set at the `options` or `view` level and inherited by zones, this can lead to some zones unintentionally forwarding updates.

allow-transfer This specifies which hosts are allowed to receive zone transfers from the server. `allow-transfer` may also be specified in the `zone` statement, in which case it overrides the `allow-transfer` statement set in `options` or `view`. If not specified, the default is to allow transfers to all hosts.

blackhole This specifies a list of addresses which the server does not accept queries from or use to resolve a query. Queries from these addresses are not responded to. The default is `none`.

keep-response-order This specifies a list of addresses to which the server sends responses to TCP queries, in the same order in which they were received. This disables the processing of TCP queries in parallel. The default is `none`.

no-case-compress This specifies a list of addresses which require responses to use case-insensitive compression. This ACL can be used when `named` needs to work with clients that do not comply with the requirement in **RFC 1034** to use case-insensitive name comparisons when checking for matching domain names.

If left undefined, the ACL defaults to `none`: case-insensitive compression is used for all clients. If the ACL is defined and matches a client, case is ignored when compressing domain names in DNS responses sent to that client.

This can result in slightly smaller responses; if a response contains the names "example.com" and "example.COM", case-insensitive compression treats the second one as a duplicate. It also ensures that the case of the query name exactly matches the case of the owner names of returned records, rather than matches the case of the records entered in the zone file. This allows responses to exactly match the query, which is required by some clients due to incorrect use of case-sensitive comparisons.

Case-insensitive compression is *always* used in AXFR and IXFR responses, regardless of whether the client matches this ACL.

There are circumstances in which `named` does not preserve the case of owner names of records: if a zone file defines records of different types with the same name, but the capitalization of the name is different (e.g.,

"www.example.com/A" and "WWW.EXAMPLE.COM/AAAA"), then all responses for that name use the *first* version of the name that was used in the zone file. This limitation may be addressed in a future release. However, domain names specified in the rdata of resource records (i.e., records of type NS, MX, CNAME, etc.) always have their case preserved unless the client matches this ACL.

resolver-query-timeout This is the amount of time in milliseconds that the resolver spends attempting to resolve a recursive query before failing. The default and minimum is `10000` and the maximum is `30000`. Setting it to `0` results in the default being used.

This value was originally specified in seconds. Values less than or equal to 300 are treated as seconds and converted to milliseconds before applying the above limits.

Interfaces

The interfaces and ports that the server answers queries from may be specified using the `listen-on` option. `listen-on` takes an optional port and an `address_match_list` of IPv4 addresses. (IPv6 addresses are ignored, with a logged warning.) The server listens on all interfaces allowed by the address match list. If a port is not specified, port 53 is used.

Multiple `listen-on` statements are allowed. For example:

```
listen-on { 5.6.7.8; };
listen-on port 1234 { !1.2.3.4; 1.2/16; };
```

enables the name server on port 53 for the IP address 5.6.7.8, and on port 1234 of an address on the machine in net 1.2 that is not 1.2.3.4.

If no `listen-on` is specified, the server listens on port 53 on all IPv4 interfaces.

The `listen-on-v6` option is used to specify the interfaces and the ports on which the server listens for incoming queries sent using IPv6. If not specified, the server listens on port 53 on all IPv6 interfaces.

Multiple `listen-on-v6` options can be used. For example:

```
listen-on-v6 { any; };
listen-on-v6 port 1234 { !2001:db8::/32; any; };
```

enables the name server on port 53 for any IPv6 addresses (with a single wildcard socket), and on port 1234 of IPv6 addresses that are not in the prefix 2001:db8::/32 (with separate sockets for each matched address).

To instruct the server not to listen on any IPv6 address, use:

```
listen-on-v6 { none; };
```

Query Address

If the server does not know the answer to a question, it queries other name servers. `query-source` specifies the address and port used for such queries. For queries sent over IPv6, there is a separate `query-source-v6` option. If `address` is * (asterisk) or is omitted, a wildcard IP address (`INADDR_ANY`) is used.

If `port` is * or is omitted, a random port number from a pre-configured range is picked up and used for each query. The port range(s) is specified in the `use-v4-udp-ports` (for IPv4) and `use-v6-udp-ports` (for IPv6) options, excluding the ranges specified in the `avoid-v4-udp-ports` and `avoid-v6-udp-ports` options, respectively.

The defaults of the `query-source` and `query-source-v6` options are:

```
query-source address * port *;
query-source-v6 address * port *;
```

If `use-v4-udp-ports` or `use-v6-udp-ports` is unspecified, `named` checks whether the operating system provides a programming interface to retrieve the system's default range for ephemeral ports. If such an interface is available, `named` uses the corresponding system default range; otherwise, it uses its own defaults:

```
use-v4-udp-ports { range 1024 65535; };
use-v6-udp-ports { range 1024 65535; };
```

Note: Make sure the ranges are sufficiently large for security. A desirable size depends on several parameters, but we generally recommend it contain at least 16384 ports (14 bits of entropy). Note also that the system's default range when used may be too small for this purpose, and that the range may even be changed while `named` is running; the new range is automatically applied when `named` is reloaded. Explicit configuration of `use-v4-udp-ports` and `use-v6-udp-ports` is encouraged, so that the ranges are sufficiently large and are reasonably independent from the ranges used by other applications.

Note: The operational configuration where `named` runs may prohibit the use of some ports. For example, Unix systems do not allow `named`, if run without root privilege, to use ports less than 1024. If such ports are included in the specified (or detected) set of query ports, the corresponding query attempts will fail, resulting in resolution failures or delay. It is therefore important to configure the set of ports that can be safely used in the expected operational environment.

The defaults of the `avoid-v4-udp-ports` and `avoid-v6-udp-ports` options are:

```
avoid-v4-udp-ports {};
avoid-v6-udp-ports {};
```

Note: BIND 9.5.0 introduced the `use-queryport-pool` option to support a pool of such random ports, but this option is now obsolete because reusing the same ports in the pool may not be sufficiently secure. For the same reason, it is generally strongly discouraged to specify a particular port for the `query-source` or `query-source-v6` options; it implicitly disables the use of randomized port numbers.

---

use-queryport-pool This option is obsolete.

queryport-pool-ports This option is obsolete.

queryport-pool-updateinterval This option is obsolete.

Note: The address specified in the `query-source` option is used for both UDP and TCP queries, but the port applies only to UDP queries. TCP queries always use a random unprivileged port.

Note: Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

Warning: Specifying a single port is discouraged, as it removes a layer of protection against spoofing errors.

> **Warning:** The configured `port` must not be same as the listening port.

Note: See also `transfer-source`, `notify-source` and `parental-source`.

Zone Transfers

BIND has mechanisms in place to facilitate zone transfers and set limits on the amount of load that transfers place on the system. The following options apply to zone transfers.

`also-notify` This option defines a global list of IP addresses of name servers that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded, in addition to the servers listed in the zone's NS records. This helps to ensure that copies of the zones quickly converge on stealth servers. Optionally, a port may be specified with each `also-notify` address to send the notify messages to a port other than the default of 53. An optional TSIG key can also be specified with each address to cause the notify messages to be signed; this can be useful when sending notifies to multiple views. In place of explicit addresses, one or more named `primaries` lists can be used.

If an `also-notify` list is given in a `zone` statement, it overrides the `options also-notify` statement. When a `zone notify` statement is set to `no`, the IP addresses in the global `also-notify` list are not sent NOTIFY messages for that zone. The default is the empty list (no global notification list).

`max-transfer-time-in` Inbound zone transfers running longer than this many minutes are terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

`max-transfer-idle-in` Inbound zone transfers making no progress in this many minutes are terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

`max-transfer-time-out` Outbound zone transfers running longer than this many minutes are terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

`max-transfer-idle-out` Outbound zone transfers making no progress in this many minutes are terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

`notify-rate` This specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations. (NOTIFY requests due to initial zone loading are subject to a separate rate limit; see below.) The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

`startup-notify-rate` This is the rate at which NOTIFY requests are sent when the name server is first starting up, or when zones have been newly added to the name server. The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

`serial-query-rate` Secondary servers periodically query primary servers to find out if zone serial numbers have changed. Each such query uses a minute amount of the secondary server's network bandwidth. To limit the amount of bandwidth used, BIND 9 limits the rate at which queries are sent. The value of the `serial-query-rate` option, an integer, is the maximum number of queries sent per second. The default is 20 per second. The lowest possible rate is one per second; when set to zero, it is silently raised to one.

`transfer-format` Zone transfers can be sent using two different formats, `one-answer` and `many-answers`. The `transfer-format` option is used on the primary server to determine which format it sends. `one-answer` uses one DNS message per resource record transferred. `many-answers` packs as many resource records as possible into one message. `many-answers` is more efficient; the default is `many-answers`. `transfer-format` may be overridden on a per-server basis by using the `server` statement.

`transfer-message-size` This is an upper bound on the uncompressed size of DNS messages used in zone transfers over TCP. If a message grows larger than this size, additional messages are used to complete the zone transfer.

(Note, however, that this is a hint, not a hard limit; if a message contains a single resource record whose RDATA does not fit within the size limit, a larger message will be permitted so the record can be transferred.)

Valid values are between 512 and 65535 octets; any values outside that range are adjusted to the nearest value within it. The default is `20480`, which was selected to improve message compression; most DNS messages of this size will compress to less than 16536 bytes. Larger messages cannot be compressed as effectively, because 16536 is the largest permissible compression offset pointer in a DNS message.

This option is mainly intended for server testing; there is rarely any benefit in setting a value other than the default.

transfers-in This is the maximum number of inbound zone transfers that can run concurrently. The default value is `10`. Increasing `transfers-in` may speed up the convergence of secondary zones, but it also may increase the load on the local system.

transfers-out This is the maximum number of outbound zone transfers that can run concurrently. Zone transfer requests in excess of the limit are refused. The default value is `10`.

transfers-per-ns This is the maximum number of inbound zone transfers that can concurrently transfer from a given remote name server. The default value is `2`. Increasing `transfers-per-ns` may speed up the convergence of secondary zones, but it also may increase the load on the remote name server. `transfers-per-ns` may be overridden on a per-server basis by using the `transfers` phrase of the `server` statement.

transfer-source `transfer-source` determines which local address is bound to IPv4 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv4 address, and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. If not set, it defaults to a system-controlled value which is usually the address of the interface "closest to" the remote end. This address must appear in the remote end's `allow-transfer` option for the zone being transferred, if one is specified. This statement sets the `transfer-source` for all zones, but can be overridden on a per-view or per-zone basis by including a `transfer-source` statement within the `view` or `zone` block in the configuration file.

Note: Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

> **Warning:** Specifying a single port is discouraged, as it removes a layer of protection against spoofing errors.

> **Warning:** The configured `port` must not be same as the listening port.

transfer-source-v6 This option is the same as `transfer-source`, except zone transfers are performed using IPv6.

alt-transfer-source This indicates an alternate transfer source if the one listed in `transfer-source` fails and `use-alt-transfer-source` is set.

Note: To avoid using the alternate transfer source, set `use-alt-transfer-source` appropriately and do not depend upon getting an answer back to the first refresh query.

---

alt-transfer-source-v6 This indicates an alternate transfer source if the one listed in `transfer-source-v6` fails and `use-alt-transfer-source` is set.

use-alt-transfer-source This indicates whether the alternate transfer sources should be used. If views are specified, this defaults to `no`; otherwise, it defaults to `yes`.

notify-source `notify-source` determines which local source address, and optionally UDP port, is used to send NOTIFY messages. This address must appear in the secondary server's `primaries` zone clause or in an

---

allow-notify clause. This statement sets the notify-source for all zones, but can be overridden on a per-zone or per-view basis by including a notify-source statement within the zone or view block in the configuration file.

Note: Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

> **Warning:** Specifying a single port is discouraged, as it removes a layer of protection against spoofing errors.

> **Warning:** The configured port must not be same as the listening port.

notify-source-v6 This option acts like notify-source, but applies to notify messages sent to IPv6 addresses.

UDP Port Lists

use-v4-udp-ports, avoid-v4-udp-ports, use-v6-udp-ports, and avoid-v6-udp-ports specify a list of IPv4 and IPv6 UDP ports that are or are not used as source ports for UDP messages. See *Query Address* about how the available ports are determined. For example, with the following configuration:

```
use-v6-udp-ports { range 32768 65535; };
avoid-v6-udp-ports { 40000; range 50000 60000; };
```

UDP ports of IPv6 messages sent from named are in one of the following ranges: 32768 to 39999, 40001 to 49999, and 60001 to 65535.

avoid-v4-udp-ports and avoid-v6-udp-ports can be used to prevent named from choosing as its random source port a port that is blocked by a firewall or a port that is used by other applications; if a query went out with a source port blocked by a firewall, the answer would not pass through the firewall and the name server would have to query again. Note: the desired range can also be represented only with use-v4-udp-ports and use-v6-udp-ports, and the avoid- options are redundant in that sense; they are provided for backward compatibility and to possibly simplify the port specification.

Operating System Resource Limits

The server's usage of many system resources can be limited. Scaled values are allowed when specifying resource limits. For example, 1G can be used instead of 1073741824 to specify a limit of one gigabyte. unlimited requests unlimited use, or the maximum available amount. default uses the limit that was in force when the server was started. See the description of size_spec in *Configuration File Elements*.

The following options set operating system resource limits for the name server process. Some operating systems do not support some or any of the limits; on such systems, a warning is issued if an unsupported limit is used.

coresize This sets the maximum size of a core dump. The default is default.

datasize This sets the maximum amount of data memory the server may use. The default is default. This is a hard limit on server memory usage; if the server attempts to allocate memory in excess of this limit, the allocation will fail, which may in turn leave the server unable to perform DNS service. Therefore, this option is rarely useful as a way to limit the amount of memory used by the server, but it can be used to raise an operating system data size limit that is too small by default. To limit the amount of memory used by the server, use the max-cache-size and recursive-clients options instead.

files This sets the maximum number of files the server may have open concurrently. The default is unlimited.

---

stacksize This sets the maximum amount of stack memory the server may use. The default is `default`.

Server Resource Limits

The following options set limits on the server's resource consumption that are enforced internally by the server rather than by the operating system.

max-journal-size This sets a maximum size for each journal file (see *The Journal File*), expressed in bytes or, if followed by an optional unit suffix ('k', 'm', or 'g'), in kilobytes, megabytes, or gigabytes. When the journal file approaches the specified size, some of the oldest transactions in the journal are automatically removed. The largest permitted value is 2 gigabytes. Very small values are rounded up to 4096 bytes. It is possible to specify `unlimited`, which also means 2 gigabytes. If the limit is set to `default` or left unset, the journal is allowed to grow up to twice as large as the zone. (There is little benefit in storing larger journals.)

This option may also be set on a per-zone basis.

max-records This sets the maximum number of records permitted in a zone. The default is zero, which means the maximum is unlimited.

recursive-clients This sets the maximum number (a "hard quota") of simultaneous recursive lookups the server performs on behalf of clients. The default is `1000`. Because each recursing client uses a fair bit of memory (on the order of 20 kilobytes), the value of the `recursive-clients` option may have to be decreased on hosts with limited memory.

`recursive-clients` defines a "hard quota" limit for pending recursive clients; when more clients than this are pending, new incoming requests are not accepted, and for each incoming request a previous pending request is dropped.

A "soft quota" is also set. When this lower quota is exceeded, incoming requests are accepted, but for each one, a pending request is dropped. If `recursive-clients` is greater than 1000, the soft quota is set to `recursive-clients` minus 100; otherwise it is set to 90% of `recursive-clients`.

tcp-clients This is the maximum number of simultaneous client TCP connections that the server accepts. The default is `150`.

clients-per-query; max-clients-per-query These set the initial value (minimum) and maximum number of recursive simultaneous clients for any given query (<qname,qtype,qclass>) that the server accepts before dropping additional clients. `named` attempts to self-tune this value and changes are logged. The default values are 10 and 100.

This value should reflect how many queries come in for a given name in the time it takes to resolve that name. If the number of queries exceeds this value, `named` assumes that it is dealing with a non-responsive zone and drops additional queries. If it gets a response after dropping queries, it raises the estimate. The estimate is then lowered in 20 minutes if it has remained unchanged.

If `clients-per-query` is set to zero, there is no limit on the number of clients per query and no queries are dropped.

If `max-clients-per-query` is set to zero, there is no upper bound other than that imposed by `recursive-clients`.

fetches-per-zone This sets the maximum number of simultaneous iterative queries to any one domain that the server permits before blocking new queries for data in or beneath that zone. This value should reflect how many fetches would normally be sent to any one zone in the time it would take to resolve them. It should be smaller than `recursive-clients`.

When many clients simultaneously query for the same name and type, the clients are all attached to the same fetch, up to the `max-clients-per-query` limit, and only one iterative query is sent. However, when clients are simultaneously querying for *different* names or types, multiple queries are sent and `max-clients-per-query` is not effective as a limit.

Optionally, this value may be followed by the keyword `drop` or `fail`, indicating whether queries which exceed the fetch quota for a zone are dropped with no response, or answered with SERVFAIL. The default is `drop`.

If `fetches-per-zone` is set to zero, there is no limit on the number of fetches per query and no queries are dropped. The default is zero.

The current list of active fetches can be dumped by running `rndc recursing`. The list includes the number of active fetches for each domain and the number of queries that have been passed (allowed) or dropped (spilled) as a result of the `fetches-per-zone` limit. (Note: these counters are not cumulative over time; whenever the number of active fetches for a domain drops to zero, the counter for that domain is deleted, and the next time a fetch is sent to that domain, it is recreated with the counters set to zero.)

fetches-per-server This sets the maximum number of simultaneous iterative queries that the server allows to be sent to a single upstream name server before blocking additional queries. This value should reflect how many fetches would normally be sent to any one server in the time it would take to resolve them. It should be smaller than `recursive-clients`.

Optionally, this value may be followed by the keyword `drop` or `fail`, indicating whether queries are dropped with no response or answered with SERVFAIL, when all of the servers authoritative for a zone are found to have exceeded the per-server quota. The default is `fail`.

If `fetches-per-server` is set to zero, there is no limit on the number of fetches per query and no queries are dropped. The default is zero.

The `fetches-per-server` quota is dynamically adjusted in response to detected congestion. As queries are sent to a server and either are answered or time out, an exponentially weighted moving average is calculated of the ratio of timeouts to responses. If the current average timeout ratio rises above a "high" threshold, then `fetches-per-server` is reduced for that server. If the timeout ratio drops below a "low" threshold, then `fetches-per-server` is increased. The `fetch-quota-params` options can be used to adjust the parameters for this calculation.

fetch-quota-params This sets the parameters to use for dynamic resizing of the `fetches-per-server` quota in response to detected congestion.

The first argument is an integer value indicating how frequently to recalculate the moving average of the ratio of timeouts to responses for each server. The default is 100, meaning that BIND recalculates the average ratio after every 100 queries have either been answered or timed out.

The remaining three arguments represent the "low" threshold (defaulting to a timeout ratio of 0.1), the "high" threshold (defaulting to a timeout ratio of 0.3), and the discount rate for the moving average (defaulting to 0.7). A higher discount rate causes recent events to weigh more heavily when calculating the moving average; a lower discount rate causes past events to weigh more heavily, smoothing out short-term blips in the timeout ratio. These arguments are all fixed-point numbers with precision of 1/100; at most two places after the decimal point are significant.

reserved-sockets This sets the number of file descriptors reserved for TCP, stdio, etc. This needs to be big enough to cover the number of interfaces `named` listens on plus `tcp-clients`, as well as to provide room for outgoing TCP queries and incoming zone transfers. The default is `512`. The minimum value is `128` and the maximum value is `128` fewer than maxsockets (-S). This option may be removed in the future.

This option has little effect on Windows.

max-cache-size This sets the maximum amount of memory to use for an individual cache database and its associated metadata, in bytes or percentage of total physical memory. By default, each view has its own separate cache, which means the total amount of memory required for cache data is the sum of the cache database sizes for all views (unless the *attach-cache* option is used).

When the amount of data in a cache database reaches the configured limit, `named` starts purging non-expired records (following an LRU-based strategy).

The default size limit for each individual cache is:

- 90% of physical memory for views with `recursion` set to `yes` (the default), or

- 2 MB for views with `recursion` set to `no`.

Any positive value smaller than 2 MB is ignored and reset to 2 MB. The keyword `unlimited`, or the value `0`, places no limit on the cache size; records are then purged from the cache only when they expire (according to their TTLs).

Note: For configurations which define multiple views with separate caches and recursion enabled, it is recommended to set `max-cache-size` appropriately for each view, as using the default value of that option (90% of physical memory for each individual cache) may lead to memory exhaustion over time.

Upon startup and reconfiguration, caches with a limited size preallocate a small amount of memory (less than 1% of `max-cache-size` for a given view). This preallocation serves as an optimization to eliminate extra latency introduced by resizing internal cache structures.

On systems where detection of the amount of physical memory is not supported, percentage-based values fall back to `unlimited`. Note that the amount of physical memory available is only detected on startup, so `named` does not adjust the cache size limits if the amount of physical memory is changed at runtime.

tcp-listen-queue This sets the listen-queue depth. The default and minimum is 10. If the kernel supports the accept filter "dataready", this also controls how many TCP connections are queued in kernel space waiting for some data before being passed to accept. Non-zero values less than 10 are silently raised. A value of 0 may also be used; on most platforms this sets the listen-queue length to a system-defined default value.

tcp-initial-timeout This sets the amount of time (in units of 100 milliseconds) that the server waits on a new TCP connection for the first message from the client. The default is 300 (30 seconds), the minimum is 25 (2.5 seconds), and the maximum is 1200 (two minutes). Values above the maximum or below the minimum are adjusted with a logged warning. (Note: this value must be greater than the expected round-trip delay time; otherwise, no client will ever have enough time to submit a message.) This value can be updated at runtime by using `rndc tcp-timeouts`.

tcp-idle-timeout This sets the amount of time (in units of 100 milliseconds) that the server waits on an idle TCP connection before closing it, when the client is not using the EDNS TCP keepalive option. The default is 300 (30 seconds), the maximum is 1200 (two minutes), and the minimum is 1 (one-tenth of a second). Values above the maximum or below the minimum are adjusted with a logged warning. See `tcp-keepalive-timeout` for clients using the EDNS TCP keepalive option. This value can be updated at runtime by using `rndc tcp-timeouts`.

tcp-keepalive-timeout This sets the amount of time (in units of 100 milliseconds) that the server waits on an idle TCP connection before closing it, when the client is using the EDNS TCP keepalive option. The default is 300 (30 seconds), the maximum is 65535 (about 1.8 hours), and the minimum is 1 (one-tenth of a second). Values above the maximum or below the minimum are adjusted with a logged warning. This value may be greater than `tcp-idle-timeout` because clients using the EDNS TCP keepalive option are expected to use TCP connections for more than one message. This value can be updated at runtime by using `rndc tcp-timeouts`.

tcp-advertised-timeout This sets the timeout value (in units of 100 milliseconds) that the server sends in responses containing the EDNS TCP keepalive option, which informs a client of the amount of time it may keep the session open. The default is 300 (30 seconds), the maximum is 65535 (about 1.8 hours), and the minimum is 0, which signals that the clients must close TCP connections immediately. Ordinarily this should be set to the same value as `tcp-keepalive-timeout`. This value can be updated at runtime by using `rndc tcp-timeouts`.

Periodic Task Intervals

cleaning-interval This option is obsolete.

heartbeat-interval The server performs zone maintenance tasks for all zones marked as dialup whenever this interval expires. The default is 60 minutes. Reasonable values are up to 1 day (1440 minutes). The maximum value is 28 days (40320 minutes). If set to 0, no zone maintenance for these zones occurs.

interface-interval The server scans the network interface list every interface-interval minutes. The default is 60 minutes; the maximum value is 28 days (40320 minutes). If set to 0, interface scanning only occurs when the configuration file is loaded, or when automatic-interface-scan is enabled and supported by the operating system. After the scan, the server begins listening for queries on any newly discovered interfaces (provided they are allowed by the listen-on configuration), and stops listening on interfaces that have gone away. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The sortlist Statement

The response to a DNS query may consist of multiple resource records (RRs) forming a resource record set (RRset). The name server normally returns the RRs within the RRset in an indeterminate order (but see the rrset-order statement in *RRset Ordering*). The client resolver code should rearrange the RRs as appropriate: that is, using any addresses on the local net in preference to other addresses. However, not all resolvers can do this or are correctly configured. When a client is using a local server, the sorting can be performed in the server, based on the client's address. This only requires configuring the name servers, not all the clients.

The sortlist statement (see below) takes an address_match_list and interprets it in a special way. Each top-level statement in the sortlist must itself be an explicit address_match_list with one or two elements. The first element (which may be an IP address, an IP prefix, an ACL name, or a nested address_match_list) of each top-level list is checked against the source address of the query until a match is found. When the addresses in the first element overlap, the first rule to match is selected.

Once the source address of the query has been matched, if the top-level statement contains only one element, the actual primitive element that matched the source address is used to select the address in the response to move to the beginning of the response. If the statement is a list of two elements, then the second element is interpreted as a topology preference list. Each top-level element is assigned a distance, and the address in the response with the minimum distance is moved to the beginning of the response.

In the following example, any queries received from any of the addresses of the host itself get responses preferring addresses on any of the locally connected networks. Next most preferred are addresses on the 192.168.1/24 network, and after that either the 192.168.2/24 or 192.168.3/24 network, with no preference shown between these two networks. Queries received from a host on the 192.168.1/24 network prefer other addresses on that network to the 192.168.2/24 and 192.168.3/24 networks. Queries received from a host on the 192.168.4/24 or the 192.168.5/24 network only prefer other addresses on their directly connected networks.

```
sortlist {
    // IF the local host
    // THEN first fit on the following nets
    { localhost;
    { localnets;
        192.168.1/24;
        { 192.168.2/24; 192.168.3/24; }; }; };
    // IF on class C 192.168.1 THEN use .1, or .2 or .3
    { 192.168.1/24;
    { 192.168.1/24;
        { 192.168.2/24; 192.168.3/24; }; }; };
    // IF on class C 192.168.2 THEN use .2, or .1 or .3
```

(continues on next page)

```
        { 192.168.2/24;
        { 192.168.2/24;
            { 192.168.1/24; 192.168.3/24; }; }; };
        // IF on class C 192.168.3 THEN use .3, or .1 or .2
        { 192.168.3/24;
        { 192.168.3/24;
            { 192.168.1/24; 192.168.2/24; }; }; };
        // IF .4 or .5 THEN prefer that net
        { { 192.168.4/24; 192.168.5/24; };
        };
};
```

The following example illllustrates reasonable behavior for the local host and hosts on directly connected networks. Responses sent to queries from the local host favor any of the directly connected networks. Responses sent to queries from any other hosts on a directly connected network prefer addresses on that same network. Responses to other queries are not sorted.

```
sortlist {
        { localhost; localnets; };
        { localnets; };
};
```

RRset Ordering

Note: While alternating the order of records in a DNS response between subsequent queries is a known load distribution technique, certain caveats apply (mostly stemming from caching) which usually make it a suboptimal choice for load balancing purposes when used on its own.

The `rrset-order` statement permits configuration of the ordering of the records in a multiple-record response. See also: *The sortlist Statement*.

Each rule in an `rrset-order` statement is defined as follows:

```
[class <class_name>] [type <type_name>] [name "<domain_name>"] order <ordering>
```

The default qualifiers for each rule are:

- If no `class` is specified, the default is `ANY`.

- If no `type` is specified, the default is `ANY`.

- If no `name` is specified, the default is `*` (asterisk).

`<domain_name>` only matches the name itself, not any of its subdomains. To make a rule match all subdomains of a given name, a wildcard name (`*.<domain_name>`) must be used. Note that `*.<domain_name>` does *not* match `<domain_name>` itself; to specify RRset ordering for a name and all of its subdomains, two separate rules must be defined: one for `<domain_name>` and one for `*.<domain_name>`.

The legal values for `<ordering>` are:

fixed Records are returned in the order they are defined in the zone file.

Note: The `fixed` option is only available if BIND is configured with `--enable-fixed-rrset` at compile time.

random Records are returned in a random order.

cyclic Records are returned in a cyclic round-robin order, rotating by one record per query.

none Records are returned in the order they were retrieved from the database. This order is indeterminate, but remains consistent as long as the database is not modified.

The default RRset order used depends on whether any `rrset-order` statements are present in the configuration file used by `named`:

- If no `rrset-order` statement is present in the configuration file, the implicit default is to return all records in `random` order.

- If any `rrset-order` statements are present in the configuration file, but no ordering rule specified in these statements matches a given RRset, the default order for that RRset is `none`.

Note that if multiple `rrset-order` statements are present in the configuration file (at both the `options` and `view` levels), they are *not* combined; instead, the more-specific one (`view`) replaces the less-specific one (`options`).

If multiple rules within a single `rrset-order` statement match a given RRset, the first matching rule is applied.

Example:

```
rrset-order {
    type A name "foo.isc.org" order random;
    type AAAA name "foo.isc.org" order cyclic;
    name "bar.isc.org" order fixed;
    name "*.bar.isc.org" order random;
    name "*.baz.isc.org" order cyclic;
};
```

With the above configuration, the following RRset ordering is used:

QNAME	QTYPE	RRset Order
`foo.isc.org`	`A`	`random`
`foo.isc.org`	`AAAA`	`cyclic`
`foo.isc.org`	`TXT`	`none`
`sub.foo.isc.org`	all	`none`
`bar.isc.org`	all	`fixed`
`sub.bar.isc.org`	all	`random`
`baz.isc.org`	all	`none`
`sub.baz.isc.org`	all	`cyclic`

Tuning

lame-ttl This is always set to 0. More information is available in the security advisory for CVE-2021-25219.

servfail-ttl This sets the number of seconds to cache a SERVFAIL response due to DNSSEC validation failure or other general server failure. If set to `0`, SERVFAIL caching is disabled. The SERVFAIL cache is not consulted if a query has the CD (Checking Disabled) bit set; this allows a query that failed due to DNSSEC validation to be retried without waiting for the SERVFAIL TTL to expire.

The maximum value is `30` seconds; any higher value is silently reduced. The default is `1` second.

min-ncache-ttl To reduce network traffic and increase performance, the server stores negative answers. `min-ncache-ttl` is used to set a minimum retention time for these answers in the server, in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The default `min-ncache-ttl` is `0` seconds. `min-ncache-ttl` cannot exceed 90 seconds and is truncated to 90 seconds if set to a greater value.

min-cache-ttl This sets the minimum time for which the server caches ordinary (positive) answers, in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The default `min-cache-ttl` is `0` seconds. `min-cache-ttl` cannot exceed 90 seconds and is truncated to 90 seconds if set to a greater value.

max-ncache-ttl To reduce network traffic and increase performance, the server stores negative answers. `max-ncache-ttl` is used to set a maximum retention time for these answers in the server, in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The default `max-ncache-ttl` is 10800 seconds (3 hours). `max-ncache-ttl` cannot exceed 7 days and is silently truncated to 7 days if set to a greater value.

max-cache-ttl This sets the maximum time for which the server caches ordinary (positive) answers, in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

The default `max-cache-ttl` is 604800 (one week). A value of zero may cause all queries to return SERVFAIL, because of lost caches of intermediate RRsets (such as NS and glue AAAA/A records) in the resolution process.

max-stale-ttl If retaining stale RRsets in cache is enabled, and returning of stale cached answers is also enabled, `max-stale-ttl` sets the maximum time for which the server retains records past their normal expiry to return them as stale records, when the servers for those records are not reachable. The default is 1 day. The minimum allowed is 1 second; a value of 0 is updated silently to 1 second.

For stale answers to be returned, the retaining of them in cache must be enabled via the configuration option `stale-cache-enable`, and returning cached answers must be enabled, either in the configuration file using the `stale-answer-enable` option or by calling `rndc serve-stale on`.

When `stale-cache-enable` is set to `no`, setting the `max-stale-ttl` has no effect, the value of `max-cache-ttl` will be 0 in such case.

resolver-nonbackoff-tries This specifies how many retries occur before exponential backoff kicks in. The default is `3`.

resolver-retry-interval This sets the base retry interval in milliseconds. The default is `800`.

sig-validity-interval this specifies the upper bound of the number of days that RRSIGs generated by `named` are valid; the default is `30` days, with a maximum of 3660 days (10 years). The optional second value specifies the minimum bound on those RRSIGs and also determines how long before expiry `named` starts regenerating those RRSIGs. The default value for the lower bound is 1/4 of the upper bound; it is expressed in days if the upper bound is greater than 7, and hours if it is less than or equal to 7 days.

When new RRSIGs are generated, the length of time is randomly chosen between these two limits, to spread out the re-signing load. When RRSIGs are re-generated, the upper bound is used, with a small amount of jitter added. New RRSIGs are generated by a number of processes, including the processing of UPDATE requests (ref:*dynamic_update*), the addition and removal of records via in-line signing, and the initial signing of a zone.

The signature inception time is unconditionally set to one hour before the current time, to allow for a limited amount of clock skew.

The `sig-validity-interval` can be overridden for DNSKEY records by setting `dnskey-sig-validity`.

The `sig-validity-interval` should be at least several multiples of the SOA expire interval, to allow for reasonable interaction between the various timer and expiry dates.

dnskey-sig-validity This specifies the number of days into the future when DNSSEC signatures that are automatically generated for DNSKEY RRsets as a result of dynamic updates (*Dynamic Update*) will expire. If set to a non-zero value, this overrides the value set by `sig-validity-interval`. The default is zero, meaning `sig-validity-interval` is used. The maximum value is 3660 days (10 years), and higher values are rejected.

sig-signing-nodes This specifies the maximum number of nodes to be examined in each quantum, when signing a zone with a new DNSKEY. The default is `100`.

sig-signing-signatures This specifies a threshold number of signatures that terminates processing a quantum, when signing a zone with a new DNSKEY. The default is `10`.

sig-signing-type This specifies a private RDATA type to be used when generating signing-state records. The default is `65534`.

This parameter may be removed in a future version, once there is a standard type.

Signing-state records are used internally by `named` to track the current state of a zone-signing process, i.e., whether it is still active or has been completed. The records can be inspected using the command `rndc signing -list zone`. Once `named` has finished signing a zone with a particular key, the signing-state record associated with that key can be removed from the zone by running `rndc signing -clear keyid/algorithm zone`. To clear all of the completed signing-state records for a zone, use `rndc signing -clear all zone`.

min-refresh-time; max-refresh-time; min-retry-time; max-retry-time These options control the server's behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, up to a hard-coded maximum expiry of 24 weeks. However, these values are set by the primary, giving secondary server administrators little control over their contents.

These options allow the administrator to set a minimum and maximum refresh and retry time in seconds per-zone, per-view, or globally. These options are valid for secondary and stub zones, and clamp the SOA refresh and retry times to the specified values.

The following defaults apply: `min-refresh-time` 300 seconds, `max-refresh-time` 2419200 seconds (4 weeks), `min-retry-time` 500 seconds, and `max-retry-time` 1209600 seconds (2 weeks).

edns-udp-size This sets the maximum advertised EDNS UDP buffer size, in bytes, to control the size of packets received from authoritative servers in response to recursive queries. Valid values are 512 to 4096; values outside this range are silently adjusted to the nearest value within it. The default value is 1232.

The usual reason for setting `edns-udp-size` to a non-default value is to get UDP answers to pass through broken firewalls that block fragmented packets and/or block UDP DNS packets that are greater than 512 bytes.

When `named` first queries a remote server, it advertises a UDP buffer size of 512, as this has the greatest chance of success on the first try.

If the initial query is successful with EDNS advertising a buffer size of 512, then `named` will advertise progressively larger buffer sizes on successive queries, until responses begin timing out or `edns-udp-size` is reached.

The default buffer sizes used by `named` are 512, 1232, 1432, and 4096, but never exceeding `edns-udp-size`. (The values 1232 and 1432 are chosen to allow for an IPv4-/IPv6-encapsulated UDP message to be sent without fragmentation at the minimum MTU sizes for Ethernet and IPv6 networks.)

The `named` now sets the DON'T FRAGMENT flag on outgoing UDP packets. According to the measurements done by multiple parties this should not be causing any operational problems as most of the Internet "core" is able to cope with IP message sizes between 1400-1500 bytes, the 1232 size was picked as a conservative minimal number that could be changed by the DNS operator to a estimated path MTU minus the estimated header space. In practice, the smallest MTU witnessed in the operational DNS community is 1500 octets, the Ethernet maximum payload size, so a a useful default for maximum DNS/UDP payload size on **reliable** networks would be 1432.

Any server-specific `edns-udp-size` setting has precedence over all the above rules.

max-udp-size This sets the maximum EDNS UDP message size that `named` sends, in bytes. Valid values are 512 to 4096; values outside this range are silently adjusted to the nearest value within it. The default value is 1232.

This value applies to responses sent by a server; to set the advertised buffer size in queries, see `edns-udp-size`.

The usual reason for setting `max-udp-size` to a non-default value is to allow UDP answers to pass through broken firewalls that block fragmented packets and/or block UDP packets that are greater than 512 bytes. This is independent of the advertised receive buffer (`edns-udp-size`).

Setting this to a low value encourages additional TCP traffic to the name server.

masterfile-format This specifies the file format of zone files (see *Additional File Formats* for details). The default value is `text`, which is the standard textual representation, except for secondary zones, in which the default value is `raw`. Files in formats other than `text` are typically expected to be generated by the `named-compilezone` tool, or dumped by `named`.

Note that when a zone file in a format other than `text` is loaded, `named` may omit some of the checks which are performed for a file in `text` format. For example, `check-names` only applies when loading zones in `text` format, and `max-zone-ttl` only applies to `text` and `raw`. Zone files in binary formats should be generated with the same check level as that specified in the `named` configuration file.

`map` format files are loaded directly into memory via memory mapping, with only minimal validity checking. Because they are not guaranteed to be compatible from one version of BIND 9 to another, and are not compatible from one system architecture to another, they should be used with caution. See *Additional File Formats* for further discussion.

When configured in `options`, this statement sets the `masterfile-format` for all zones, but it can be overridden on a per-zone or per-view basis by including a `masterfile-format` statement within the `zone` or `view` block in the configuration file.

masterfile-style This specifies the formatting of zone files during dump, when the `masterfile-format` is `text`. This option is ignored with any other `masterfile-format`.

When set to `relative`, records are printed in a multi-line format, with owner names expressed relative to a shared origin. When set to `full`, records are printed in a single-line format with absolute owner names. The `full` format is most suitable when a zone file needs to be processed automatically by a script. The `relative` format is more human-readable, and is thus suitable when a zone is to be edited by hand. The default is `relative`.

max-recursion-depth This sets the maximum number of levels of recursion that are permitted at any one time while servicing a recursive query. Resolving a name may require looking up a name server address, which in turn requires resolving another name, etc.; if the number of recursions exceeds this value, the recursive query is terminated and returns SERVFAIL. The default is 7.

max-recursion-queries This sets the maximum number of iterative queries that may be sent while servicing a recursive query. If more queries are sent, the recursive query is terminated and returns SERVFAIL. The default is 100.

notify-delay This sets the delay, in seconds, between sending sets of NOTIFY messages for a zone. Whenever a NOTIFY message is sent for a zone, a timer will be set for this duration. If the zone is updated again before the timer expires, the NOTIFY for that update will be postponed. The default is 5 seconds.

The overall rate at which NOTIFY messages are sent for all zones is controlled by `notify-rate`.

max-rsa-exponent-size This sets the maximum RSA exponent size, in bits, that is accepted when validating. Valid values are 35 to 4096 bits. The default, zero, is also accepted and is equivalent to 4096.

prefetch When a query is received for cached data which is to expire shortly, `named` can refresh the data from the authoritative server immediately, ensuring that the cache always has an answer available.

`prefetch` specifies the "trigger" TTL value at which prefetch of the current query takes place; when a cache record with a lower TTL value is encountered during query processing, it is refreshed. Valid trigger TTL values

are 1 to 10 seconds. Values larger than 10 seconds are silently reduced to 10. Setting a trigger TTL to zero causes prefetch to be disabled. The default trigger TTL is `2`.

An optional second argument specifies the "eligibility" TTL: the smallest *original* TTL value that is accepted for a record to be eligible for prefetching. The eligibility TTL must be at least six seconds longer than the trigger TTL; if not, `named` silently adjusts it upward. The default eligibility TTL is `9`.

v6-bias When determining the next name server to try, this indicates by how many milliseconds to prefer IPv6 name servers. The default is `50` milliseconds.

Built-in Server Information Zones

The server provides some helpful diagnostic information through a number of built-in zones under the pseudo-top-level-domain `bind` in the `CHAOS` class. These zones are part of a built-in view (see *view Statement Grammar*) of class `CHAOS`, which is separate from the default view of class `IN`. Most global configuration options (`allow-query`, etc.) apply to this view, but some are locally overridden: `notify`, `recursion`, and `allow-new-zones` are always set to `no`, and `rate-limit` is set to allow three responses per second.

To disable these zones, use the options below or hide the built-in `CHAOS` view by defining an explicit view of class `CHAOS` that matches all clients.

version This is the version the server should report via a query of the name `version.bind` with type `TXT` and class `CHAOS`. The default is the real version number of this server. Specifying `version none` disables processing of the queries.

Setting `version` to any value (including `none`) also disables queries for `authors.bind TXT CH`.

hostname This is the hostname the server should report via a query of the name `hostname.bind` with type `TXT` and class `CHAOS`. This defaults to the hostname of the machine hosting the name server, as found by the `gethostname()` function. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering the queries. Specifying `hostname none;` disables processing of the queries.

server-id This is the ID the server should report when receiving a Name Server Identifier (NSID) query, or a query of the name `ID.SERVER` with type `TXT` and class `CHAOS`. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering the queries. Specifying `server-id none;` disables processing of the queries. Specifying `server-id hostname;` causes `named` to use the hostname as found by the `gethostname()` function. The default `server-id` is `none`.

Built-in Empty Zones

The `named` server has some built-in empty zones, for SOA and NS records only. These are for zones that should normally be answered locally and for which queries should not be sent to the Internet's root servers. The official servers that cover these namespaces return NXDOMAIN responses to these queries. In particular, these cover the reverse namespaces for addresses from **RFC 1918**, **RFC 4193**, **RFC 5737**, and **RFC 6598**. They also include the reverse namespace for the IPv6 local address (locally assigned), IPv6 link local addresses, the IPv6 loopback address, and the IPv6 unknown address.

The server attempts to determine if a built-in zone already exists or is active (covered by a forward-only forwarding declaration) and does not create an empty zone if either is true.

The current list of empty zones is:

- 10.IN-ADDR.ARPA
- 16.172.IN-ADDR.ARPA
- 17.172.IN-ADDR.ARPA
- 18.172.IN-ADDR.ARPA

- 19.172.IN-ADDR.ARPA
- 20.172.IN-ADDR.ARPA
- 21.172.IN-ADDR.ARPA
- 22.172.IN-ADDR.ARPA
- 23.172.IN-ADDR.ARPA
- 24.172.IN-ADDR.ARPA
- 25.172.IN-ADDR.ARPA
- 26.172.IN-ADDR.ARPA
- 27.172.IN-ADDR.ARPA
- 28.172.IN-ADDR.ARPA
- 29.172.IN-ADDR.ARPA
- 30.172.IN-ADDR.ARPA
- 31.172.IN-ADDR.ARPA
- 168.192.IN-ADDR.ARPA
- 64.100.IN-ADDR.ARPA
- 65.100.IN-ADDR.ARPA
- 66.100.IN-ADDR.ARPA
- 67.100.IN-ADDR.ARPA
- 68.100.IN-ADDR.ARPA
- 69.100.IN-ADDR.ARPA
- 70.100.IN-ADDR.ARPA
- 71.100.IN-ADDR.ARPA
- 72.100.IN-ADDR.ARPA
- 73.100.IN-ADDR.ARPA
- 74.100.IN-ADDR.ARPA
- 75.100.IN-ADDR.ARPA
- 76.100.IN-ADDR.ARPA
- 77.100.IN-ADDR.ARPA
- 78.100.IN-ADDR.ARPA
- 79.100.IN-ADDR.ARPA
- 80.100.IN-ADDR.ARPA
- 81.100.IN-ADDR.ARPA
- 82.100.IN-ADDR.ARPA
- 83.100.IN-ADDR.ARPA
- 84.100.IN-ADDR.ARPA
- 85.100.IN-ADDR.ARPA

- 86.100.IN-ADDR.ARPA
- 87.100.IN-ADDR.ARPA
- 88.100.IN-ADDR.ARPA
- 89.100.IN-ADDR.ARPA
- 90.100.IN-ADDR.ARPA
- 91.100.IN-ADDR.ARPA
- 92.100.IN-ADDR.ARPA
- 93.100.IN-ADDR.ARPA
- 94.100.IN-ADDR.ARPA
- 95.100.IN-ADDR.ARPA
- 96.100.IN-ADDR.ARPA
- 97.100.IN-ADDR.ARPA
- 98.100.IN-ADDR.ARPA
- 99.100.IN-ADDR.ARPA
- 100.100.IN-ADDR.ARPA
- 101.100.IN-ADDR.ARPA
- 102.100.IN-ADDR.ARPA
- 103.100.IN-ADDR.ARPA
- 104.100.IN-ADDR.ARPA
- 105.100.IN-ADDR.ARPA
- 106.100.IN-ADDR.ARPA
- 107.100.IN-ADDR.ARPA
- 108.100.IN-ADDR.ARPA
- 109.100.IN-ADDR.ARPA
- 110.100.IN-ADDR.ARPA
- 111.100.IN-ADDR.ARPA
- 112.100.IN-ADDR.ARPA
- 113.100.IN-ADDR.ARPA
- 114.100.IN-ADDR.ARPA
- 115.100.IN-ADDR.ARPA
- 116.100.IN-ADDR.ARPA
- 117.100.IN-ADDR.ARPA
- 118.100.IN-ADDR.ARPA
- 119.100.IN-ADDR.ARPA
- 120.100.IN-ADDR.ARPA
- 121.100.IN-ADDR.ARPA

- 122.100.IN-ADDR.ARPA
- 123.100.IN-ADDR.ARPA
- 124.100.IN-ADDR.ARPA
- 125.100.IN-ADDR.ARPA
- 126.100.IN-ADDR.ARPA
- 127.100.IN-ADDR.ARPA
- 0.IN-ADDR.ARPA
- 127.IN-ADDR.ARPA
- 254.169.IN-ADDR.ARPA
- 2.0.192.IN-ADDR.ARPA
- 100.51.198.IN-ADDR.ARPA
- 113.0.203.IN-ADDR.ARPA
- 255.255.255.255.IN-ADDR.ARPA
- 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA
- 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA
- 8.B.D.0.1.0.0.2.IP6.ARPA
- D.F.IP6.ARPA
- 8.E.F.IP6.ARPA
- 9.E.F.IP6.ARPA
- A.E.F.IP6.ARPA
- B.E.F.IP6.ARPA
- EMPTY.AS112.ARPA
- HOME.ARPA

Empty zones can be set at the view level and only apply to views of class IN. Disabled empty zones are only inherited from options if there are no disabled empty zones specified at the view level. To override the options list of disabled zones, disable the root zone at the view level. For example:

```
disable-empty-zone ".";
```

If using the address ranges covered here, reverse zones covering the addresses should already be in place. In practice this appears to not be the case, with many queries being made to the infrastructure servers for names in these spaces. So many, in fact, that sacrificial servers had to be deployed to channel the query load away from the infrastructure servers.

Note: The real parent servers for these zones should disable all empty zones under the parent zone they serve. For the real root servers, this is all built-in empty zones. This enables them to return referrals to deeper in the tree.

---

empty-server This specifies the server name that appears in the returned SOA record for empty zones. If none is specified, the zone's name is used.

empty-contact This specifies the contact name that appears in the returned SOA record for empty zones. If none is specified, "." is used.

empty-zones-enable This enables or disables all empty zones. By default, they are enabled.

---

disable-empty-zone This disables individual empty zones. By default, none are disabled. This option can be specified multiple times.

Content Filtering

BIND 9 provides the ability to filter out responses from external DNS servers containing certain types of data in the answer section. Specifically, it can reject address (A or AAAA) records if the corresponding IPv4 or IPv6 addresses match the given `address_match_list` of the `deny-answer-addresses` option. It can also reject CNAME or DNAME records if the "alias" name (i.e., the CNAME alias or the substituted query name due to DNAME) matches the given `namelist` of the `deny-answer-aliases` option, where "match" means the alias name is a subdomain of one of the `name_list` elements. If the optional `namelist` is specified with `except-from`, records whose query name matches the list are accepted regardless of the filter setting. Likewise, if the alias name is a subdomain of the corresponding zone, the `deny-answer-aliases` filter does not apply; for example, even if "example.com" is specified for `deny-answer-aliases`,

```
www.example.com. CNAME xxx.example.com.
```

returned by an "example.com" server is accepted.

In the `address_match_list` of the `deny-answer-addresses` option, only `ip_addr` and `ip_prefix` are meaningful; any `key_id` is silently ignored.

If a response message is rejected due to the filtering, the entire message is discarded without being cached, and a SERV-FAIL error is returned to the client.

This filtering is intended to prevent "DNS rebinding attacks," in which an attacker, in response to a query for a domain name the attacker controls, returns an IP address within the user's own network or an alias name within the user's own domain. A naive web browser or script could then serve as an unintended proxy, allowing the attacker to get access to an internal node of the local network that could not be externally accessed otherwise. See the paper available at https://dl.acm.org/doi/10.1145/1315245.1315298 for more details about these attacks.

For example, with a domain named "example.net" and an internal network using an IPv4 prefix 192.0.2.0/24, an administrator might specify the following rules:

```
deny-answer-addresses { 192.0.2.0/24; } except-from { "example.net"; };
deny-answer-aliases { "example.net"; };
```

If an external attacker let a web browser in the local network look up an IPv4 address of "attacker.example.com", the attacker's DNS server would return a response like this:

```
attacker.example.com. A 192.0.2.1
```

in the answer section. Since the rdata of this record (the IPv4 address) matches the specified prefix 192.0.2.0/24, this response would be ignored.

On the other hand, if the browser looked up a legitimate internal web server "www.example.net" and the following response were returned to the BIND 9 server:

```
www.example.net. A 192.0.2.2
```

it would be accepted, since the owner name "www.example.net" matches the `except-from` element, "example.net".

Note that this is not really an attack on the DNS per se. In fact, there is nothing wrong with having an "external" name mapped to an "internal" IP address or domain name from the DNS point of view; it might actually be provided for a legitimate purpose, such as for debugging. As long as the mapping is provided by the correct owner, it either is not possible or does not make sense to detect whether the intent of the mapping is legitimate within the DNS. The "rebinding" attack must primarily be protected at the application that uses the DNS. For a large site, however, it may be difficult to

protect all possible applications at once. This filtering feature is provided only to help such an operational environment; turning it on is generally discouraged unless there is no other choice and the attack is a real threat to applications.

Care should be particularly taken if using this option for addresses within 127.0.0.0/8. These addresses are obviously "internal," but many applications conventionally rely on a DNS mapping from some name to such an address. Filtering out DNS records containing this address spuriously can break such applications.

Response Policy Zone (RPZ) Rewriting

BIND 9 includes a limited mechanism to modify DNS responses for requests analogous to email anti-spam DNS rejection lists. Responses can be changed to deny the existence of domains (NXDOMAIN), deny the existence of IP addresses for domains (NODATA), or contain other IP addresses or data.

Response policy zones are named in the `response-policy` option for the view, or among the global options if there is no `response-policy` option for the view. Response policy zones are ordinary DNS zones containing RRsets that can be queried normally if allowed. It is usually best to restrict those queries with something like `allow-query { localhost; };`. Note that zones using `masterfile-format map` cannot be used as policy zones.

A `response-policy` option can support multiple policy zones. To maximize performance, a radix tree is used to quickly identify response policy zones containing triggers that match the current query. This imposes an upper limit of 64 on the number of policy zones in a single `response-policy` option; more than that is a configuration error.

Rules encoded in response policy zones are processed after those defined in *Access Control*. All queries from clients which are not permitted access to the resolver are answered with a status code of REFUSED, regardless of configured RPZ rules.

Five policy triggers can be encoded in RPZ records.

RPZ-CLIENT-IP IP records are triggered by the IP address of the DNS client. Client IP address triggers are encoded in records that have owner names that are subdomains of `rpz-client-ip`, relativized to the policy zone origin name, and that encode an address or address block. IPv4 addresses are represented as `prefixlength.B4.B3.B2.B1.rpz-client-ip`. The IPv4 prefix length must be between 1 and 32. All four bytes - B4, B3, B2, and B1 - must be present. B4 is the decimal value of the least significant byte of the IPv4 address as in IN-ADDR.ARPA.

IPv6 addresses are encoded in a format similar to the standard IPv6 text representation, `prefixlength.W8.W7.W6.W5.W4.W3.W2.W1.rpz-client-ip`. Each of W8,…,W1 is a one- to four-digit hexadecimal number representing 16 bits of the IPv6 address as in the standard text representation of IPv6 addresses, but reversed as in IP6.ARPA. (Note that this representation of IPv6 addresses is different from IP6.ARPA, where each hex digit occupies a label.) All 8 words must be present except when one set of consecutive zero words is replaced with `.zz.`, analogous to double colons (::) in standard IPv6 text encodings. The IPv6 prefix length must be between 1 and 128.

QNAME QNAME policy records are triggered by query names of requests and targets of CNAME records resolved to generate the response. The owner name of a QNAME policy record is the query name relativized to the policy zone.

RPZ-IP IP triggers are IP addresses in an A or AAAA record in the ANSWER section of a response. They are encoded like client-IP triggers, except as subdomains of `rpz-ip`.

RPZ-NSDNAME NSDNAME triggers match names of authoritative servers for the query name, a parent of the query name, a CNAME for the query name, or a parent of a CNAME. They are encoded as subdomains of `rpz-nsdname`, relativized to the RPZ origin name. NSIP triggers match IP addresses in A and AAAA RRsets for domains that can be checked against NSDNAME policy records. The `nsdname-enable` phrase turns NSDNAME triggers off or on for a single policy zone or for all zones.

If authoritative name servers for the query name are not yet known, `named` recursively looks up the authoritative servers for the query name before applying an RPZ-NSDNAME rule, which can cause a processing delay.

RPZ-NSIP NSIP triggers match the IP addresses of authoritative servers. They are encoded like IP triggers, except as subdomains of `rpz-nsip`. NSDNAME and NSIP triggers are checked only for names with at least `min-ns-dots` dots. The default value of `min-ns-dots` is 1, to exclude top-level domains. The `nsip-enable` phrase turns NSIP triggers off or on for a single policy zone or for all zones.

If a name server's IP address is not yet known, `named` recursively looks up the IP address before applying an RPZ-NSIP rule, which can cause a processing delay. To speed up processing at the cost of precision, the `nsip-wait-recurse` option can be used; when set to `no`, RPZ-NSIP rules are only applied when a name server's IP address has already been looked up and cached. If a server's IP address is not in the cache, the RPZ-NSIP rule is ignored, but the address is looked up in the background and the rule is applied to subsequent queries. The default is `yes`, meaning RPZ-NSIP rules are always applied, even if an address needs to be looked up first.

The query response is checked against all response policy zones, so two or more policy records can be triggered by a response. Because DNS responses are rewritten according to at most one policy record, a single record encoding an action (other than `DISABLED` actions) must be chosen. Triggers, or the records that encode them, are chosen for rewriting in the following order:

1. Choose the triggered record in the zone that appears first in the response-policy option.

2. Prefer CLIENT-IP to QNAME to IP to NSDNAME to NSIP triggers in a single zone.

3. Among NSDNAME triggers, prefer the trigger that matches the smallest name under the DNSSEC ordering.

4. Among IP or NSIP triggers, prefer the trigger with the longest prefix.

5. Among triggers with the same prefix length, prefer the IP or NSIP trigger that matches the smallest IP address.

When the processing of a response is restarted to resolve DNAME or CNAME records and a policy record set has not been triggered, all response policy zones are again consulted for the DNAME or CNAME names and addresses.

RPZ record sets are any types of DNS record, except DNAME or DNSSEC, that encode actions or responses to individual queries. Any of the policies can be used with any of the triggers. For example, while the `TCP-only` policy is commonly used with `client-IP` triggers, it can be used with any type of trigger to force the use of TCP for responses with owner names in a zone.

PASSTHRU The auto-acceptance policy is specified by a CNAME whose target is `rpz-passthru`. It causes the response to not be rewritten and is most often used to "poke holes" in policies for CIDR blocks.

DROP The auto-rejection policy is specified by a CNAME whose target is `rpz-drop`. It causes the response to be discarded. Nothing is sent to the DNS client.

TCP-Only The "slip" policy is specified by a CNAME whose target is `rpz-tcp-only`. It changes UDP responses to short, truncated DNS responses that require the DNS client to try again with TCP. It is used to mitigate distributed DNS reflection attacks.

NXDOMAIN The "domain undefined" response is encoded by a CNAME whose target is the root domain (.).

NODATA The empty set of resource records is specified by a CNAME whose target is the wildcard top-level domain (`*.`). It rewrites the response to NODATA or ANCOUNT=0.

Local Data A set of ordinary DNS records can be used to answer queries. Queries for record types not in the set are answered with NODATA.

A special form of local data is a CNAME whose target is a wildcard such as *.example.com. It is used as if an ordinary CNAME after the asterisk (*) has been replaced with the query name. This special form is useful for query logging in the walled garden's authoritative DNS server.

All of the actions specified in all of the individual records in a policy zone can be overridden with a `policy` clause in the `response-policy` option. An organization using a policy zone provided by another organization might use this mechanism to redirect domains to its own walled garden.

GIVEN The placeholder policy says "do not override but perform the action specified in the zone."

DISABLED The testing override policy causes policy zone records to do nothing but log what they would have done if the policy zone were not disabled. The response to the DNS query is written (or not) according to any triggered policy records that are not disabled. Disabled policy zones should appear first, because they are often not logged if a higher-precedence trigger is found first.

PASSTHRU; DROP; TCP-Only; NXDOMAIN; NODATA These settings each override the corresponding per-record policy.

CNAME domain This causes all RPZ policy records to act as if they were "cname domain" records.

By default, the actions encoded in a response policy zone are applied only to queries that ask for recursion (RD=1). That default can be changed for a single policy zone, or for all response policy zones in a view, with a `recursive-only no` clause. This feature is useful for serving the same zone files both inside and outside an **RFC 1918** cloud and using RPZ to delete answers that would otherwise contain **RFC 1918** values on the externally visible name server or view.

Also by default, RPZ actions are applied only to DNS requests that either do not request DNSSEC metadata (DO=0) or when no DNSSEC records are available for the requested name in the original zone (not the response policy zone). This default can be changed for all response policy zones in a view with a `break-dnssec yes` clause. In that case, RPZ actions are applied regardless of DNSSEC. The name of the clause option reflects the fact that results rewritten by RPZ actions cannot verify.

No DNS records are needed for a QNAME or Client-IP trigger; the name or IP address itself is sufficient, so in principle the query name need not be recursively resolved. However, not resolving the requested name can leak the fact that response policy rewriting is in use, and that the name is listed in a policy zone, to operators of servers for listed names. To prevent that information leak, by default any recursion needed for a request is done before any policy triggers are considered. Because listed domains often have slow authoritative servers, this behavior can cost significant time. The `qname-wait-recurse no` option overrides the default and enables that behavior when recursion cannot change a non-error response. The option does not affect QNAME or client-IP triggers in policy zones listed after other zones containing IP, NSIP, and NSDNAME triggers, because those may depend on the A, AAAA, and NS records that would be found during recursive resolution. It also does not affect DNSSEC requests (DO=1) unless `break-dnssec yes` is in use, because the response would depend on whether RRSIG records were found during resolution. Using this option can cause error responses such as SERVFAIL to appear to be rewritten, since no recursion is being done to discover problems at the authoritative server.

The `dnsrps-enable yes` option turns on the DNS Response Policy Service (DNSRPS) interface, if it has been compiled in `named` using `configure --enable-dnsrps`.

The `dnsrps-options` block provides additional RPZ configuration settings, which are passed through to the DNSRPS provider library. Multiple DNSRPS settings in an `dnsrps-options` string should be separated with semi-colons (;). The DNSRPS provider, librpz, is passed a configuration string consisting of the `dnsrps-options` text, concatenated with settings derived from the `response-policy` statement.

Note: the `dnsrps-options` text should only include configuration settings that are specific to the DNSRPS provider. For example, the DNSRPS provider from Farsight Security takes options such as `dnsrpzd-conf`, `dnsrpzd-sock`, and `dnzrpzd-args` (for details of these options, see the `librpz` documentation). Other RPZ configuration settings could be included in `dnsrps-options` as well, but if `named` were switched back to traditional RPZ by setting `dnsrps-enable` to "no", those options would be ignored.

The TTL of a record modified by RPZ policies is set from the TTL of the relevant record in the policy zone. It is then limited to a maximum value. The `max-policy-ttl` clause changes the maximum number of seconds from its default of 5. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

For example, an administrator might use this option statement:

```
response-policy { zone "badlist"; };
```

and this zone statement:

```
zone "badlist" {type primary; file "primary/badlist"; allow-query {none;}; };
```

with this zone file:

```
$TTL 1H
@                      SOA LOCALHOST. named-mgr.example.com (1 1h 15m 30d 2h)
           NS  LOCALHOST.

; QNAME policy records.  There are no periods (.) after the owner names.
nxdomain.domain.com     CNAME   .                  ; NXDOMAIN policy
*.nxdomain.domain.com   CNAME   .                  ; NXDOMAIN policy
nodata.domain.com       CNAME   *.                 ; NODATA policy
*.nodata.domain.com     CNAME   *.                 ; NODATA policy
bad.domain.com          A       10.0.0.1           ; redirect to a walled garden
           AAAA    2001:2::1
bzone.domain.com        CNAME   garden.example.com.

; do not rewrite (PASSTHRU) OK.DOMAIN.COM
ok.domain.com           CNAME   rpz-passthru.

; redirect x.bzone.domain.com to x.bzone.domain.com.garden.example.com
*.bzone.domain.com      CNAME   *.garden.example.com.

; IP policy records that rewrite all responses containing A records in 127/8
;      except 127.0.0.1
8.0.0.0.127.rpz-ip      CNAME   .
32.1.0.0.127.rpz-ip     CNAME   rpz-passthru.

; NSDNAME and NSIP policy records
ns.domain.com.rpz-nsdname   CNAME   .
48.zz.2.2001.rpz-nsip       CNAME   .

; auto-reject and auto-accept some DNS clients
112.zz.2001.rpz-client-ip   CNAME   rpz-drop.
8.0.0.0.127.rpz-client-ip   CNAME   rpz-drop.

; force some DNS clients and responses in the example.com zone to TCP
16.0.0.1.10.rpz-client-ip   CNAME   rpz-tcp-only.
example.com                 CNAME   rpz-tcp-only.
*.example.com               CNAME   rpz-tcp-only.
```

RPZ can affect server performance. Each configured response policy zone requires the server to perform one to four additional database lookups before a query can be answered. For example, a DNS server with four policy zones, each with all four kinds of response triggers (QNAME, IP, NSIP, and NSDNAME), requires a total of 17 times as many database lookups as a similar DNS server with no response policy zones. A BIND 9 server with adequate memory and one response policy zone with QNAME and IP triggers might achieve a maximum queries-per-second (QPS) rate about 20% lower. A server with four response policy zones with QNAME and IP triggers might have a maximum QPS rate about 50% lower.

Responses rewritten by RPZ are counted in the `RPZRewrites` statistics.

The `log` clause can be used to optionally turn off rewrite logging for a particular response policy zone. By default, all rewrites are logged.

The `add-soa` option controls whether the RPZ's SOA record is added to the section for traceback of changes from this zone. This can be set at the individual policy zone level or at the response-policy level. The default is `yes`.

Updates to RPZ zones are processed asynchronously; if there is more than one update pending they are bundled together. If an update to a RPZ zone (for example, via IXFR) happens less than `min-update-interval` seconds after the most

recent update, the changes are not carried out until this interval has elapsed. The default is `60` seconds. For convenience, TTL-style time-unit suffixes may be used to specify the value. It also accepts ISO 8601 duration formats.

Response Rate Limiting

Excessive, almost-identical UDP *responses* can be controlled by configuring a `rate-limit` clause in an `options` or `view` statement. This mechanism keeps authoritative BIND 9 from being used to amplify reflection denial-of-service (DoS) attacks. Short BADCOOKIE errors or truncated (TC=1) responses can be sent to provide rate-limited responses to legitimate clients within a range of forged, attacked IP addresses. Legitimate clients react to dropped responses by retrying, to BADCOOKIE errors by including a server cookie when retrying, and to truncated responses by switching to TCP.

This mechanism is intended for authoritative DNS servers. It can be used on recursive servers, but can slow applications such as SMTP servers (mail receivers) and HTTP clients (web browsers) that repeatedly request the same domains. When possible, closing "open" recursive servers is better.

Response rate limiting uses a "credit" or "token bucket" scheme. Each combination of identical response and client has a conceptual "account" that earns a specified number of credits every second. A prospective response debits its account by one. Responses are dropped or truncated while the account is negative. Responses are tracked within a rolling window of time which defaults to 15 seconds, but which can be configured with the `window` option to any value from 1 to 3600 seconds (1 hour). The account cannot become more positive than the per-second limit or more negative than `window` times the per-second limit. When the specified number of credits for a class of responses is set to 0, those responses are not rate-limited.

The notions of "identical response" and "DNS client" for rate limiting are not simplistic. All responses to an address block are counted as if to a single client. The prefix lengths of address blocks are specified with `ipv4-prefix-length` (default 24) and `ipv6-prefix-length` (default 56).

All non-empty responses for a valid domain name (qname) and record type (qtype) are identical and have a limit specified with `responses-per-second` (default 0 or no limit). All empty (NODATA) responses for a valid domain, regardless of query type, are identical. Responses in the NODATA class are limited by `nodata-per-second` (default `responses-per-second`). Requests for any and all undefined subdomains of a given valid domain result in NX-DOMAIN errors, and are identical regardless of query type. They are limited by `nxdomains-per-second` (default `responses-per-second`). This controls some attacks using random names, but can be relaxed or turned off (set to 0) on servers that expect many legitimate NXDOMAIN responses, such as from anti-spam rejection lists. Referrals or delegations to the server of a given domain are identical and are limited by `referrals-per-second` (default `responses-per-second`).

Responses generated from local wildcards are counted and limited as if they were for the parent domain name. This controls flooding using random.wild.example.com.

All requests that result in DNS errors other than NXDOMAIN, such as SERVFAIL and FORMERR, are identical regardless of requested name (qname) or record type (qtype). This controls attacks using invalid requests or distant, broken authoritative servers. By default the limit on errors is the same as the `responses-per-second` value, but it can be set separately with `errors-per-second`.

Many attacks using DNS involve UDP requests with forged source addresses. Rate limiting prevents the use of BIND 9 to flood a network with responses to requests with forged source addresses, but could let a third party block responses to legitimate requests. There is a mechanism that can answer some legitimate requests from a client whose address is being forged in a flood. Setting `slip` to 2 (its default) causes every other UDP request without a valid server cookie to be answered with a small response. The small size and reduced frequency, and resulting lack of amplification, of "slipped" responses make them unattractive for reflection DoS attacks. `slip` must be between 0 and 10. A value of 0 does not "slip"; no small responses are sent due to rate limiting. Rather, all responses are dropped. A value of 1 causes every response to slip; values between 2 and 10 cause every nth response to slip.

If the request included a client cookie, then a "slipped" response is a BADCOOKIE error with a server cookie, which allows a legitimate client to include the server cookie to be exempted from the rate limiting when it retries the request. If

the request did not include a cookie, then a "slipped" response is a truncated (TC=1) response, which prompts a legitimate client to switch to TCP and thus be exempted from the rate limiting. Some error responses, including REFUSED and SERVFAIL, cannot be replaced with truncated responses and are instead leaked at the `slip` rate.

(Note: dropped responses from an authoritative server may reduce the difficulty of a third party successfully forging a response to a recursive resolver. The best security against forged responses is for authoritative operators to sign their zones using DNSSEC and for resolver operators to validate the responses. When this is not an option, operators who are more concerned with response integrity than with flood mitigation may consider setting `slip` to 1, causing all rate-limited responses to be truncated rather than dropped. This reduces the effectiveness of rate-limiting against reflection attacks.)

When the approximate query-per-second rate exceeds the `qps-scale` value, the `responses-per-second`, `errors-per-second`, `nxdomains-per-second`, and `all-per-second` values are reduced by the ratio of the current rate to the `qps-scale` value. This feature can tighten defenses during attacks. For example, with `qps-scale 250; responses-per-second 20;` and a total query rate of 1000 queries/second for all queries from all DNS clients including via TCP, then the effective responses/second limit changes to (250/1000)*20, or 5. Responses to requests that included a valid server cookie, and responses sent via TCP, are not limited but are counted to compute the query-per-second rate.

Communities of DNS clients can be given their own parameters or no rate limiting by putting `rate-limit` statements in `view` statements instead of in the global `option` statement. A `rate-limit` statement in a view replaces, rather than supplements, a `rate-limit` statement among the main options. DNS clients within a view can be exempted from rate limits with the `exempt-clients` clause.

UDP responses of all kinds can be limited with the `all-per-second` phrase. This rate limiting is unlike the rate limiting provided by `responses-per-second`, `errors-per-second`, and `nxdomains-per-second` on a DNS server, which are often invisible to the victim of a DNS reflection attack. Unless the forged requests of the attack are the same as the legitimate requests of the victim, the victim's requests are not affected. Responses affected by an `all-per-second` limit are always dropped; the `slip` value has no effect. An `all-per-second` limit should be at least 4 times as large as the other limits, because single DNS clients often send bursts of legitimate requests. For example, the receipt of a single mail message can prompt requests from an SMTP server for NS, PTR, A, and AAAA records as the incoming SMTP/TCP/IP connection is considered. The SMTP server can need additional NS, A, AAAA, MX, TXT, and SPF records as it considers the SMTP `Mail From` command. Web browsers often repeatedly resolve the same names that are duplicated in HTML <IMG> tags in a page. `all-per-second` is similar to the rate limiting offered by firewalls but is often inferior. Attacks that justify ignoring the contents of DNS responses are likely to be attacks on the DNS server itself. They usually should be discarded before the DNS server spends resources making TCP connections or parsing DNS requests, but that rate limiting must be done before the DNS server sees the requests.

The maximum size of the table used to track requests and rate-limit responses is set with `max-table-size`. Each entry in the table is between 40 and 80 bytes. The table needs approximately as many entries as the number of requests received per second. The default is 20,000. To reduce the cold start of growing the table, `min-table-size` (default 500) can set the minimum table size. Enable `rate-limit` category logging to monitor expansions of the table and inform choices for the initial and maximum table size.

Use `log-only yes` to test rate-limiting parameters without actually dropping any requests.

Responses dropped by rate limits are included in the `RateDropped` and `QryDropped` statistics. Responses that are truncated by rate limits are included in `RateSlipped` and `RespTruncated`.

NXDOMAIN Redirection

`named` supports NXDOMAIN redirection via two methods:

- Redirect zone (*zone Statement Grammar*)

- Redirect namespace

With either method, when `named` gets an NXDOMAIN response it examines a separate namespace to see if the NX-DOMAIN response should be replaced with an alternative response.

With a redirect zone (`zone "." { type redirect; };`), the data used to replace the NXDOMAIN is held in a single zone which is not part of the normal namespace. All the redirect information is contained in the zone; there are no delegations.

With a redirect namespace (`option { nxdomain-redirect <suffix> };`), the data used to replace the NXDOMAIN is part of the normal namespace and is looked up by appending the specified suffix to the original query name. This roughly doubles the cache required to process NXDOMAIN responses, as both the original NXDOMAIN response and the replacement data (or an NXDOMAIN indicating that there is no replacement) must be stored.

If both a redirect zone and a redirect namespace are configured, the redirect zone is tried first.

4.2.17 `server` Statement Grammar

```
server <netprefix> {
    bogus <boolean>;
    edns <boolean>;
    edns-udp-size <integer>;
    edns-version <integer>;
    keys <server_key>;
    max-udp-size <integer>;
    notify-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [
        dscp <integer> ];
    notify-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ]
        [ dscp <integer> ];
    padding <integer>;
    provide-ixfr <boolean>;
    query-source ( ( [ address ] ( <ipv4_address> | * ) [ port (
        <integer> | * ) ] ) | ( [ [ address ] ( <ipv4_address> | * ) ]
        port ( <integer> | * ) ) ) [ dscp <integer> ];
    query-source-v6 ( ( [ address ] ( <ipv6_address> | * ) [ port (
        <integer> | * ) ] ) | ( [ [ address ] ( <ipv6_address> | * ) ]
        port ( <integer> | * ) ) ) [ dscp <integer> ];
    request-expire <boolean>;
    request-ixfr <boolean>;
    request-nsid <boolean>;
    send-cookie <boolean>;
    tcp-keepalive <boolean>;
    tcp-only <boolean>;
    transfer-format ( many-answers | one-answer );
    transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [
        dscp <integer> ];
    transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * )
        ] [ dscp <integer> ];
    transfers <integer>;
};
```

4.2.18 `server` Statement Definition and Usage

The `server` statement defines characteristics to be associated with a remote name server. If a prefix length is specified, then a range of servers is covered. Only the most specific server clause applies, regardless of the order in `named.conf`.

The `server` statement can occur at the top level of the configuration file or inside a `view` statement. If a `view` statement contains one or more `server` statements, only those apply to the view and any top-level ones are ignored. If a view contains no `server` statements, any top-level `server` statements are used as defaults.

If a remote server is giving out bad data, marking it as bogus prevents further queries to it. The default value of `bogus` is `no`.

The `provide-ixfr` clause determines whether the local server, acting as primary, responds with an incremental zone transfer when the given remote server, a secondary, requests it. If set to `yes`, incremental transfer is provided whenever possible. If set to `no`, all transfers to the remote server are non-incremental. If not set, the value of the `provide-ixfr` option in the view or global options block is used as a default.

The `request-ixfr` clause determines whether the local server, acting as a secondary, requests incremental zone transfers from the given remote server, a primary. If not set, the value of the `request-ixfr` option in the view or global options block is used as a default. It may also be set in the zone block; if set there, it overrides the global or view setting for that zone.

IXFR requests to servers that do not support IXFR automatically fall back to AXFR. Therefore, there is no need to manually list which servers support IXFR and which ones do not; the global default of `yes` should always work. The purpose of the `provide-ixfr` and `request-ixfr` clauses is to make it possible to disable the use of IXFR even when both primary and secondary claim to support it: for example, if one of the servers is buggy and crashes or corrupts data when IXFR is used.

The `request-expire` clause determines whether the local server, when acting as a secondary, requests the EDNS EXPIRE value. The EDNS EXPIRE value indicates the remaining time before the zone data expires and needs to be refreshed. This is used when a secondary server transfers a zone from another secondary server; when transferring from the primary, the expiration timer is set from the EXPIRE field of the SOA record instead. The default is `yes`.

The `edns` clause determines whether the local server attempts to use EDNS when communicating with the remote server. The default is `yes`.

The `edns-udp-size` option sets the EDNS UDP size that is advertised by `named` when querying the remote server. Valid values are 512 to 4096 bytes; values outside this range are silently adjusted to the nearest value within it. This option is useful when advertising a different value to this server than the value advertised globally: for example, when there is a firewall at the remote site that is blocking large replies. Note: currently, this sets a single UDP size for all packets sent to the server; `named` does not deviate from this value. This differs from the behavior of `edns-udp-size` in `options` or `view` statements, where it specifies a maximum value. The `server` statement behavior may be brought into conformance with the `options`/`view` behavior in future releases.

The `edns-version` option sets the maximum EDNS VERSION that is sent to the server(s) by the resolver. The actual EDNS version sent is still subject to normal EDNS version-negotiation rules (see **RFC 6891**), the maximum EDNS version supported by the server, and any other heuristics that indicate that a lower version should be sent. This option is intended to be used when a remote server reacts badly to a given EDNS version or higher; it should be set to the highest version the remote server is known to support. Valid values are 0 to 255; higher values are silently adjusted. This option is not needed until higher EDNS versions than 0 are in use.

The `max-udp-size` option sets the maximum EDNS UDP message size `named` sends. Valid values are 512 to 4096 bytes; values outside this range are silently adjusted. This option is useful when there is a firewall that is blocking large replies from `named`.

The `padding` option adds EDNS Padding options to outgoing messages, increasing the packet size to a multiple of the specified block size. Valid block sizes range from 0 (the default, which disables the use of EDNS Padding) to 512 bytes. Larger values are reduced to 512, with a logged warning. Note: this option is not currently compatible with no TSIG or

SIG(0), as the EDNS OPT record containing the padding would have to be added to the packet after it had already been signed.

The `tcp-only` option sets the transport protocol to TCP. The default is to use the UDP transport and to fallback on TCP only when a truncated response is received.

The `tcp-keepalive` option adds EDNS TCP keepalive to messages sent over TCP. Note that currently idle timeouts in responses are ignored.

The server supports two zone transfer methods. The first, `one-answer`, uses one DNS message per resource record transferred. `many-answers` packs as many resource records as possible into a single message, which is more efficient. It is possible to specify which method to use for a server via the `transfer-format` option; if not set there, the `transfer-format` specified by the `options` statement is used.

`transfers` is used to limit the number of concurrent inbound zone transfers from the specified server. If no `transfers` clause is specified, the limit is set according to the `transfers-per-ns` option.

The `keys` clause identifies a `key_id` defined by the `key` statement, to be used for transaction security (see *TSIG*) when talking to the remote server. When a request is sent to the remote server, a request signature is generated using the key specified here and appended to the message. A request originating from the remote server is not required to be signed by this key.

Only a single key per server is currently supported.

The `transfer-source` and `transfer-source-v6` clauses specify the IPv4 and IPv6 source address, respectively, to be used for zone transfer with the remote server. For an IPv4 remote server, only `transfer-source` can be specified. Similarly, for an IPv6 remote server, only `transfer-source-v6` can be specified. For more details, see the description of `transfer-source` and `transfer-source-v6` in *Zone Transfers*.

The `notify-source` and `notify-source-v6` clauses specify the IPv4 and IPv6 source address, respectively, to be used for notify messages sent to remote servers. For an IPv4 remote server, only `notify-source` can be specified. Similarly, for an IPv6 remote server, only `notify-source-v6` can be specified.

The `query-source` and `query-source-v6` clauses specify the IPv4 and IPv6 source address, respectively, to be used for queries sent to remote servers. For an IPv4 remote server, only `query-source` can be specified. Similarly, for an IPv6 remote server, only `query-source-v6` can be specified.

The `request-nsid` clause determines whether the local server adds an NSID EDNS option to requests sent to the server. This overrides `request-nsid` set at the view or option level.

The `send-cookie` clause determines whether the local server adds a COOKIE EDNS option to requests sent to the server. This overrides `send-cookie` set at the view or option level. The `named` server may determine that COOKIE is not supported by the remote server and not add a COOKIE EDNS option to requests.

4.2.19 `statistics-channels` Statement Grammar

```
statistics-channels {
    inet ( <ipv4_address> | <ipv6_address> |
        * ) [ port ( <integer> | * ) ] [
        allow { <address_match_element>; ...
        } ];
};
```

4.2.20 `statistics-channels` Statement Definition and Usage

The `statistics-channels` statement declares communication channels to be used by system administrators to get access to statistics information on the name server.

This statement is intended to be flexible to support multiple communication protocols in the future, but currently only HTTP access is supported. It requires that BIND 9 be compiled with libxml2 and/or json-c (also known as libjson0); the `statistics-channels` statement is still accepted even if it is built without the library, but any HTTP access fails with an error.

An `inet` control channel is a TCP socket listening at the specified `ip_port` on the specified `ip_addr`, which can be an IPv4 or IPv6 address. An `ip_addr` of `*` (asterisk) is interpreted as the IPv4 wildcard address; connections are accepted on any of the system's IPv4 addresses. To listen on the IPv6 wildcard address, use an `ip_addr` of `::`.

If no port is specified, port 80 is used for HTTP channels. The asterisk (`*`) cannot be used for `ip_port`.

Attempts to open a statistics channel are restricted by the optional `allow` clause. Connections to the statistics channel are permitted based on the `address_match_list`. If no `allow` clause is present, `named` accepts connection attempts from any address; since the statistics may contain sensitive internal information, it is highly recommended to restrict the source of connection requests appropriately.

If no `statistics-channels` statement is present, `named` does not open any communication channels.

The statistics are available in various formats and views, depending on the URI used to access them. For example, if the statistics channel is configured to listen on 127.0.0.1 port 8888, then the statistics are accessible in XML format at http://127.0.0.1:8888/ or http://127.0.0.1:8888/xml. A CSS file is included, which can format the XML statistics into tables when viewed with a stylesheet-capable browser, and into charts and graphs using the Google Charts API when using a JavaScript-capable browser.

Broken-out subsets of the statistics can be viewed at http://127.0.0.1:8888/xml/v3/status (server uptime and last reconfiguration time), http://127.0.0.1:8888/xml/v3/server (server and resolver statistics), http://127.0.0.1:8888/xml/v3/zones (zone statistics), http://127.0.0.1:8888/xml/v3/net (network status and socket statistics), http://127.0.0.1:8888/xml/v3/mem (memory manager statistics), http://127.0.0.1:8888/xml/v3/tasks (task manager statistics), and http://127.0.0.1:8888/xml/v3/traffic (traffic sizes).

The full set of statistics can also be read in JSON format at http://127.0.0.1:8888/json, with the broken-out subsets at http://127.0.0.1:8888/json/v1/status (server uptime and last reconfiguration time), http://127.0.0.1:8888/json/v1/server (server and resolver statistics), http://127.0.0.1:8888/json/v1/zones (zone statistics), http://127.0.0.1:8888/json/v1/net (network status and socket statistics), http://127.0.0.1:8888/json/v1/mem (memory manager statistics), http://127.0.0.1:8888/json/v1/tasks (task manager statistics), and http://127.0.0.1:8888/json/v1/traffic (traffic sizes).

4.2.21 `trust-anchors` Statement Grammar

```
trust-anchors { <string> ( static-key |
    initial-key | static-ds | initial-ds )
    <integer> <integer> <integer>
    <quoted_string>; ... };
```

4.2.22 `trust-anchors` Statement Definition and Usage

The `trust-anchors` statement defines DNSSEC trust anchors. DNSSEC is described in *DNSSEC*.

A trust anchor is defined when the public key or public key digest for a non-authoritative zone is known but cannot be securely obtained through DNS, either because it is the DNS root zone or because its parent zone is unsigned. Once a key or digest has been configured as a trust anchor, it is treated as if it has been validated and proven secure.

The resolver attempts DNSSEC validation on all DNS data in subdomains of configured trust anchors. Validation below specified names can be temporarily disabled by using `rndc   nta`, or permanently disabled with the `validate-except` option.

All keys listed in `trust-anchors`, and their corresponding zones, are deemed to exist regardless of what parent zones say. Only keys configured as trust anchors are used to validate the DNSKEY RRset for the corresponding name. The parent's DS RRset is not used.

`trust-anchors` may be set at the top level of `named.conf` or within a view. If it is set in both places, the configurations are additive; keys defined at the top level are inherited by all views, but keys defined in a view are only used within that view.

The `trust-anchors` statement can contain multiple trust-anchor entries, each consisting of a domain name, followed by an "anchor type" keyword indicating the trust anchor's format, followed by the key or digest data.

If the anchor type is `static-key` or `initial-key`, then it is followed with the key's flags, protocol, and algorithm, plus the Base64 representation of the public key data. This is identical to the text representation of a DNSKEY record. Spaces, tabs, newlines, and carriage returns are ignored in the key data, so the configuration may be split into multiple lines.

If the anchor type is `static-ds` or `initial-ds`, it is followed with the key tag, algorithm, digest type, and the hexadecimal representation of the key digest. This is identical to the text representation of a DS record. Spaces, tabs, newlines, and carriage returns are ignored.

Trust anchors configured with the `static-key` or `static-ds` anchor types are immutable, while keys configured with `initial-key` or `initial-ds` can be kept up-to-date automatically, without intervention from the resolver operator. (`static-key` keys are identical to keys configured using the deprecated `trusted-keys` statement.)

Suppose, for example, that a zone's key-signing key was compromised, and the zone owner had to revoke and replace the key. A resolver which had the original key configured using `static-key` or `static-ds` would be unable to validate this zone any longer; it would reply with a SERVFAIL response code. This would continue until the resolver operator had updated the `trust-anchors` statement with the new key.

If, however, the trust anchor had been configured using `initial-key` or `initial-ds` instead, the zone owner could add a "stand-by" key to the zone in advance. `named` would store the stand-by key, and when the original key was revoked, `named` would be able to transition smoothly to the new key. It would also recognize that the old key had been revoked and cease using that key to validate answers, minimizing the damage that the compromised key could do. This is the process used to keep the ICANN root DNSSEC key up-to-date.

Whereas `static-key` and `static-ds` trust anchors continue to be trusted until they are removed from `named. conf`, an `initial-key` or `initial-ds` is only trusted *once*: for as long as it takes to load the managed key database and start the **RFC 5011** key maintenance process.

It is not possible to mix static with initial trust anchors for the same domain name.

The first time `named` runs with an `initial-key` or `initial-ds` configured in `named.conf`, it fetches the DNSKEY RRset directly from the zone apex, and validates it using the trust anchor specified in `trust-anchors`. If the DNSKEY RRset is validly signed by a key matching the trust anchor, then it is used as the basis for a new managed-keys database.

From that point on, whenever `named` runs, it sees the `initial-key` or `initial-ds` listed in `trust-anchors`, checks to make sure **RFC 5011** key maintenance has already been initialized for the specified domain, and if so, simply

moves on. The key specified in the `trust-anchors` statement is not used to validate answers; it is superseded by the key or keys stored in the managed-keys database.

The next time `named` runs after an `initial-key` or `initial-ds` has been *removed* from the `trust-anchors` statement (or changed to a `static-key` or `static-ds`), the corresponding zone is removed from the managed-keys database, and RFC 5011 key maintenance is no longer used for that domain.

In the current implementation, the managed-keys database is stored as a master-format zone file.

On servers which do not use views, this file is named `managed-keys.bind`. When views are in use, there is a separate managed-keys database for each view; the filename is the view name (or, if a view name contains characters which would make it illegal as a filename, a hash of the view name), followed by the suffix `.mkeys`.

When the key database is changed, the zone is updated. As with any other dynamic zone, changes are written into a journal file, e.g., `managed-keys.bind.jnl` or `internal.mkeys.jnl`. Changes are committed to the primary file as soon as possible afterward, usually within 30 seconds. Whenever `named` is using automatic key maintenance, the zone file and journal file can be expected to exist in the working directory. (For this reason, among others, the working directory should be always be writable by `named`.)

If the `dnssec-validation` option is set to `auto`, `named` automatically initializes an `initial-key` for the root zone. The key that is used to initialize the key-maintenance process is stored in `bind.keys`; the location of this file can be overridden with the `bindkeys-file` option. As a fallback in the event no `bind.keys` can be found, the initializing key is also compiled directly into `named`.

4.2.23 `dnssec-policy` Statement Grammar

4.2.24 `dnssec-policy` Statement Definition and Usage

The `dnssec-policy` statement defines a key and signing policy (KASP) for zones.

A KASP determines how one or more zones are signed with DNSSEC. For example, it specifies how often keys should roll, which cryptographic algorithms to use, and how often RRSIG records need to be refreshed.

Keys are not shared among zones, which means that one set of keys per zone is generated even if they have the same policy. If multiple views are configured with different versions of the same zone, each separate version uses the same set of signing keys.

Multiple key and signing policies can be configured. To attach a policy to a zone, add a `dnssec-policy` option to the `zone` statement, specifying the name of the policy that should be used.

Key rollover timing is computed for each key according to the key lifetime defined in the KASP. The lifetime may be modified by zone TTLs and propagation delays, to prevent validation failures. When a key reaches the end of its lifetime, `named` generates and publishes a new key automatically, then deactivates the old key and activates the new one; finally, the old key is retired according to a computed schedule.

Zone-signing key (ZSK) rollovers require no operator input. Key-signing key (KSK) and combined-signing key (CSK) rollovers require action to be taken to submit a DS record to the parent. Rollover timing for KSKs and CSKs is adjusted to take into account delays in processing and propagating DS updates.

There are two predefined `dnssec-policy` names: `none` and `default`. Setting a zone's policy to `none` is the same as not setting `dnssec-policy` at all; the zone is not signed. Policy `default` causes the zone to be signed with a single combined-signing key (CSK) using algorithm ECDSAP256SHA256; this key has an unlimited lifetime. (A verbose copy of this policy may be found in the source tree, in the file `doc/misc/dnssec-policy.default.conf`.)

Note: The default signing policy may change in future releases. This could require changes to a signing policy when upgrading to a new version of BIND. Check the release notes carefully when upgrading to be informed of such changes. To prevent policy changes on upgrade, use an explicitly defined `dnssec-policy`, rather than `default`.

If a `dnssec-policy` statement is modified and the server restarted or reconfigured, `named` attempts to change the policy smoothly from the old one to the new. For example, if the key algorithm is changed, then a new key is generated with the new algorithm, and the old algorithm is retired when the existing key's lifetime ends.

Note: Rolling to a new policy while another key rollover is already in progress is not yet supported, and may result in unexpected behavior.

The following options can be specified in a `dnssec-policy` statement:

dnskey-ttl This indicates the TTL to use when generating DNSKEY resource records. The default is 1 hour (3600 seconds).

keys This is a list specifying the algorithms and roles to use when generating keys and signing the zone. Entries in this list do not represent specific DNSSEC keys, which may be changed on a regular basis, but the roles that keys play in the signing policy. For example, configuring a KSK of algorithm RSASHA256 ensures that the DNSKEY RRset always includes a key-signing key for that algorithm.

Here is an example (for illustration purposes only) of some possible entries in a `keys` list:

```
keys {
    ksk key-directory lifetime unlimited algorithm rsasha256 2048;
    zsk lifetime P30D algorithm 8;
    csk lifetime P6MT12H3M15S algorithm ecdsa256;
};
```

This example specifies that three keys should be used in the zone. The first token determines which role the key plays in signing RRsets. If set to `ksk`, then this is a key-signing key; it has the KSK flag set and is only used to sign DNSKEY, CDS, and CDNSKEY RRsets. If set to `zsk`, this is a zone-signing key; the KSK flag is unset, and the key signs all RRsets *except* DNSKEY, CDS, and CDNSKEY. If set to `csk`, the key has the KSK flag set and is used to sign all RRsets.

An optional second token determines where the key is stored. Currently, keys can only be stored in the configured `key-directory`. This token may be used in the future to store keys in hardware security modules or separate directories.

The `lifetime` parameter specifies how long a key may be used before rolling over. In the example above, the first key has an unlimited lifetime, the second key may be used for 30 days, and the third key has a rather peculiar lifetime of 6 months, 12 hours, 3 minutes, and 15 seconds. A lifetime of 0 seconds is the same as `unlimited`.

Note that the lifetime of a key may be extended if retiring it too soon would cause validation failures. For example, if the key were configured to roll more frequently than its own TTL, its lifetime would automatically be extended to account for this.

The `algorithm` parameter specifies the key's algorithm, expressed either as a string ("rsasha256", "ecdsa384", etc.) or as a decimal number. An optional second parameter specifies the key's size in bits. If it is omitted, as shown in the example for the second and third keys, an appropriate default size for the algorithm is used. Each KSK/ZSK pair must have the same algorithm. A CSK combines the functionality of a ZSK and a KSK.

purge-keys This is the time after when DNSSEC keys that have been deleted from the zone can be removed from disk. If a key still determined to have presence (for example in some resolver cache), `named` will not remove the key files.

The default is `P90D` (90 days). Set this option to `0` to never purge deleted keys.

publish-safety This is a margin that is added to the pre-publication interval in rollover timing calculations, to give some extra time to cover unforeseen events. This increases the time between when keys are published and when they become active. The default is `PT1H` (1 hour).

---

retire-safety This is a margin that is added to the post-publication interval in rollover timing calculations, to give some extra time to cover unforeseen events. This increases the time a key remains published after it is no longer active. The default is `PT1H` (1 hour).

signatures-refresh This determines how frequently an RRSIG record needs to be refreshed. The signature is renewed when the time until the expiration time is less than the specified interval. The default is `P5D` (5 days), meaning signatures that expire in 5 days or sooner are refreshed.

signatures-validity This indicates the validity period of an RRSIG record (subject to inception offset and jitter). The default is `P2W` (2 weeks).

signatures-validity-dnskey This is similar to `signatures-validity`, but for DNSKEY records. The default is `P2W` (2 weeks).

max-zone-ttl Like the `max-zone-ttl` zone option, this specifies the maximum permissible TTL value, in seconds, for the zone.

This is needed in DNSSEC-maintained zones because when rolling to a new DNSKEY, the old key needs to remain available until RRSIG records have expired from caches. The `max-zone-ttl` option guarantees that the largest TTL in the zone is no higher than the set value.

Note: Because `map`-format files load directly into memory, this option cannot be used with them.

The default value is `PT24H` (24 hours). A `max-zone-ttl` of zero is treated as if the default value were in use.

nsec3param Use NSEC3 instead of NSEC, and optionally set the NSEC3 parameters.

Here is an example of an `nsec3` configuration:

```
nsec3param iterations 5 optout no salt-length 8;
```

The default is to use NSEC. The `iterations`, `optout` and `salt-length` parts are optional, but if not set, the values in the example above are the default NSEC3 parameters. Note that you don't specify a specific salt string, `named` will create a salt for you of the provided salt length.

zone-propagation-delay This is the expected propagation delay from the time when a zone is first updated to the time when the new version of the zone is served by all secondary servers. The default is `PT5M` (5 minutes).

parent-ds-ttl This is the TTL of the DS RRset that the parent zone uses. The default is `P1D` (1 day).

parent-propagation-delay This is the expected propagation delay from the time when the parent zone is updated to the time when the new version is served by all of the parent zone's name servers. The default is `PT1H` (1 hour).

Automated KSK Rollovers

BIND has mechanisms in place to facilitate automated KSK rollovers. It publishes CDS and CDNSKEY records that can be used by the parent zone to publish or withdraw the zone's DS records. BIND will query the parental agents to see if the new DS is actually published before withdrawing the old DNSSEC key.

Note: The DS response is not validated so it is recommended to set up a trust relationship with the parental agent. For example, use TSIG to authenticate the parental agent, or point to a validating resolver.

The following options apply to DS queries sent to `parental-agents`:

parental-source `parental-source` determines which local source address, and optionally UDP port, is used to send parental DS queries. This address must appear in the secondary server's `parental-agents` zone clause. This statement sets the `parental-source` for all zones, but can be overridden on a per-zone or per-view basis by including a `parental-source` statement within the `zone` or `view` block in the configuration file.

Note: Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

> **Warning:** Specifying a single port is discouraged, as it removes a layer of protection against spoofing errors.

> **Warning:** The configured `port` must not be same as the listening port.

parental-source-v6 This option acts like `parental-source`, but applies to parental DS queries sent to IPv6 addresses.

4.2.25 `managed-keys` Statement Grammar

```
managed-keys { <string> ( static-key
    | initial-key | static-ds |
    initial-ds ) <integer> <integer>
    <integer> <quoted_string>; ... };, deprecated
```

4.2.26 `managed-keys` Statement Definition and Usage

The `managed-keys` statement has been deprecated in favor of *trust-anchors Statement Grammar* with the `initial-key` keyword.

4.2.27 `trusted-keys` Statement Grammar

```
trusted-keys { <string> <integer>
    <integer> <integer>
    <quoted_string>; ... };, deprecated
```

4.2.28 `trusted-keys` Statement Definition and Usage

The `trusted-keys` statement has been deprecated in favor of *trust-anchors Statement Grammar* with the `static-key` keyword.

4.2.29 `view` Statement Grammar

```
view view_name [ class ] {
    match-clients { address_match_list } ;
    match-destinations { address_match_list } ;
    match-recursive-only yes_or_no ;
  [ view_option ; ... ]
  [ zone_statement ; ... ]
} ;
```

4.2.30 `view` Statement Definition and Usage

The `view` statement is a powerful feature of BIND 9 that lets a name server answer a DNS query differently depending on who is asking. It is particularly useful for implementing split DNS setups without having to run multiple servers.

Each `view` statement defines a view of the DNS namespace that is seen by a subset of clients. A client matches a view if its source IP address matches the `address_match_list` of the view's `match-clients` clause, and its destination IP address matches the `address_match_list` of the view's `match-destinations` clause. If not specified, both `match-clients` and `match-destinations` default to matching all addresses. In addition to checking IP addresses, `match-clients` and `match-destinations` can also take `keys` which provide an mechanism for the client to select the view. A view can also be specified as `match-recursive-only`, which means that only recursive requests from matching clients match that view. The order of the `view` statements is significant; a client request is resolved in the context of the first `view` that it matches.

Zones defined within a `view` statement are only accessible to clients that match the `view`. By defining a zone of the same name in multiple views, different zone data can be given to different clients: for example, "internal" and "external" clients in a split DNS setup.

Many of the options given in the `options` statement can also be used within a `view` statement, and then apply only when resolving queries with that view. When no view-specific value is given, the value in the `options` statement is used as a default. Also, zone options can have default values specified in the `view` statement; these view-specific defaults take precedence over those in the `options` statement.

Views are class-specific. If no class is given, class IN is assumed. Note that all non-IN views must contain a hint zone, since only the IN class has compiled-in default hints.

If there are no `view` statements in the config file, a default view that matches any client is automatically created in class IN. Any `zone` statements specified on the top level of the configuration file are considered to be part of this default view, and the `options` statement applies to the default view. If any explicit `view` statements are present, all `zone` statements must occur inside `view` statements.

Here is an example of a typical split DNS setup implemented using `view` statements:

```
view "internal" {
      // This should match our internal networks.
      match-clients { 10.0.0.0/8; };

      // Provide recursive service to internal
      // clients only.
      recursion yes;

      // Provide a complete view of the example.com
      // zone including addresses of internal hosts.
      zone "example.com" {
        type primary;
        file "example-internal.db";
```

```
      };
};

view "external" {
      // Match all clients not matched by the
      // previous view.
      match-clients { any; };

      // Refuse recursive service to external clients.
      recursion no;

      // Provide a restricted view of the example.com
      // zone containing only publicly accessible hosts.
      zone "example.com" {
       type primary;
       file "example-external.db";
      };
};
```

4.2.31 `zone` Statement Grammar

```
zone <string> [ <class> ] {
      type ( master | primary );
      allow-query { <address_match_element>; ... };
      allow-query-on { <address_match_element>; ... };
      allow-transfer { <address_match_element>; ... };
      allow-update { <address_match_element>; ... };
      also-notify [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
→... };
      alt-transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
      alt-transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
      auto-dnssec ( allow | maintain | off );
      check-dup-records ( fail | warn | ignore );
      check-integrity <boolean>;
      check-mx ( fail | warn | ignore );
      check-mx-cname ( fail | warn | ignore );
      check-names ( fail | warn | ignore );
      check-sibling <boolean>;
      check-spf ( warn | ignore );
      check-srv-cname ( fail | warn | ignore );
      check-wildcard <boolean>;
      database <string>;
      dialup ( notify | notify-passive | passive | refresh | <boolean> );
      dlz <string>;
      dnskey-sig-validity <integer>;
      dnssec-dnskey-kskonly <boolean>;
      dnssec-loadkeys-interval <integer>;
      dnssec-policy <string>;
      dnssec-secure-to-insecure <boolean>;
      dnssec-update-mode ( maintain | no-resign );
      file <quoted_string>;
```

```
      forward ( first | only );
      forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address> | <ipv6_
↪address> ) [ port <integer> ] [ dscp <integer> ]; ... };
      inline-signing <boolean>;
      ixfr-from-differences <boolean>;
      journal <quoted_string>;
      key-directory <quoted_string>;
      masterfile-format ( map | raw | text );
      masterfile-style ( full | relative );
      max-ixfr-ratio ( unlimited | <percentage> );
      max-journal-size ( default | unlimited | <sizeval> );
      max-records <integer>;
      max-transfer-idle-out <integer>;
      max-transfer-time-out <integer>;
      max-zone-ttl ( unlimited | <duration> );
      notify ( explicit | master-only | primary-only | <boolean> );
      notify-delay <integer>;
      notify-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp <integer>
↪ ];
      notify-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
↪<integer> ];
      notify-to-soa <boolean>;
      parental-agents [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> |
↪<ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key
↪<string> ]; ... };
      parental-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
↪<integer> ];
      parental-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
↪<integer> ];
      serial-update-method ( date | increment | unixtime );
      sig-signing-nodes <integer>;
      sig-signing-signatures <integer>;
      sig-signing-type <integer>;
      sig-validity-interval <integer> [ <integer> ];
      update-check-ksk <boolean>;
      update-policy ( local | { ( deny | grant ) <string> ( 6to4-self | external |␣
↪krb5-self | krb5-selfsub | krb5-subdomain | ms-self | ms-selfsub | ms-subdomain |␣
↪name | self | selfsub | selfwild | subdomain | tcp-self | wildcard | zonesub ) [
↪<string> ] <rrtypelist>; ... };
      zero-no-soa-ttl <boolean>;
      zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
      type ( slave | secondary );
      allow-notify { <address_match_element>; ... };
      allow-query { <address_match_element>; ... };
      allow-query-on { <address_match_element>; ... };
      allow-transfer { <address_match_element>; ... };
      allow-update-forwarding { <address_match_element>; ... };
      also-notify [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
↪address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
↪... };
      alt-transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
↪<integer> ];
      alt-transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
↪<integer> ];
```

```
      auto-dnssec ( allow | maintain | off );
      check-names ( fail | warn | ignore );
      database <string>;
      dialup ( notify | notify-passive | passive | refresh | <boolean> );
      dlz <string>;
      dnskey-sig-validity <integer>;
      dnssec-dnskey-kskonly <boolean>;
      dnssec-loadkeys-interval <integer>;
      dnssec-policy <string>;
      dnssec-update-mode ( maintain | no-resign );
      file <quoted_string>;
      forward ( first | only );
      forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address> | <ipv6_
→address> ) [ port <integer> ] [ dscp <integer> ]; ... };
      inline-signing <boolean>;
      ixfr-from-differences <boolean>;
      journal <quoted_string>;
      key-directory <quoted_string>;
      masterfile-format ( map | raw | text );
      masterfile-style ( full | relative );
      masters [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
→... };
      max-ixfr-ratio ( unlimited | <percentage> );
      max-journal-size ( default | unlimited | <sizeval> );
      max-records <integer>;
      max-refresh-time <integer>;
      max-retry-time <integer>;
      max-transfer-idle-in <integer>;
      max-transfer-idle-out <integer>;
      max-transfer-time-in <integer>;
      max-transfer-time-out <integer>;
      min-refresh-time <integer>;
      min-retry-time <integer>;
      multi-master <boolean>;
      notify ( explicit | master-only | primary-only | <boolean> );
      notify-delay <integer>;
      notify-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp <integer>
→ ];
      notify-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
      notify-to-soa <boolean>;
      parental-agents [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> |
→<ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key
→<string> ]; ... };
      parental-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
      parental-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
      primaries [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
→... };
      request-expire <boolean>;
      request-ixfr <boolean>;
      sig-signing-nodes <integer>;
      sig-signing-signatures <integer>;
      sig-signing-type <integer>;
```

```
     sig-validity-interval <integer> [ <integer> ];
     transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
     transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
     try-tcp-refresh <boolean>;
     update-check-ksk <boolean>;
     use-alt-transfer-source <boolean>;
     zero-no-soa-ttl <boolean>;
     zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
     type mirror;
     allow-notify { <address_match_element>; ... };
     allow-query { <address_match_element>; ... };
     allow-query-on { <address_match_element>; ... };
     allow-transfer { <address_match_element>; ... };
     allow-update-forwarding { <address_match_element>; ... };
     also-notify [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];
→... };
     alt-transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
     alt-transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
     check-names ( fail | warn | ignore );
     database <string>;
     file <quoted_string>;
     ixfr-from-differences <boolean>;
     journal <quoted_string>;
     masterfile-format ( map | raw | text );
     masterfile-style ( full | relative );
     masters [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];
→... };
     max-ixfr-ratio ( unlimited | <percentage> );
     max-journal-size ( default | unlimited | <sizeval> );
     max-records <integer>;
     max-refresh-time <integer>;
     max-retry-time <integer>;
     max-transfer-idle-in <integer>;
     max-transfer-idle-out <integer>;
     max-transfer-time-in <integer>;
     max-transfer-time-out <integer>;
     min-refresh-time <integer>;
     min-retry-time <integer>;
     multi-master <boolean>;
     notify ( explicit | master-only | primary-only | <boolean> );
     notify-delay <integer>;
     notify-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp <integer>
→ ];
     notify-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
     primaries [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];
→... };
```

```
        request-expire <boolean>;
        request-ixfr <boolean>;
        transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
        transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
        try-tcp-refresh <boolean>;
        use-alt-transfer-source <boolean>;
        zero-no-soa-ttl <boolean>;
        zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
        type hint;
        check-names ( fail | warn | ignore );
        delegation-only <boolean>;
        file <quoted_string>;
};
```

```
zone <string> [ <class> ] {
        type stub;
        allow-query { <address_match_element>; ... };
        allow-query-on { <address_match_element>; ... };
        check-names ( fail | warn | ignore );
        database <string>;
        delegation-only <boolean>;
        dialup ( notify | notify-passive | passive | refresh | <boolean> );
        file <quoted_string>;
        forward ( first | only );
        forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address> | <ipv6_
→address> ) [ port <integer> ] [ dscp <integer> ]; ... };
        masterfile-format ( map | raw | text );
        masterfile-style ( full | relative );
        masters [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
→... };
        max-records <integer>;
        max-refresh-time <integer>;
        max-retry-time <integer>;
        max-transfer-idle-in <integer>;
        max-transfer-time-in <integer>;
        min-refresh-time <integer>;
        min-retry-time <integer>;
        multi-master <boolean>;
        primaries [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];␣
→... };
        transfer-source ( <ipv4_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
        transfer-source-v6 ( <ipv6_address> | * ) [ port ( <integer> | * ) ] [ dscp
→<integer> ];
        use-alt-transfer-source <boolean>;
        zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
      type static-stub;
      allow-query { <address_match_element>; ... };
      allow-query-on { <address_match_element>; ... };
      forward ( first | only );
      forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address> | <ipv6_
→address> ) [ port <integer> ] [ dscp <integer> ]; ... };
      max-records <integer>;
      server-addresses { ( <ipv4_address> | <ipv6_address> ); ... };
      server-names { <string>; ... };
      zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
      type forward;
      delegation-only <boolean>;
      forward ( first | only );
      forwarders [ port <integer> ] [ dscp <integer> ] { ( <ipv4_address> | <ipv6_
→address> ) [ port <integer> ] [ dscp <integer> ]; ... };
};
```

```
zone <string> [ <class> ] {
      type redirect;
      allow-query { <address_match_element>; ... };
      allow-query-on { <address_match_element>; ... };
      dlz <string>;
      file <quoted_string>;
      masterfile-format ( map | raw | text );
      masterfile-style ( full | relative );
      masters [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];
→... };
      max-records <integer>;
      max-zone-ttl ( unlimited | <duration> );
      primaries [ port <integer> ] [ dscp <integer> ] { ( <remote-servers> | <ipv4_
→address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ];
→... };
      zone-statistics ( full | terse | none | <boolean> );
};
```

```
zone <string> [ <class> ] {
      type delegation-only;
};
```

```
zone <string> [ <class> ] {
      in-view <string>;
};
```

4.2.32 `zone` Statement Definition and Usage

Zone Types

The `type` keyword is required for the `zone` configuration unless it is an `in-view` configuration. Its acceptable values are: `primary` (or `master`), `secondary` (or `slave`), `mirror`, `hint`, `stub`, `static-stub`, `forward`, `redirect`, or `delegation-only`.

primary A primary zone has a master copy of the data for the zone and is able to provide authoritative answers for it. Type `master` is a synonym for `primary`.

secondary A secondary zone is a replica of a primary zone. Type `slave` is a synonym for `secondary`. The `primaries` list specifies one or more IP addresses of primary servers that the secondary contacts to update its copy of the zone. Primaries list elements can also be names of other primaries lists. By default, transfers are made from port 53 on the servers; this can be changed for all servers by specifying a port number before the list of IP addresses, or on a per-server basis after the IP address. Authentication to the primary can also be done with per-server TSIG keys. If a file is specified, then the replica is written to this file whenever the zone is changed, and reloaded from this file on a server restart. Use of a file is recommended, since it often speeds server startup and eliminates a needless waste of bandwidth. Note that for large numbers (in the tens or hundreds of thousands) of zones per server, it is best to use a two-level naming scheme for zone filenames. For example, a secondary server for the zone `example.com` might place the zone contents into a file called `ex/example.com`, where `ex/` is just the first two letters of the zone name. (Most operating systems behave very slowly if there are 100000 files in a single directory.)

mirror A mirror zone is similar to a zone of type `secondary`, except its data is subject to DNSSEC validation before being used in answers. Validation is applied to the entire zone during the zone transfer process, and again when the zone file is loaded from disk upon restarting `named`. If validation of a new version of a mirror zone fails, a retransfer is scheduled; in the meantime, the most recent correctly validated version of that zone is used until it either expires or a newer version validates correctly. If no usable zone data is available for a mirror zone, due to either transfer failure or expiration, traditional DNS recursion is used to look up the answers instead. Mirror zones cannot be used in a view that does not have recursion enabled.

Answers coming from a mirror zone look almost exactly like answers from a zone of type `secondary`, with the notable exceptions that the AA bit ("authoritative answer") is not set, and the AD bit ("authenticated data") is.

Mirror zones are intended to be used to set up a fast local copy of the root zone (see **RFC 8806**). A default list of primary servers for the IANA root zone is built into `named`, so its mirroring can be enabled using the following configuration:

```
zone "." {
    type mirror;
};
```

Mirror zone validation always happens for the entire zone contents. This ensures that each version of the zone used by the resolver is fully self-consistent with respect to DNSSEC. For incoming mirror zone IXFRs, every revision of the zone contained in the IXFR sequence is validated independently, in the order in which the zone revisions appear on the wire. For this reason, it might be useful to force use of AXFR for mirror zones by setting `request-ixfr no;` for the relevant zone (or view). Other, more efficient zone verification methods may be added in the future.

To make mirror zone contents persist between `named` restarts, use the *file* option.

Mirroring a zone other than root requires an explicit list of primary servers to be provided using the `primaries` option (see *primaries Statement Grammar* for details), and a key-signing key (KSK) for the specified zone to be explicitly configured as a trust anchor (see *trust-anchors Statement Definition and Usage*).

When configuring NOTIFY for a mirror zone, only `notify no;` and `notify explicit;` can be used at the zone level; any other `notify` setting at the zone level is a configuration error. Using any other `notify` setting at the `options` or `view` level causes that setting to be overridden with `notify explicit;` for the mirror

zone. The global default for the `notify` option is `yes`, so mirror zones are by default configured with `notify explicit;`.

Outgoing transfers of mirror zones are disabled by default but may be enabled using *allow-transfer*.

Note: Use of this zone type with any zone other than the root should be considered *experimental* and may cause performance issues, especially for zones that are large and/or frequently updated.

---

hint The initial set of root name servers is specified using a hint zone. When the server starts, it uses the root hints to find a root name server and get the most recent list of root name servers. If no hint zone is specified for class IN, the server uses a compiled-in default set of root servers hints. Classes other than IN have no built-in default hints.

stub A stub zone is similar to a secondary zone, except that it replicates only the NS records of a primary zone instead of the entire zone. Stub zones are not a standard part of the DNS; they are a feature specific to the BIND implementation.

Stub zones can be used to eliminate the need for a glue NS record in a parent zone, at the expense of maintaining a stub zone entry and a set of name server addresses in `named.conf`. This usage is not recommended for new configurations, and BIND 9 supports it only in a limited way. If a BIND 9 primary, serving a parent zone, has child stub zones configured, all the secondary servers for the parent zone also need to have the same child stub zones configured.

Stub zones can also be used as a way to force the resolution of a given domain to use a particular set of authoritative servers. For example, the caching name servers on a private network using **RFC 1918** addressing may be configured with stub zones for `10.in-addr.arpa` to use a set of internal name servers as the authoritative servers for that domain.

static-stub A static-stub zone is similar to a stub zone, with the following exceptions: the zone data is statically configured, rather than transferred from a primary server; and when recursion is necessary for a query that matches a static-stub zone, the locally configured data (name server names and glue addresses) is always used, even if different authoritative information is cached.

Zone data is configured via the `server-addresses` and `server-names` zone options.

The zone data is maintained in the form of NS and (if necessary) glue A or AAAA RRs internally, which can be seen by dumping zone databases with `rndc dumpdb -all`. The configured RRs are considered local configuration parameters rather than public data. Non-recursive queries (i.e., those with the RD bit off) to a static-stub zone are therefore prohibited and are responded to with REFUSED.

Since the data is statically configured, no zone maintenance action takes place for a static-stub zone. For example, there is no periodic refresh attempt, and an incoming notify message is rejected with an rcode of NOTAUTH.

Each static-stub zone is configured with internally generated NS and (if necessary) glue A or AAAA RRs.

forward A forward zone is a way to configure forwarding on a per-domain basis. A `zone` statement of type `forward` can contain a `forward` and/or `forwarders` statement, which applies to queries within the domain given by the zone name. If no `forwarders` statement is present, or an empty list for `forwarders` is given, then no forwarding is done for the domain, canceling the effects of any forwarders in the `options` statement. Thus, to use this type of zone to change the behavior of the global `forward` option (that is, "forward first" to, then "forward only", or vice versa), but use the same servers as set globally, re-specify the global forwarders.

redirect Redirect zones are used to provide answers to queries when normal resolution would result in NXDOMAIN being returned. Only one redirect zone is supported per view. `allow-query` can be used to restrict which clients see these answers.

If the client has requested DNSSEC records (DO=1) and the NXDOMAIN response is signed, no substitution occurs.

To redirect all NXDOMAIN responses to 100.100.100.2 and 2001:ffff:ffff::100.100.100.2, configure a type `redirect` zone named ".", with the zone file containing wildcard records that point to the desired addresses: `*. IN A 100.100.100.2` and `*. IN AAAA 2001:ffff:ffff::100.100.100.2`.

As another example, to redirect all Spanish names (under .ES), use similar entries but with the names `*.ES.` instead of `*.`. To redirect all commercial Spanish names (under COM.ES), use wildcard entries called `*.COM.ES.`.

Note that the redirect zone supports all possible types; it is not limited to A and AAAA records.

If a redirect zone is configured with a `primaries` option, then it is transferred in as if it were a secondary zone. Otherwise, it is loaded from a file as if it were a primary zone.

Because redirect zones are not referenced directly by name, they are not kept in the zone lookup table with normal primary and secondary zones. To reload a redirect zone, use `rndc reload -redirect`; to retransfer a redirect zone configured as a secondary, use `rndc retransfer -redirect`. When using `rndc reload` without specifying a zone name, redirect zones are reloaded along with other zones.

delegation-only This zone type is used to enforce the delegation-only status of infrastructure zones (e.g., COM, NET, ORG). Any answer that is received without an explicit or implicit delegation in the authority section is treated as NXDOMAIN. This does not apply to the zone apex, and should not be applied to leaf zones.

`delegation-only` has no effect on answers received from forwarders.

See caveats in *root-delegation-only*.

in-view When using multiple views, a `primary` or `secondary` zone configured in one view can be referenced in a subsequent view. This allows both views to use the same zone without the overhead of loading it more than once. This is configured using a `zone` statement, with an `in-view` option specifying the view in which the zone is defined. A `zone` statement containing `in-view` does not need to specify a type, since that is part of the zone definition in the other view.

See *Multiple Views* for more information.

Class

The zone's name may optionally be followed by a class. If a class is not specified, class `IN` (for `Internet`) is assumed. This is correct for the vast majority of cases.

The `hesiod` class is named for an information service from MIT's Project Athena. It was used to share information about various systems databases, such as users, groups, printers, and so on. The keyword `HS` is a synonym for hesiod.

Another MIT development is Chaosnet, a LAN protocol created in the mid-1970s. Zone data for it can be specified with the `CHAOS` class.

Zone Options

allow-notify See the description of `allow-notify` in *Access Control*.

allow-query See the description of `allow-query` in *Access Control*.

allow-query-on See the description of `allow-query-on` in *Access Control*.

allow-transfer See the description of `allow-transfer` in *Access Control*.

allow-update See the description of `allow-update` in *Access Control*.

update-policy This specifies a "Simple Secure Update" policy. See *Dynamic Update Policies*.

allow-update-forwarding See the description of `allow-update-forwarding` in *Access Control*.

---

also-notify This option is only meaningful if `notify` is active for this zone. The set of machines that receive a `DNS NOTIFY` message for this zone is made up of all the listed name servers (other than the primary) for the zone, plus any IP addresses specified with `also-notify`. A port may be specified with each `also-notify` address to send the notify messages to a port other than the default of 53. A TSIG key may also be specified to cause the `NOTIFY` to be signed by the given key. `also-notify` is not meaningful for stub zones. The default is the empty list.

check-names This option is used to restrict the character set and syntax of certain domain names in primary files and/or DNS responses received from the network. The default varies according to zone type. For `primary` zones the default is `fail`; for `secondary` zones the default is `warn`. It is not implemented for `hint` zones.

check-mx See the description of `check-mx` in *Boolean Options*.

check-spf See the description of `check-spf` in *Boolean Options*.

check-wildcard See the description of `check-wildcard` in *Boolean Options*.

check-integrity See the description of `check-integrity` in *Boolean Options*.

check-sibling See the description of `check-sibling` in *Boolean Options*.

zero-no-soa-ttl See the description of `zero-no-soa-ttl` in *Boolean Options*.

update-check-ksk See the description of `update-check-ksk` in *Boolean Options*.

dnssec-loadkeys-interval See the description of `dnssec-loadkeys-interval` in *options Statement Definition and Usage*.

dnssec-update-mode See the description of `dnssec-update-mode` in *options Statement Definition and Usage*.

dnssec-dnskey-kskonly See the description of `dnssec-dnskey-kskonly` in *Boolean Options*.

try-tcp-refresh See the description of `try-tcp-refresh` in *Boolean Options*.

database This specifies the type of database to be used to store the zone data. The string following the `database` keyword is interpreted as a list of whitespace-delimited words. The first word identifies the database type, and any subsequent words are passed as arguments to the database to be interpreted in a way specific to the database type.

The default is `rbt`, BIND 9's native in-memory red-black tree database. This database does not take arguments.

Other values are possible if additional database drivers have been linked into the server. Some sample drivers are included with the distribution but none are linked in by default.

dialup See the description of `dialup` in *Boolean Options*.

delegation-only This flag only applies to forward, hint, and stub zones. If set to `yes`, then the zone is treated as if it is also a delegation-only type zone.

See caveats in *root-delegation-only*.

file This sets the zone's filename. In `primary`, `hint`, and `redirect` zones which do not have `primaries` defined, zone data is loaded from this file. In `secondary`, `mirror`, `stub`, and `redirect` zones which do have `primaries` defined, zone data is retrieved from another server and saved in this file. This option is not applicable to other zone types.

forward This option is only meaningful if the zone has a forwarders list. The `only` value causes the lookup to fail after trying the forwarders and getting no answer, while `first` allows a normal lookup to be tried.

forwarders This is used to override the list of global forwarders. If it is not specified in a zone of type `forward`, no forwarding is done for the zone and the global options are not used.

journal This allows the default journal's filename to be overridden. The default is the zone's filename with ".`jnl`" appended. This is applicable to `primary` and `secondary` zones.

max-ixfr-ratio See the description of `max-ixfr-ratio` in *options Statement Definition and Usage*.

max-journal-size See the description of max-journal-size in *Server Resource Limits*.

max-records See the description of max-records in *Server Resource Limits*.

max-transfer-time-in See the description of max-transfer-time-in in *Zone Transfers*.

max-transfer-idle-in See the description of max-transfer-idle-in in *Zone Transfers*.

max-transfer-time-out See the description of max-transfer-time-out in *Zone Transfers*.

max-transfer-idle-out See the description of max-transfer-idle-out in *Zone Transfers*.

notify See the description of notify in *Boolean Options*.

notify-delay See the description of notify-delay in *Tuning*.

notify-to-soa See the description of notify-to-soa in *Boolean Options*.

zone-statistics See the description of zone-statistics in *options Statement Definition and Usage*.

server-addresses This option is only meaningful for static-stub zones. This is a list of IP addresses to which queries should be sent in recursive resolution for the zone. A non-empty list for this option internally configures the apex NS RR with associated glue A or AAAA RRs.

For example, if "example.com" is configured as a static-stub zone with 192.0.2.1 and 2001:db8::1234 in a server-addresses option, the following RRs are internally configured:

```
example.com. NS example.com.
example.com. A 192.0.2.1
example.com. AAAA 2001:db8::1234
```

These records are used internally to resolve names under the static-stub zone. For instance, if the server receives a query for "www.example.com" with the RD bit on, the server initiates recursive resolution and sends queries to 192.0.2.1 and/or 2001:db8::1234.

server-names This option is only meaningful for static-stub zones. This is a list of domain names of name servers that act as authoritative servers of the static-stub zone. These names are resolved to IP addresses when named needs to send queries to these servers. For this supplemental resolution to be successful, these names must not be a subdomain of the origin name of the static-stub zone. That is, when "example.net" is the origin of a static-stub zone, "ns.example" and "master.example.com" can be specified in the server-names option, but "ns.example.net" cannot; it is rejected by the configuration parser.

A non-empty list for this option internally configures the apex NS RR with the specified names. For example, if "example.com" is configured as a static-stub zone with "ns1.example.net" and "ns2.example.net" in a server-names option, the following RRs are internally configured:

```
example.com. NS ns1.example.net.
example.com. NS ns2.example.net.
```

These records are used internally to resolve names under the static-stub zone. For instance, if the server receives a query for "www.example.com" with the RD bit on, the server initiates recursive resolution, resolves "ns1.example.net" and/or "ns2.example.net" to IP addresses, and then sends queries to one or more of these addresses.

sig-validity-interval See the description of sig-validity-interval in *Tuning*.

sig-signing-nodes See the description of sig-signing-nodes in *Tuning*.

sig-signing-signatures See the description of sig-signing-signatures in *Tuning*.

sig-signing-type See the description of sig-signing-type in *Tuning*.

transfer-source See the description of transfer-source in *Zone Transfers*.

transfer-source-v6 See the description of `transfer-source-v6` in *Zone Transfers*.

alt-transfer-source See the description of `alt-transfer-source` in *Zone Transfers*.

alt-transfer-source-v6 See the description of `alt-transfer-source-v6` in *Zone Transfers*.

use-alt-transfer-source See the description of `use-alt-transfer-source` in *Zone Transfers*.

notify-source See the description of `notify-source` in *Zone Transfers*.

notify-source-v6 See the description of `notify-source-v6` in *Zone Transfers*.

min-refresh-time; max-refresh-time; min-retry-time; max-retry-time See the descriptions in *Tuning*.

ixfr-from-differences See the description of `ixfr-from-differences` in *Boolean Options*. (Note that the `ixfr-from-differences` choices of `primary` and `secondary` are not available at the zone level.)

key-directory See the description of `key-directory` in *options Statement Definition and Usage*.

auto-dnssec See the description of `auto-dnssec` in *options Statement Definition and Usage*.

serial-update-method See the description of `serial-update-method` in *options Statement Definition and Usage*.

inline-signing If `yes`, this enables "bump in the wire" signing of a zone, where an unsigned zone is transferred in or loaded from disk and a signed version of the zone is served with, possibly, a different serial number. This behavior is disabled by default.

multi-master See the description of `multi-master` in *Boolean Options*.

masterfile-format See the description of `masterfile-format` in *Tuning*.

max-zone-ttl See the description of `max-zone-ttl` in *options Statement Definition and Usage*.

dnssec-secure-to-insecure See the description of `dnssec-secure-to-insecure` in *Boolean Options*.

Dynamic Update Policies

BIND 9 supports two methods of granting clients the right to perform dynamic updates to a zone, configured by the `allow-update` or `update-policy` options.

The `allow-update` clause is a simple access control list. Any client that matches the ACL is granted permission to update any record in the zone.

The `update-policy` clause allows more fine-grained control over which updates are allowed. It specifies a set of rules, in which each rule either grants or denies permission for one or more names in the zone to be updated by one or more identities. Identity is determined by the key that signed the update request, using either TSIG or SIG(0). In most cases, `update-policy` rules only apply to key-based identities. There is no way to specify update permissions based on the client source address.

`update-policy` rules are only meaningful for zones of type `primary`, and are not allowed in any other zone type. It is a configuration error to specify both `allow-update` and `update-policy` at the same time.

A pre-defined `update-policy` rule can be switched on with the command `update-policy local;`. `named` automatically generates a TSIG session key when starting and stores it in a file; this key can then be used by local clients to update the zone while `named` is running. By default, the session key is stored in the file `/var/run/named/session.key`, the key name is "local-ddns", and the key algorithm is HMAC-SHA256. These values are configurable with the `session-keyfile`, `session-keyname`, and `session-keyalg` options, respectively. A client running on the local system, if run with appropriate permissions, may read the session key from the key file and use it to sign update requests. The zone's update policy is set to allow that key to change any record within the zone. Assuming the key name is "local-ddns", this policy is equivalent to:

```
update-policy { grant local-ddns zonesub any; };
```

with the additional restriction that only clients connecting from the local system are permitted to send updates.

Note that only one session key is generated by `named`; all zones configured to use `update-policy local` accept the same key.

The command `nsupdate -l` implements this feature, sending requests to localhost and signing them using the key retrieved from the session key file.

Other rule definitions look like this:

```
( grant | deny ) identity ruletype  name   types
```

Each rule grants or denies privileges. Rules are checked in the order in which they are specified in the `update-policy` statement. Once a message has successfully matched a rule, the operation is immediately granted or denied, and no further rules are examined. There are 13 types of rules; the rule type is specified by the `ruletype` field, and the interpretation of other fields varies depending on the rule type.

In general, a rule is matched when the key that signed an update request matches the `identity` field, the name of the record to be updated matches the `name` field (in the manner specified by the `ruletype` field), and the type of the record to be updated matches the `types` field. Details for each rule type are described below.

The `identity` field must be set to a fully qualified domain name. In most cases, this represents the name of the TSIG or SIG(0) key that must be used to sign the update request. If the specified name is a wildcard, it is subject to DNS wildcard expansion, and the rule may apply to multiple identities. When a TKEY exchange has been used to create a shared secret, the identity of the key used to authenticate the TKEY exchange is used as the identity of the shared secret. Some rule types use identities matching the client's Kerberos principal (e.g, `"host/machine@REALM"`) or Windows realm (`machine$@REALM`).

The `name` field also specifies a fully qualified domain name. This often represents the name of the record to be updated. Interpretation of this field is dependent on rule type.

If no `types` are explicitly specified, then a rule matches all types except RRSIG, NS, SOA, NSEC, and NSEC3. Types may be specified by name, including `ANY`; ANY matches all types except NSEC and NSEC3, which can never be updated. Note that when an attempt is made to delete all records associated with a name, the rules are checked for each existing record type.

The ruletype field has 16 values: `name`, `subdomain`, `zonesub`, `wildcard`, `self`, `selfsub`, `selfwild`, `ms-self`, `ms-selfsub`, `ms-subdomain`, `krb5-self`, `krb5-selfsub`, `krb5-subdomain`, `tcp-self`, `6to4-self`, and `external`.

name With exact-match semantics, this rule matches when the name being updated is identical to the contents of the `name` field.

subdomain This rule matches when the name being updated is a subdomain of, or identical to, the contents of the `name` field.

zonesub This rule is similar to subdomain, except that it matches when the name being updated is a subdomain of the zone in which the `update-policy` statement appears. This obviates the need to type the zone name twice, and enables the use of a standard `update-policy` statement in multiple zones without modification. When this rule is used, the `name` field is omitted.

wildcard The `name` field is subject to DNS wildcard expansion, and this rule matches when the name being updated is a valid expansion of the wildcard.

self This rule matches when the name of the record being updated matches the contents of the `identity` field. The `name` field is ignored. To avoid confusion, it is recommended that this field be set to the same value as the `identity` field or to ".". The `self` rule type is most useful when allowing one key per name to update, where

the key has the same name as the record to be updated. In this case, the `identity` field can be specified as `*` (asterisk).

selfsub This rule is similar to `self`, except that subdomains of `self` can also be updated.

selfwild This rule is similar to `self`, except that only subdomains of `self` can be updated.

ms-self When a client sends an UPDATE using a Windows machine principal (for example, `machine$@REALM`), this rule allows records with the absolute name of `machine.REALM` to be updated.

The realm to be matched is specified in the `identity` field.

The `name` field has no effect on this rule; it should be set to ".", as a placeholder.

For example, `grant EXAMPLE.COM ms-self . A AAAA` allows any machine with a valid principal in the realm `EXAMPLE.COM` to update its own address records.

ms-selfsub This is similar to `ms-self`, except it also allows updates to any subdomain of the name specified in the Windows machine principal, not just to the name itself.

ms-subdomain When a client sends an UPDATE using a Windows machine principal (for example, `machine$@REALM`), this rule allows any machine in the specified realm to update any record in the zone or in a specified subdomain of the zone.

The realm to be matched is specified in the `identity` field.

The `name` field specifies the subdomain that may be updated. If set to ".", or any other name at or above the zone apex, any name in the zone can be updated.

For example, if `update-policy` for the zone "example.com" includes `grant EXAMPLE.COM ms-subdomain hosts.example.com. AA AAAA`, any machine with a valid principal in the realm `EXAMPLE.COM` is able to update address records at or below `hosts.example.com`.

krb5-self When a client sends an UPDATE using a Kerberos machine principal (for example, `host/machine@REALM`), this rule allows records with the absolute name of `machine` to be updated, provided it has been authenticated by REALM. This is similar but not identical to `ms-self`, due to the `machine` part of the Kerberos principal being an absolute name instead of an unqualified name.

The realm to be matched is specified in the `identity` field.

The `name` field has no effect on this rule; it should be set to ".", as a placeholder.

For example, `grant EXAMPLE.COM krb5-self . A AAAA` allows any machine with a valid principal in the realm `EXAMPLE.COM` to update its own address records.

krb5-selfsub This is similar to `krb5-self`, except it also allows updates to any subdomain of the name specified in the `machine` part of the Kerberos principal, not just to the name itself.

krb5-subdomain This rule is identical to `ms-subdomain`, except that it works with Kerberos machine principals (i.e., `host/machine@REALM`) rather than Windows machine principals.

tcp-self This rule allows updates that have been sent via TCP and for which the standard mapping from the client's IP address into the `in-addr.arpa` and `ip6.arpa` namespaces matches the name to be updated. The `identity` field must match that name. The `name` field should be set to ".". Note that, since identity is based on the client's IP address, it is not necessary for update request messages to be signed.

Note: It is theoretically possible to spoof these TCP sessions.

6to4-self This allows the name matching a 6to4 IPv6 prefix, as specified in [RFC 3056](#), to be updated by any TCP connection from either the 6to4 network or from the corresponding IPv4 address. This is intended to allow NS or DNAME RRsets to be added to the `ip6.arpa` reverse tree.

The `identity` field must match the 6to4 prefix in `ip6.arpa`. The `name` field should be set to ".". Note that, since identity is based on the client's IP address, it is not necessary for update request messages to be signed.

In addition, if specified for an `ip6.arpa` name outside of the `2.0.0.2.ip6.arpa` namespace, the corresponding /48 reverse name can be updated. For example, TCP/IPv6 connections from 2001:DB8:ED0C::/48 can update records at `C.0.D.E.8.B.D.0.1.0.0.2.ip6.arpa`.

Note: It is theoretically possible to spoof these TCP sessions.

---

external This rule allows `named` to defer the decision of whether to allow a given update to an external daemon.

The method of communicating with the daemon is specified in the `identity` field, the format of which is "`local:path`", where "path" is the location of a Unix-domain socket. (Currently, "local" is the only supported mechanism.)

Requests to the external daemon are sent over the Unix-domain socket as datagrams with the following format:

```
Protocol version number (4 bytes, network byte order, currently 1)
Request length (4 bytes, network byte order)
Signer (null-terminated string)
Name (null-terminated string)
TCP source address (null-terminated string)
Rdata type (null-terminated string)
Key (null-terminated string)
TKEY token length (4 bytes, network byte order)
TKEY token (remainder of packet)
```

The daemon replies with a four-byte value in network byte order, containing either 0 or 1; 0 indicates that the specified update is not permitted, and 1 indicates that it is.

Multiple Views

When multiple views are in use, a zone may be referenced by more than one of them. Often, the views contain different zones with the same name, allowing different clients to receive different answers for the same queries. At times, however, it is desirable for multiple views to contain identical zones. The `in-view` zone option provides an efficient way to do this; it allows a view to reference a zone that was defined in a previously configured view. For example:

```
view internal {
    match-clients { 10/8; };

    zone example.com {
    type primary;
    file "example-external.db";
    };
};

view external {
    match-clients { any; };

    zone example.com {
    in-view internal;
    };
};
```

An `in-view` option cannot refer to a view that is configured later in the configuration file.

A `zone` statement which uses the `in-view` option may not use any other options, with the exception of `forward` and `forwarders`. (These options control the behavior of the containing view, rather than change the zone object itself.)

Zone-level ACLs (e.g., allow-query, allow-transfer), and other configuration details of the zone, are all set in the view the referenced zone is defined in. Be careful to ensure that ACLs are wide enough for all views referencing the zone.

An `in-view` zone cannot be used as a response policy zone.

An `in-view` zone is not intended to reference a `forward` zone.

4.3 Zone File

4.3.1 Types of Resource Records and When to Use Them

This section, largely borrowed from **RFC 1034**, describes the concept of a Resource Record (RR) and explains when each type is used. Since the publication of **RFC 1034**, several new RRs have been identified and implemented in the DNS. These are also included.

Resource Records

A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of separate RRs. The order of RRs in a set is not significant and need not be preserved by name servers, resolvers, or other parts of the DNS. However, sorting of multiple RRs is permitted for optimization purposes: for example, to specify that a particular nearby server be tried first. See *The sortlist Statement* and *RRset Ordering*.

The components of a Resource Record are:

owner name The domain name where the RR is found.

type An encoded 16-bit value that specifies the type of the resource record.

TTL The time-to-live of the RR. This field is a 32-bit integer in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.

class An encoded 16-bit value that identifies a protocol family or an instance of a protocol.

RDATA The resource data. The format of the data is type- and sometimes class-specific.

For a complete list of *types* of valid RRs, including those that have been obsoleted, please refer to https://en.wikipedia.org/wiki/List_of_DNS_record_types.

The following *classes* of resource records are currently valid in the DNS:

IN The Internet.

CH Chaosnet, a LAN protocol created at MIT in the mid-1970s. It was rarely used for its historical purpose, but was reused for BIND's built-in server information zones, e.g., `version.bind`.

HS Hesiod, an information service developed by MIT's Project Athena. It was used to share information about various systems databases, such as users, groups, printers, etc.

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL), which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; that also times out, but follows the refreshing policies for the zone. The TTL is assigned by the administrator

for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change is anticipated, the TTL can be reduced prior to the change to minimize inconsistency, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

Textual Expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In the examples provided in **RFC 1034**, a style similar to that used in primary files was employed in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner are listed the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. To avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interest of clarity.

The resource data or RDATA section of the RR is given using knowledge of the typical representation for the data.

For example, the RRs carried in a message might be shown as:

ISI.EDU.	MX	10 VENERA.ISI.EDU.
	MX	10 VAXA.ISI.EDU
VENERA.ISI.EDU	A	128.9.0.32
	A	10.1.0.52
VAXA.ISI.EDU	A	10.2.0.27
	A	128.9.0.33

The MX RRs have an RDATA section which consists of a 16-bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32-bit Internet address.

The above example shows six RRs, with two RRs at each of three domain names.

Here is another possible example:

XX.LCS.MIT.EDU.	IN A	10.0.0.44
	CH A	MIT.EDU. 2420

This shows two addresses for `XX.LCS.MIT.EDU`, each of a different class.

4.3.2 Discussion of MX Records

As described above, domain servers store information as a series of resource records, each of which contains a particular piece of information about a given domain name (which is usually, but not always, a host). The simplest way to think of an RR is as a typed pair of data, a domain name matched with a relevant datum and stored with some additional type information, to help systems determine when the RR is relevant.

MX records are used to control delivery of email. The data specified in the record is a priority and a domain name. The priority controls the order in which email delivery is attempted, with the lowest number first. If two priorities are the same, a server is chosen randomly. If no servers at a given priority are responding, the mail transport agent falls back to the next largest priority. Priority numbers do not have any absolute meaning; they are relevant only respective to other MX records for that domain name. The domain name given is the machine to which the mail is delivered. It *must* have an associated address record (A or AAAA); CNAME is not sufficient.

For a given domain, if there is both a CNAME record and an MX record, the MX record is in error and is ignored. Instead, the mail is delivered to the server specified in the MX record pointed to by the CNAME. For example:

example.com.	IN	MX	10	mail.example.com.
	IN	MX	10	mail2.example.com.
	IN	MX	20	mail.backup.org.
mail.example.com.	IN	A	10.0.0.1	
mail2.example.com.	IN	A	10.0.0.2	

Mail delivery is attempted to `mail.example.com` and `mail2.example.com` (in any order); if neither of those succeeds, delivery to `mail.backup.org` is attempted.

4.3.3 Setting TTLs

The time-to-live (TTL) of the RR field is a 32-bit integer represented in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long an RR can be cached before it should be discarded. The following three types of TTLs are currently used in a zone file.

SOA The last field in the SOA is the negative caching TTL. This controls how long other servers cache no-such-domain (NXDOMAIN) responses from this server.

The maximum time for negative caching is 3 hours (3h).

$TTL The $TTL directive at the top of the zone file (before the SOA) gives a default TTL for every RR without a specific TTL set.

RR TTLs Each RR can have a TTL as the second field in the RR, which controls how long other servers can cache it.

All of these TTLs default to units of seconds, though units can be explicitly specified: for example, `1h30m`.

4.3.4 Inverse Mapping in IPv4

Reverse name resolution (that is, translation from IP address to name) is achieved by means of the `in-addr.arpa` domain and PTR records. Entries in the in-addr.arpa domain are made in least-to-most significant order, read left to right. This is the opposite order to the way IP addresses are usually written. Thus, a machine with an IP address of 10.1.2.3 would have a corresponding in-addr.arpa name of 3.2.1.10.in-addr.arpa. This name should have a PTR resource record whose data field is the name of the machine or, optionally, multiple PTR records if the machine has more than one name. For example, in the `example.com` domain:

$ORIGIN	2.1.10.in-addr.arpa
3	IN PTR foo.example.com.

Note: The $ORIGIN line in this example is only to provide context; it does not necessarily appear in the actual usage. It is only used here to indicate that the example is relative to the listed origin.

4.3.5 Other Zone File Directives

The DNS "master file" format was initially defined in **RFC 1035** and has subsequently been extended. While the format itself is class-independent, all records in a zone file must be of the same class.

Master file directives include $ORIGIN, $INCLUDE, and $TTL.

The @ (at-sign)

When used in the label (or name) field, the asperand or at-sign (@) symbol represents the current origin. At the start of the zone file, it is the <zone_name>, followed by a trailing dot (.).

The $ORIGIN Directive

Syntax: $ORIGIN domain-name [comment]

$ORIGIN sets the domain name that is appended to any unqualified records. When a zone is first read, there is an implicit $ORIGIN <zone_name>``.``; note the trailing dot. The current $ORIGIN is appended to the domain specified in the $ORIGIN argument if it is not absolute.

```
$ORIGIN example.com.
WWW     CNAME   MAIN-SERVER
```

is equivalent to

```
WWW.EXAMPLE.COM. CNAME MAIN-SERVER.EXAMPLE.COM.
```

The $INCLUDE Directive

Syntax: $INCLUDE filename [origin] [comment]

This reads and processes the file filename as if it were included in the file at this point. The filename can be an absolute path, or a relative path. In the latter case it is read from named's working directory. If origin is specified, the file is processed with $ORIGIN set to that value; otherwise, the current $ORIGIN is used.

The origin and the current domain name revert to the values they had prior to the $INCLUDE once the file has been read.

Note: **RFC 1035** specifies that the current origin should be restored after an $INCLUDE, but it is silent on whether the current domain name should also be restored. BIND 9 restores both of them. This could be construed as a deviation from **RFC 1035**, a feature, or both.

The `$TTL` Directive

Syntax: `$TTL` default-ttl [comment]

This sets the default Time-To-Live (TTL) for subsequent records with undefined TTLs. Valid TTLs are of the range 0-2147483647 seconds.

`$TTL` is defined in **RFC 2308**.

4.3.6 BIND Primary File Extension: the `$GENERATE` Directive

Syntax: `$GENERATE` range lhs [ttl] [class] type rhs [comment]

`$GENERATE` is used to create a series of resource records that only differ from each other by an iterator. `$GENERATE` can be used to easily generate the sets of records required to support sub-/24 reverse delegations described in **RFC 2317**.

```
$ORIGIN 0.0.192.IN-ADDR.ARPA.
$GENERATE 1-2 @ NS SERVER$.EXAMPLE.
$GENERATE 1-127 $ CNAME $.0
```

is equivalent to

```
0.0.0.192.IN-ADDR.ARPA. NS SERVER1.EXAMPLE.
0.0.0.192.IN-ADDR.ARPA. NS SERVER2.EXAMPLE.
1.0.0.192.IN-ADDR.ARPA. CNAME 1.0.0.0.192.IN-ADDR.ARPA.
2.0.0.192.IN-ADDR.ARPA. CNAME 2.0.0.0.192.IN-ADDR.ARPA.
...
127.0.0.192.IN-ADDR.ARPA. CNAME 127.0.0.0.192.IN-ADDR.ARPA.
```

Both generate a set of A and MX records. Note the MX's right-hand side is a quoted string. The quotes are stripped when the right-hand side is processed.

```
$ORIGIN EXAMPLE.
$GENERATE 1-127 HOST-$ A 1.2.3.$
$GENERATE 1-127 HOST-$ MX "0 ."
```

is equivalent to

```
HOST-1.EXAMPLE.    A  1.2.3.1
HOST-1.EXAMPLE.    MX 0 .
HOST-2.EXAMPLE.    A  1.2.3.2
HOST-2.EXAMPLE.    MX 0 .
HOST-3.EXAMPLE.    A  1.2.3.3
HOST-3.EXAMPLE.    MX 0 .
...
HOST-127.EXAMPLE. A  1.2.3.127
HOST-127.EXAMPLE. MX 0 .
```

range This can be one of two forms: start-stop or start-stop/step. If the first form is used, then step is set to 1. "start", "stop", and "step" must be positive integers between 0 and (2^31)-1. "start" must not be larger than "stop".

owner This describes the owner name of the resource records to be created. Any single `$` (dollar sign) symbols within the `owner` string are replaced by the iterator value. To get a `$` in the output, escape the `$` using a backslash `\`, e.g., `\$`. The `$` may optionally be followed by modifiers which change the offset from the iterator, field width, and base.

Modifiers are introduced by a `{` (left brace) immediately following the `$`, as in `${offset[,width[,base]]}`. For example, `${-20,3,d}` subtracts 20 from the current value and prints the result as a decimal in a zero-padded

field of width 3. Available output forms are decimal (d), octal (o), hexadecimal (x or X for uppercase), and nibble (n or N for uppercase).

The default modifier is `${0,0,d}`. If the `owner` is not absolute, the current `$ORIGIN` is appended to the name.

In nibble mode, the value is treated as if it were a reversed hexadecimal string, with each hexadecimal digit as a separate label. The width field includes the label separator.

For compatibility with earlier versions, `$$` is still recognized as indicating a literal $ in the output.

ttl This specifies the time-to-live of the generated records. If not specified, this is inherited using the normal TTL inheritance rules.

class and ttl can be entered in either order.

class This specifies the class of the generated records. This must match the zone class if it is specified.

class and ttl can be entered in either order.

type This can be any valid type.

rdata This is a string containing the RDATA of the resource record to be created. It may be quoted if there are spaces in the string; the quotation marks do not appear in the generated record.

The `$GENERATE` directive is a BIND extension and not part of the standard zone file format.

4.3.7 Additional File Formats

In addition to the standard text format, BIND 9 supports the ability to read or dump to zone files in other formats.

The `raw` format is a binary representation of zone data in a manner similar to that used in zone transfers. Since it does not require parsing text, load time is significantly reduced.

An even faster alternative is the `map` format, which is an image of a BIND 9 in-memory zone database; it can be loaded directly into memory via the `mmap()` function and the zone can begin serving queries almost immediately. Because records are not indivdually processed when loading a `map` file, zones using this format cannot be used in `response-policy` statements.

For a primary server, a zone file in `raw` or `map` format is expected to be generated from a text zone file by the `named-compilezone` command. For a secondary server or a dynamic zone, the zone file is automatically generated when `named` dumps the zone contents after zone transfer or when applying prior updates, if one of these formats is specified by the `masterfile-format` option.

If a zone file in a binary format needs manual modification, it first must be converted to `text` format by the `named-compilezone` command, then converted back after editing. For example:

:: named-compilezone -f map -F text -o zonefile.text <origin> zonefile.map [edit zonefile.text] named-compilezone -f text -F map -o zonefile.map <origin> zonefile.text

Note that the `map` format is highly architecture-specific. A `map` file *cannot* be used on a system with different pointer size, endianness, or data alignment than the system on which it was generated, and should in general be used only inside a single system.

The `map` format is also dependent on the internal memory representation of a zone database, which may change from one release of BIND 9 to another. `map` files are never compatible across major releases, and may not be compatible across minor releases; any upgrade to BIND 9 may cause `map` files to be rejected when loading. If a `map` file is being used for a primary zone, it will need to be regenerated from text before restarting the server. If it used for a secondary zone, this is unnecessary; the rejection of the file will trigger a retransfer of the zone from the primary. (To avoid a spike in traffic upon restart, it may be desirable in some cases to convert `map` files to `text` format using `named-compilezone` before an upgrade, then back to `map` format with the new version of `named-compilezone` afterward.)

The use of `map` format may also be limited by operating system mmap(2) limits like `sysctl vm.max_map_count`. For Linux, this defaults to 65536, which limits the number of mapped zones that can be used without increasing `vm.max_map_count`.

`raw` format uses network byte order and avoids architecture- dependent data alignment so that it is as portable as possible, but it is still primarily expected to be used inside the same single system. To export a zone file in either `raw` or `map` format, or make a portable backup of such a file, conversion to `text` format is recommended.

4.4 BIND 9 Statistics

BIND 9 maintains lots of statistics information and provides several interfaces for users to access those statistics. The available statistics include all statistics counters that are meaningful in BIND 9, and other information that is considered useful.

The statistics information is categorized into the following sections:

Incoming Requests The number of incoming DNS requests for each OPCODE.

Incoming Queries The number of incoming queries for each RR type.

Outgoing Queries The number of outgoing queries for each RR type sent from the internal resolver, maintained per view.

Name Server Statistics Statistics counters for incoming request processing.

Zone Maintenance Statistics Statistics counters regarding zone maintenance operations, such as zone transfers.

Resolver Statistics Statistics counters for name resolutions performed in the internal resolver, maintained per view.

Cache DB RRsets Statistics counters related to cache contents, maintained per view.

> The "NXDOMAIN" counter is the number of names that have been cached as nonexistent. Counters named for RR types indicate the number of active RRsets for each type in the cache database.

> If an RR type name is preceded by an exclamation point (!), it represents the number of records in the cache which indicate that the type does not exist for a particular name; this is also known as "NXRRSET". If an RR type name is preceded by a hash mark (#), it represents the number of RRsets for this type that are present in the cache but whose TTLs have expired; these RRsets may only be used if stale answers are enabled. If an RR type name is preceded by a tilde (~), it represents the number of RRsets for this type that are present in the cache database but are marked for garbage collection; these RRsets cannot be used.

Socket I/O Statistics Statistics counters for network-related events.

A subset of Name Server Statistics is collected and shown per zone for which the server has the authority, when `zone-statistics` is set to `full` (or `yes`), for backward compatibility. See the description of `zone-statistics` in *options Statement Definition and Usage* for further details.

These statistics counters are shown with their zone and view names. The view name is omitted when the server is not configured with explicit views.

There are currently two user interfaces to get access to the statistics. One is in plain-text format, dumped to the file specified by the `statistics-file` configuration option; the other is remotely accessible via a statistics channel when the `statistics-channels` statement is specified in the configuration file (see *statistics-channels Statement Grammar*.)

4.4.1 The Statistics File

The text format statistics dump begins with a line, like:

```
+++ Statistics Dump +++ (973798949)
```

The number in parentheses is a standard Unix-style timestamp, measured in seconds since January 1, 1970. Following that line is a set of statistics information, which is categorized as described above. Each section begins with a line, like:

```
++ Name Server Statistics ++
```

Each section consists of lines, each containing the statistics counter value followed by its textual description; see below for available counters. For brevity, counters that have a value of 0 are not shown in the statistics file.

The statistics dump ends with the line where the number is identical to the number in the beginning line; for example:

```
--- Statistics Dump --- (973798949)
```

4.4.2 Statistics Counters

The following lists summarize the statistics counters that BIND 9 provides. For each counter, the abbreviated symbol name is given; these symbols are shown in the statistics information accessed via an HTTP statistics channel. The description of the counter is also shown in the statistics file but, in this document, may be slightly modified for better readability.

Name Server Statistics Counters

Requestv4 This indicates the number of IPv4 requests received. Note: this also counts non-query requests.

Requestv6 This indicates the number of IPv6 requests received. Note: this also counts non-query requests.

ReqEdns0 This indicates the number of requests received with EDNS(0).

ReqBadEDN SVer This indicates the number of requests received with an unsupported EDNS version.

ReqTSIG This indicates the number of requests received with TSIG.

ReqSIG0 This indicates the number of requests received with SIG(0).

ReqBadSIG This indicates the number of requests received with an invalid (TSIG or SIG(0)) signature.

ReqTCP This indicates the number of TCP requests received.

AuthQryRej This indicates the number of rejected authoritative (non-recursive) queries.

RecQryRej This indicates the number of rejected recursive queries.

XfrRej This indicates the number of rejected zone transfer requests.

UpdateRej This indicates the number of rejected dynamic update requests.

Response This indicates the number of responses sent.

RespTruncated This indicates the number of truncated responses sent.

RespEDNS0 This indicates the number of responses sent with EDNS(0).

RespTSIG This indicates the number of responses sent with TSIG.

RespSIG0 This indicates the number of responses sent with SIG(0).

QrySuccess This indicates the number of queries that resulted in a successful answer, meaning queries which return a NOERROR response with at least one answer RR. This corresponds to the `success` counter of previous versions of BIND 9.

QryAuthAns This indicates the number of queries that resulted in an authoritative answer.

QryNoauthAns This indicates the number of queries that resulted in a non-authoritative answer.

QryReferral This indicates the number of queries that resulted in a referral answer. This corresponds to the `re-ferral` counter of previous versions of BIND 9.

QryNxrrset This indicates the number of queries that resulted in NOERROR responses with no data. This corresponds to the `nxrrset` counter of previous versions of BIND 9.

QrySERVFAIL This indicates the number of queries that resulted in SERVFAIL.

QryFORMERR This indicates the number of queries that resulted in FORMERR.

QryNXDOMAIN This indicates the number of queries that resulted in NXDOMAIN. This corresponds to the `nxdomain` counter of previous versions of BIND 9.

QryRecursion This indicates the number of queries that caused the server to perform recursion in order to find the final answer. This corresponds to the `recursion` counter of previous versions of BIND 9.

QryDuplicate This indicates the number of queries which the server attempted to recurse but for which it discovered an existing query with the same IP address, port, query ID, name, type, and class already being processed. This corresponds to the `duplicate` counter of previous versions of BIND 9.

QryDropped This indicates the number of recursive queries for which the server discovered an excessive number of existing recursive queries for the same name, type, and class, and which were subsequently dropped. This is the number of dropped queries due to the reason explained with the `clients-per-query` and `max-clients-per-query` options (see *clients-per-query*). This corresponds to the `dropped` counter of previous versions of BIND 9.

QryFailure This indicates the number of query failures. This corresponds to the `failure` counter of previous versions of BIND 9. Note: this counter is provided mainly for backward compatibility with previous versions; normally, more fine-grained counters such as `AuthQryRej` and `RecQryRej` that would also fall into this counter are provided, so this counter is not of much interest in practice.

QryNXRedir This indicates the number of queries that resulted in NXDOMAIN that were redirected.

QryNXRedirRLookup This indicates the number of queries that resulted in NXDOMAIN that were redirected and resulted in a successful remote lookup.

XfrReqDone This indicates the number of requested and completed zone transfers.

UpdateReqFwd This indicates the number of forwarded update requests.

UpdateRespFwd This indicates the number of forwarded update responses.

UpdateFwdFail This indicates the number of forwarded dynamic updates that failed.

UpdateDone This indicates the number of completed dynamic updates.

UpdateFail This indicates the number of failed dynamic updates.

UpdateBadPrereq This indicates the number of dynamic updates rejected due to a prerequisite failure.

RateDropped This indicates the number of responses dropped due to rate limits.

RateSlipped This indicates the number of responses truncated by rate limits.

RPZRewrites This indicates the number of response policy zone rewrites.

Zone Maintenance Statistics Counters

NotifyOutv4 This indicates the number of IPv4 notifies sent.

NotifyOutv6 This indicates the number of IPv6 notifies sent.

NotifyInv4 This indicates the number of IPv4 notifies received.

NotifyInv6 This indicates the number of IPv6 notifies received.

NotifyRej This indicates the number of incoming notifies rejected.

SOAOutv4 This indicates the number of IPv4 SOA queries sent.

SOAOutv6 This indicates the number of IPv6 SOA queries sent.

AXFRReqv4 This indicates the number of requested IPv4 AXFRs.

AXFRReqv6 This indicates the number of requested IPv6 AXFRs.

IXFRReqv4 This indicates the number of requested IPv4 IXFRs.

IXFRReqv6 This indicates the number of requested IPv6 IXFRs.

XfrSuccess This indicates the number of successful zone transfer requests.

XfrFail This indicates the number of failed zone transfer requests.

Resolver Statistics Counters

Queryv4 This indicates the number of IPv4 queries sent.

Queryv6 This indicates the number of IPv6 queries sent.

Responsev4 This indicates the number of IPv4 responses received.

Responsev6 This indicates the number of IPv6 responses received.

NXDOMAIN This indicates the number of NXDOMAINs received.

SERVFAIL This indicates the number of SERVFAILs received.

FORMERR This indicates the number of FORMERRs received.

OtherError This indicates the number of other errors received.

EDNS0Fail This indicates the number of EDNS(0) query failures.

Mismatch This indicates the number of mismatched responses received, meaning the DNS ID, response's source address, and/or the response's source port does not match what was expected. (The port must be 53 or as defined by the `port` option.) This may be an indication of a cache poisoning attempt.

Truncated This indicates the number of truncated responses received.

Lame This indicates the number of lame delegations received.

Retry This indicates the number of query retries performed.

QueryAbort This indicates the number of queries aborted due to quota control.

QuerySockFail This indicates the number of failures in opening query sockets. One common reason for such failures is due to a limitation on file descriptors.

QueryTimeout This indicates the number of query timeouts.

GlueFetchv4 This indicates the number of IPv4 NS address fetches invoked.

GlueFetchv6 This indicates the number of IPv6 NS address fetches invoked.

GlueFetchv4Fail This indicates the number of failed IPv4 NS address fetches.

GlueFetchv6Fail This indicates the number of failed IPv6 NS address fetches.

ValAttempt This indicates the number of attempted DNSSEC validations.

ValOk This indicates the number of successful DNSSEC validations.

ValNegOk This indicates the number of successful DNSSEC validations on negative information.

ValFail This indicates the number of failed DNSSEC validations.

QryRTTnn This provides a frequency table on query round-trip times (RTTs). Each `nn` specifies the corresponding frequency. In the sequence of `nn_1`, `nn_2`, …, `nn_m`, the value of `nn_i` is the number of queries whose RTTs are between `nn_(i-1)` (inclusive) and `nn_i` (exclusive) milliseconds. For the sake of convenience, we define `nn_0` to be 0. The last entry should be represented as `nn_m+`, which means the number of queries whose RTTs are equal to or greater than `nn_m` milliseconds.

Socket I/O Statistics Counters

Socket I/O statistics counters are defined per socket type, which are `UDP4` (UDP/IPv4), `UDP6` (UDP/IPv6), `TCP4` (TCP/IPv4), `TCP6` (TCP/IPv6), `Unix` (Unix Domain), and `FDwatch` (sockets opened outside the socket module). In the following list, `<TYPE>` represents a socket type. Not all counters are available for all socket types; exceptions are noted in the descriptions.

\<TYPE>Open This indicates the number of sockets opened successfully. This counter does not apply to the `FDwatch` type.

\<TYPE>OpenFail This indicates the number of failures to open sockets. This counter does not apply to the `FDwatch` type.

\<TYPE>Close This indicates the number of closed sockets.

\<TYPE>BindFail This indicates the number of failures to bind sockets.

\<TYPE>ConnFail This indicates the number of failures to connect sockets.

\<TYPE>Conn This indicates the number of connections established successfully.

\<TYPE>AcceptFail This indicates the number of failures to accept incoming connection requests. This counter does not apply to the `UDP` and `FDwatch` types.

\<TYPE>Accept This indicates the number of incoming connections successfully accepted. This counter does not apply to the `UDP` and `FDwatch` types.

\<TYPE>SendErr This indicates the number of errors in socket send operations.

\<TYPE>RecvErr This indicates the number of errors in socket receive operations, including errors of send operations on a connected UDP socket, notified by an ICMP error message.

ADVANCED DNS FEATURES

5.1 Notify

DNS NOTIFY is a mechanism that allows primary servers to notify their secondary servers of changes to a zone's data. In response to a `NOTIFY` from a primary server, the secondary checks to see that its version of the zone is the current version and, if not, initiates a zone transfer.

For more information about DNS `NOTIFY`, see the description of the `notify` option in *Boolean Options* and the description of the zone option `also-notify` in *Zone Transfers*. The `NOTIFY` protocol is specified in **RFC 1996**.

Note: As a secondary zone can also be a primary to other secondaries, `named`, by default, sends `NOTIFY` messages for every zone it loads. Specifying `notify primary-only;` causes `named` to only send `NOTIFY` for primary zones that it loads.

5.2 Dynamic Update

Dynamic update is a method for adding, replacing, or deleting records in a primary server by sending it a special form of DNS messages. The format and meaning of these messages is specified in **RFC 2136**.

Dynamic update is enabled by including an `allow-update` or an `update-policy` clause in the `zone` statement.

If the zone's `update-policy` is set to `local`, updates to the zone are permitted for the key `local-ddns`, which is generated by `named` at startup. See *Dynamic Update Policies* for more details.

Dynamic updates using Kerberos-signed requests can be made using the TKEY/GSS protocol, either by setting the `tkey-gssapi-keytab` option or by setting both the `tkey-gssapi-credential` and `tkey-domain` options. Once enabled, Kerberos-signed requests are matched against the update policies for the zone, using the Kerberos principal as the signer for the request.

Updating of secure zones (zones using DNSSEC) follows **RFC 3007**: RRSIG, NSEC, and NSEC3 records affected by updates are automatically regenerated by the server using an online zone key. Update authorization is based on transaction signatures and an explicit server policy.

5.2.1 The Journal File

All changes made to a zone using dynamic update are stored in the zone's journal file. This file is automatically created by the server when the first dynamic update takes place. The name of the journal file is formed by appending the extension `.jnl` to the name of the corresponding zone file unless specifically overridden. The journal file is in a binary format and should not be edited manually.

The server also occasionally writes ("dumps") the complete contents of the updated zone to its zone file. This is not done immediately after each dynamic update because that would be too slow when a large zone is updated frequently. Instead, the dump is delayed by up to 15 minutes, allowing additional updates to take place. During the dump process, transient files are created with the extensions `.jnw` and `.jbk`; under ordinary circumstances, these are removed when the dump is complete, and can be safely ignored.

When a server is restarted after a shutdown or crash, it replays the journal file to incorporate into the zone any updates that took place after the last zone dump.

Changes that result from incoming incremental zone transfers are also journaled in a similar way.

The zone files of dynamic zones cannot normally be edited by hand because they are not guaranteed to contain the most recent dynamic changes; those are only in the journal file. The only way to ensure that the zone file of a dynamic zone is up-to-date is to run `rndc stop`.

To make changes to a dynamic zone manually, follow these steps: first, disable dynamic updates to the zone using `rndc freeze zone`. This updates the zone file with the changes stored in its `.jnl` file. Then, edit the zone file. Finally, run `rndc thaw zone` to reload the changed zone and re-enable dynamic updates.

`rndc sync zone` updates the zone file with changes from the journal file without stopping dynamic updates; this may be useful for viewing the current zone state. To remove the `.jnl` file after updating the zone file, use `rndc sync -clean`.

5.3 Incremental Zone Transfers (IXFR)

The incremental zone transfer (IXFR) protocol is a way for secondary servers to transfer only changed data, instead of having to transfer an entire zone. The IXFR protocol is specified in **RFC 1995**.

When acting as a primary server, BIND 9 supports IXFR for those zones where the necessary change history information is available. These include primary zones maintained by dynamic update and secondary zones whose data was obtained by IXFR. For manually maintained primary zones, and for secondary zones obtained by performing a full zone transfer (AXFR), IXFR is supported only if the option `ixfr-from-differences` is set to `yes`.

When acting as a secondary server, BIND 9 attempts to use IXFR unless it is explicitly disabled. For more information about disabling IXFR, see the description of the `request-ixfr` clause of the `server` statement.

When a secondary server receives a zone via AXFR, it creates a new copy of the zone database and then swaps it into place; during the loading process, queries continue to be served from the old database with no interference. When receiving a zone via IXFR, however, changes are applied to the running zone, which may degrade query performance during the transfer. If a server receiving an IXFR request determines that the response size would be similar in size to an AXFR response, it may wish to send AXFR instead. The threshold at which this determination is made can be configured using the `max-ixfr-ratio` option.

5.4 Split DNS

Setting up different views of the DNS space to internal and external resolvers is usually referred to as a *split DNS* setup. There are several reasons an organization might want to set up its DNS this way.

One common reason to use split DNS is to hide "internal" DNS information from "external" clients on the Internet. There is some debate as to whether this is actually useful. Internal DNS information leaks out in many ways (via email headers, for example) and most savvy "attackers" can find the information they need using other means. However, since listing addresses of internal servers that external clients cannot possibly reach can result in connection delays and other annoyances, an organization may choose to use split DNS to present a consistent view of itself to the outside world.

Another common reason for setting up a split DNS system is to allow internal networks that are behind filters or in **RFC 1918** space (reserved IP space, as documented in **RFC 1918**) to resolve DNS on the Internet. Split DNS can also be used to allow mail from outside back into the internal network.

5.4.1 Example Split DNS Setup

Let's say a company named *Example, Inc.* (`example.com`) has several corporate sites that have an internal network with reserved Internet Protocol (IP) space and an external demilitarized zone (DMZ), or "outside" section of a network, that is available to the public.

Example, Inc. wants its internal clients to be able to resolve external hostnames and to exchange mail with people on the outside. The company also wants its internal resolvers to have access to certain internal-only zones that are not available at all outside of the internal network.

To accomplish this, the company sets up two sets of name servers. One set is on the inside network (in the reserved IP space) and the other set is on bastion hosts, which are "proxy" hosts in the DMZ that can talk to both sides of its network.

The internal servers are configured to forward all queries, except queries for `site1.internal`, `site2.internal`, `site1.example.com`, and `site2.example.com`, to the servers in the DMZ. These internal servers have complete sets of information for `site1.example.com`, `site2.example.com`, `site1.internal`, and `site2.internal`.

To protect the `site1.internal` and `site2.internal` domains, the internal name servers must be configured to disallow all queries to these domains from any external hosts, including the bastion hosts.

The external servers, which are on the bastion hosts, are configured to serve the "public" version of the `site1.example.com` and `site2.example.com` zones. This could include things such as the host records for public servers (`www.example.com` and `ftp.example.com`) and mail exchange (MX) records (`a.mx.example.com` and `b.mx.example.com`).

In addition, the public `site1.example.com` and `site2.example.com` zones should have special MX records that contain wildcard (`*`) records pointing to the bastion hosts. This is needed because external mail servers have no other way of determining how to deliver mail to those internal hosts. With the wildcard records, the mail is delivered to the bastion host, which can then forward it on to internal hosts.

Here's an example of a wildcard MX record:

```
*    IN MX 10 external1.example.com.
```

Now that they accept mail on behalf of anything in the internal network, the bastion hosts need to know how to deliver mail to internal hosts. The resolvers on the bastion hosts need to be configured to point to the internal name servers for DNS resolution.

Queries for internal hostnames are answered by the internal servers, and queries for external hostnames are forwarded back out to the DNS servers on the bastion hosts.

For all of this to work properly, internal clients need to be configured to query *only* the internal name servers for DNS queries. This could also be enforced via selective filtering on the network.

If everything has been set properly, Example, Inc.'s internal clients are now able to:

- Look up any hostnames in the `site1.example.com` and `site2.example.com` zones.

- Look up any hostnames in the `site1.internal` and `site2.internal` domains.

- Look up any hostnames on the Internet.

- Exchange mail with both internal and external users.

Hosts on the Internet are able to:

- Look up any hostnames in the `site1.example.com` and `site2.example.com` zones.

- Exchange mail with anyone in the `site1.example.com` and `site2.example.com` zones.

Here is an example configuration for the setup just described above. Note that this is only configuration information; for information on how to configure the zone files, see *Sample Configurations*.

Internal DNS server config:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };

options {
    ...
    ...
    forward only;
    // forward to external servers
    forwarders {
    bastion-ips-go-here;
    };
    // sample allow-transfer (no one)
    allow-transfer { none; };
    // restrict query access
    allow-query { internals; externals; };
    // restrict recursion
    allow-recursion { internals; };
    ...
    ...
};

// sample primary zone
zone "site1.example.com" {
  type primary;
  file "m/site1.example.com";
  // do normal iterative resolution (do not forward)
  forwarders { };
  allow-query { internals; externals; };
  allow-transfer { internals; };
};

// sample secondary zone
zone "site2.example.com" {
  type secondary;
  file "s/site2.example.com";
  primaries { 172.16.72.3; };
```

```
  forwarders { };
  allow-query { internals; externals; };
  allow-transfer { internals; };
};

zone "site1.internal" {
  type primary;
  file "m/site1.internal";
  forwarders { };
  allow-query { internals; };
  allow-transfer { internals; }
};

zone "site2.internal" {
  type secondary;
  file "s/site2.internal";
  primaries { 172.16.72.3; };
  forwarders { };
  allow-query { internals };
  allow-transfer { internals; }
};
```

External (bastion host) DNS server configuration:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };

options {
  ...
  ...
  // sample allow-transfer (no one)
  allow-transfer { none; };
  // default query access
  allow-query { any; };
  // restrict cache access
  allow-query-cache { internals; externals; };
  // restrict recursion
  allow-recursion { internals; externals; };
  ...
  ...
};

// sample secondary zone
zone "site1.example.com" {
  type primary;
  file "m/site1.foo.com";
  allow-transfer { internals; externals; };
};

zone "site2.example.com" {
  type secondary;
  file "s/site2.foo.com";
  masters { another_bastion_host_maybe; };
  allow-transfer { internals; externals; }
};
```

In the `resolv.conf` (or equivalent) on the bastion host(s):

```
search ...
nameserver 172.16.72.2
nameserver 172.16.72.3
nameserver 172.16.72.4
```

5.5 TSIG

TSIG (Transaction SIGnatures) is a mechanism for authenticating DNS messages, originally specified in **RFC 2845**. It allows DNS messages to be cryptographically signed using a shared secret. TSIG can be used in any DNS transaction, as a way to restrict access to certain server functions (e.g., recursive queries) to authorized clients when IP-based access control is insufficient or needs to be overridden, or as a way to ensure message authenticity when it is critical to the integrity of the server, such as with dynamic UPDATE messages or zone transfers from a primary to a secondary server.

This section is a guide to setting up TSIG in BIND. It describes the configuration syntax and the process of creating TSIG keys.

`named` supports TSIG for server-to-server communication, and some of the tools included with BIND support it for sending messages to `named`:

- *nsupdate - dynamic DNS update utility* supports TSIG via the `-k`, `-l`, and `-y` command-line options, or via the `key` command when running interactively.

- *dig - DNS lookup utility* supports TSIG via the `-k` and `-y` command-line options.

5.5.1 Generating a Shared Key

TSIG keys can be generated using the `tsig-keygen` command; the output of the command is a `key` directive suitable for inclusion in `named.conf`. The key name, algorithm, and size can be specified by command-line parameters; the defaults are "tsig-key", HMAC-SHA256, and 256 bits, respectively.

Any string which is a valid DNS name can be used as a key name. For example, a key to be shared between servers called `host1` and `host2` could be called "host1-host2.", and this key can be generated using:

```
$ tsig-keygen host1-host2. > host1-host2.key
```

This key may then be copied to both hosts. The key name and secret must be identical on both hosts. (Note: copying a shared secret from one server to another is beyond the scope of the DNS. A secure transport mechanism should be used: secure FTP, SSL, ssh, telephone, encrypted email, etc.)

`tsig-keygen` can also be run as `ddns-confgen`, in which case its output includes additional configuration text for setting up dynamic DNS in `named`. See *ddns-confgen - TSIG key generation tool* for details.

5.5.2 Loading a New Key

For a key shared between servers called `host1` and `host2`, the following could be added to each server's `named.conf` file:

```
key "host1-host2." {
    algorithm hmac-sha256;
    secret "DAopyf1mhCbFVZw7pgmNPBoLUq8wEUT7UuPoLENP2HY=";
};
```

(This is the same key generated above using `tsig-keygen`.)

Since this text contains a secret, it is recommended that either `named.conf` not be world-readable, or that the `key` directive be stored in a file which is not world-readable and which is included in `named.conf` via the `include` directive.

Once a key has been added to `named.conf` and the server has been restarted or reconfigured, the server can recognize the key. If the server receives a message signed by the key, it is able to verify the signature. If the signature is valid, the response is signed using the same key.

TSIG keys that are known to a server can be listed using the command `rndc tsig-list`.

5.5.3 Instructing the Server to Use a Key

A server sending a request to another server must be told whether to use a key, and if so, which key to use.

For example, a key may be specified for each server in the `primaries` statement in the definition of a secondary zone; in this case, all SOA QUERY messages, NOTIFY messages, and zone transfer requests (AXFR or IXFR) are signed using the specified key. Keys may also be specified in the `also-notify` statement of a primary or secondary zone, causing NOTIFY messages to be signed using the specified key.

Keys can also be specified in a `server` directive. Adding the following on `host1`, if the IP address of `host2` is 10.1.2.3, would cause *all* requests from `host1` to `host2`, including normal DNS queries, to be signed using the `host1-host2.` key:

```
server 10.1.2.3 {
    keys { host1-host2. ;};
};
```

Multiple keys may be present in the `keys` statement, but only the first one is used. As this directive does not contain secrets, it can be used in a world-readable file.

Requests sent by `host2` to `host1` would *not* be signed, unless a similar `server` directive were in `host2`'s configuration file.

When any server sends a TSIG-signed DNS request, it expects the response to be signed with the same key. If a response is not signed, or if the signature is not valid, the response is rejected.

5.5.4 TSIG-Based Access Control

TSIG keys may be specified in ACL definitions and ACL directives such as `allow-query`, `allow-transfer`, and `allow-update`. The above key would be denoted in an ACL element as `key host1-host2.`

Here is an example of an `allow-update` directive using a TSIG key:

```
allow-update { !{ !localnets; any; }; key host1-host2. ;};
```

This allows dynamic updates to succeed only if the UPDATE request comes from an address in `localnets`, *and* if it is signed using the `host1-host2.` key.

See *Dynamic Update Policies* for a discussion of the more flexible `update-policy` statement.

5.5.5 Errors

Processing of TSIG-signed messages can result in several errors:

- If a TSIG-aware server receives a message signed by an unknown key, the response will be unsigned, with the TSIG extended error code set to BADKEY.

- If a TSIG-aware server receives a message from a known key but with an invalid signature, the response will be unsigned, with the TSIG extended error code set to BADSIG.

- If a TSIG-aware server receives a message with a time outside of the allowed range, the response will be signed but the TSIG extended error code set to BADTIME, and the time values will be adjusted so that the response can be successfully verified.

In all of the above cases, the server returns a response code of NOTAUTH (not authenticated).

5.6 TKEY

TKEY (Transaction KEY) is a mechanism for automatically negotiating a shared secret between two hosts, originally specified in **RFC 2930**.

There are several TKEY "modes" that specify how a key is to be generated or assigned. BIND 9 implements only one of these modes: Diffie-Hellman key exchange. Both hosts are required to have a KEY record with algorithm DH (though this record is not required to be present in a zone).

The TKEY process is initiated by a client or server by sending a query of type TKEY to a TKEY-aware server. The query must include an appropriate KEY record in the additional section, and must be signed using either TSIG or SIG(0) with a previously established key. The server's response, if successful, contains a TKEY record in its answer section. After this transaction, both participants have enough information to calculate a shared secret using Diffie-Hellman key exchange. The shared secret can then be used to sign subsequent transactions between the two servers.

TSIG keys known by the server, including TKEY-negotiated keys, can be listed using `rndc tsig-list`.

TKEY-negotiated keys can be deleted from a server using `rndc tsig-delete`. This can also be done via the TKEY protocol itself, by sending an authenticated TKEY query specifying the "key deletion" mode.

5.7 SIG(0)

BIND partially supports DNSSEC SIG(0) transaction signatures as specified in **RFC 2535** and **RFC 2931**. SIG(0) uses public/private keys to authenticate messages. Access control is performed in the same manner as with TSIG keys; privileges can be granted or denied in ACL directives based on the key name.

When a SIG(0) signed message is received, it is only verified if the key is known and trusted by the server. The server does not attempt to recursively fetch or validate the key.

SIG(0) signing of multiple-message TCP streams is not supported.

The only tool shipped with BIND 9 that generates SIG(0) signed messages is `nsupdate`.

5.8 DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security ("DNSSEC-bis") extensions, defined in **RFC 4033**, **RFC 4034**, and **RFC 4035**. This section describes the creation and use of DNSSEC signed zones.

In order to set up a DNSSEC secure zone, there are a series of steps which must be followed. BIND 9 ships with several tools that are used in this process, which are explained in more detail below. In all cases, the `-h` option prints a full list of parameters. Note that the DNSSEC tools require the keyset files to be in the working directory or the directory specified by the `-d` option.

There must also be communication with the administrators of the parent and/or child zone to transmit keys. A zone's security status must be indicated by the parent zone for a DNSSEC-capable resolver to trust its data. This is done through the presence or absence of a `DS` record at the delegation point.

For other servers to trust data in this zone, they must be statically configured with either this zone's zone key or the zone key of another zone above this one in the DNS tree.

5.8.1 Generating Keys

The `dnssec-keygen` program is used to generate keys.

A secure zone must contain one or more zone keys. The zone keys sign all other records in the zone, as well as the zone keys of any secure delegated zones. Zone keys must have the same name as the zone, have a name type of `ZONE`, and be usable for authentication. It is recommended that zone keys use a cryptographic algorithm designated as "mandatory to implement" by the IETF. Currently there are two algorithms, RSASHA256 and ECDSAP256SHA256; ECDSAP256SHA256 is recommended for current and future deployments.

The following command generates an ECDSAP256SHA256 key for the `child.example` zone:

```
dnssec-keygen -a ECDSAP256SHA256 -n ZONE child.example.
```

Two output files are produced: `Kchild.example.+013+12345.key` and `Kchild.example.+013+12345.private` (where 12345 is an example of a key tag). The key filenames contain the key name (`child.example.`), the algorithm (5 is RSASHA1, 8 is RSASHA256, 13 is ECDSAP256SHA256, 15 is ED25519, etc.), and the key tag (12345 in this case). The private key (in the `.private` file) is used to generate signatures, and the public key (in the `.key` file) is used for signature verification.

To generate another key with the same properties but with a different key tag, repeat the above command.

The `dnssec-keyfromlabel` program is used to get a key pair from a crypto hardware device and build the key files. Its usage is similar to `dnssec-keygen`.

The public keys should be inserted into the zone file by including the `.key` files using `$INCLUDE` statements.

5.8.2 Signing the Zone

The `dnssec-signzone` program is used to sign a zone.

Any `keyset` files corresponding to secure sub-zones should be present. The zone signer generates `NSEC`, `NSEC3`, and `RRSIG` records for the zone, as well as `DS` for the child zones if `-g` is specified. If `-g` is not specified, then DS RRsets for the secure child zones need to be added manually.

By default, all zone keys which have an available private key are used to generate signatures. The following command signs the zone, assuming it is in a file called `zone.child.example`:

```
dnssec-signzone -o child.example zone.child.example
```

One output file is produced: `zone.child.example.signed`. This file should be referenced by `named.conf` as the input file for the zone.

`dnssec-signzone` also produces keyset and dsset files. These are used to provide the parent zone administrators with the `DNSKEY`s (or their corresponding `DS` records) that are the secure entry point to the zone.

5.8.3 Configuring Servers for DNSSEC

To enable `named` to validate answers received from other servers, the `dnssec-validation` option must be set to either `yes` or `auto`.

When `dnssec-validation` is set to `auto`, a trust anchor for the DNS root zone is automatically used. This trust anchor is provided as part of BIND and is kept up to date using **RFC 5011** key management.

When `dnssec-validation` is set to `yes`, DNSSEC validation only occurs if at least one trust anchor has been explicitly configured in `named.conf`, using a `trust-anchors` statement (or the `managed-keys` and `trusted-keys` statements, both deprecated).

When `dnssec-validation` is set to `no`, DNSSEC validation does not occur.

The default is `auto` unless BIND is built with `configure --disable-auto-validation`, in which case the default is `yes`.

The keys specified in `trust-anchors` are copies of DNSKEY RRs for zones that are used to form the first link in the cryptographic chain of trust. Keys configured with the keyword `static-key` or `static-ds` are loaded directly into the table of trust anchors, and can only be changed by altering the configuration. Keys configured with `initial-key` or `initial-ds` are used to initialize **RFC 5011** trust anchor maintenance, and are kept up-to-date automatically after the first time `named` runs.

`trust-anchors` is described in more detail later in this document.

BIND 9 does not verify signatures on load, so zone keys for authoritative zones do not need to be specified in the configuration file.

After DNSSEC is established, a typical DNSSEC configuration looks something like the following. It has one or more public keys for the root, which allows answers from outside the organization to be validated. It also has several keys for parts of the namespace that the organization controls. These are here to ensure that `named` is immune to compromised security in the DNSSEC components of parent zones.

```
trust-anchors {
    /* Root Key */
    "." initial-key 257 3 3 "BNY4wrWM1nCfJ+CXd0rVXyYmobt7sEEfK3clRbGaTwS
                 JxrGkxJWoZu6I7PzJu/E9gx4UC1zGAHlXKdE4zYIpRh
                 aBKnvcC2U9mZhkdUpd1Vso/HAdjNe8LmMlnzY3zy2Xy
                 4klWOADTPzSv9eamj8V18PHGjBLaVtYvk/ln5ZApjYg
                 hf+6fElrmLkdaz MQ2OCnACR817DF4BBa7UR/beDHyp
                 5iWTXWSi6XmoJLbG9Scqc7l70KDqlvXR3M/lUUVRbke
                 g1IPJSidmK3ZyCllh4XSKbje/45SKucHgnwU5jefMtq
                 66gKodQj+MiA21AfUVe7u99WzTLzY3qlxDhxYQQ20FQ
                 97S+LKUTpQcq27R7AT3/V5hRQxScINqwcz4jYqZD2fQ
                 dgxbcDTClU0CRBdiieyLMNzXG3";
    /* Key for our organization's forward zone */
    example.com. static-ds 54135 5 2
→"8EF922C97F1D07B23134440F19682E7519ADDAE180E20B1B1EC52E7F58B2831D"

    /* Key for our reverse zone. */
    2.0.192.IN-ADDRPA.NET. static-key 257 3 5 "AQOnS4xn/IgOUpBPJ3bogzwc
                     xOdNax071L18QqZnQQQAVVr+i
```

```
                    LhGTnNGp3HoWQLUIzKrJVZ3zg
                    gy3WwNT6kZo6c0tszYqbtvchm
                    gQC8CzKojM/W16i6MG/eafGU3
                    siaOdS0yOI6BgPsw+YZdzlYMa
                    IJGf4M4dyoKIhzdZyQ2bYQrjy
                    Q4LB0lC7aOnsMyYKHHYeRvPxj
                    IQXmdqgOJGq+vsevG06zW+1xg
                    YJh9rCIfnm1GX/KMgxLPG2vXT
                    D/RnLX+D3T3UL7HJYHJhAZD5L
                    59VvjSPsZJHeDCUyWYrvPZesZ
                    DIRvhDD52SKvbheeTJUm6Ehkz
                    ytNN2SN96QRk8j/iI8ib";
};

options {
    ...
    dnssec-validation yes;
};
```

Note: None of the keys listed in this example are valid. In particular, the root key is not valid.

When DNSSEC validation is enabled and properly configured, the resolver rejects any answers from signed, secure zones which fail to validate, and returns SERVFAIL to the client.

Responses may fail to validate for any of several reasons, including missing, expired, or invalid signatures, a key which does not match the DS RRset in the parent zone, or an insecure response from a zone which, according to its parent, should have been secure.

Note: When the validator receives a response from an unsigned zone that has a signed parent, it must confirm with the parent that the zone was intentionally left unsigned. It does this by verifying, via signed and validated NSEC/NSEC3 records, that the parent zone contains no DS records for the child.

If the validator *can* prove that the zone is insecure, then the response is accepted. However, if it cannot, the validator must assume an insecure response to be a forgery; it rejects the response and logs an error.

The logged error reads "insecurity proof failed" and "got insecure response; parent indicates it should be secure."

5.9 DNSSEC, Dynamic Zones, and Automatic Signing

5.9.1 Converting From Insecure to Secure

A zone can be changed from insecure to secure in three ways: using a dynamic DNS update, via the `auto-dnssec` zone option, or by setting a DNSSEC policy for the zone with `dnssec-policy`.

For any method, `named` must be configured so that it can see the `K*` files which contain the public and private parts of the keys that are used to sign the zone. These files are generated by `dnssec-keygen`, or created when needed by `named` if `dnssec-policy` is used. Keys should be placed in the key-directory, as specified in `named.conf`:

```
zone example.net {
    type primary;
    update-policy local;
```

```
    file "dynamic/example.net/example.net";
    key-directory "dynamic/example.net";
};
```

If one KSK and one ZSK DNSKEY key have been generated, this configuration causes all records in the zone to be signed with the ZSK, and the DNSKEY RRset to be signed with the KSK. An NSEC chain is generated as part of the initial signing process.

With `dnssec-policy`, it is possible to specify which keys should be KSK and/or ZSK. To sign all records with a key, a CSK must be specified. For example:

```
dnssec-policy csk {
    keys {
        csk lifetime unlimited algorithm 13;
    };
};
```

5.9.2 Dynamic DNS Update Method

To insert the keys via dynamic update:

```
% nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7␣
↪AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y I1m/
↪SAQBxIqMfLtIwqWPdgthsu36azGQAX8=
> update add example.net DNSKEY 257 3 7 AwEAAd/7odU/64o2LGsifbLtQmtO8dFDtTAZXSX2+X3e/
↪UNlq9IHq3Y0 XtC0Iuawl/qkaKVxXe2lo8Ct+dM6UehyCqk=
> send
```

While the update request completes almost immediately, the zone is not completely signed until `named` has had time to "walk" the zone and generate the NSEC and RRSIG records. The NSEC record at the apex is added last, to signal that there is a complete NSEC chain.

To sign using NSEC3 instead of NSEC, add an NSEC3PARAM record to the initial update request. The OPTOUT bit in the NSEC3 chain can be set in the flags field of the NSEC3PARAM record.

```
% nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7␣
↪AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y I1m/
↪SAQBxIqMfLtIwqWPdgthsu36azGQAX8=
> update add example.net DNSKEY 257 3 7 AwEAAd/7odU/64o2LGsifbLtQmtO8dFDtTAZXSX2+X3e/
↪UNlq9IHq3Y0 XtC0Iuawl/qkaKVxXe2lo8Ct+dM6UehyCqk=
> update add example.net NSEC3PARAM 1 1 100 1234567890
> send
```

Again, this update request completes almost immediately; however, the record does not show up until `named` has had a chance to build/remove the relevant chain. A private type record is created to record the state of the operation (see below for more details), and is removed once the operation completes.

While the initial signing and NSEC/NSEC3 chain generation is happening, other updates are possible as well.

5.9.3 Fully Automatic Zone Signing

To enable automatic signing, set a `dnssec-policy` or add the `auto-dnssec` option to the zone statement in `named.conf`. `auto-dnssec` has two possible arguments: `allow` or `maintain`.

With `auto-dnssec allow`, `named` can search the key directory for keys matching the zone, insert them into the zone, and use them to sign the zone. It does so only when it receives an `rndc sign <zonename>`.

`auto-dnssec maintain` includes the above functionality, but also automatically adjusts the zone's DNSKEY records on a schedule according to the keys' timing metadata. (See *dnssec-keygen: DNSSEC key generation tool* and *dnssec-settime: set the key timing metadata for a DNSSEC key* for more information.)

`dnssec-policy` is similar to `auto-dnssec maintain`, but `dnssec-policy` also automatically creates new keys when necessary. In addition, any configuration related to DNSSEC signing is retrieved from the policy, ignoring existing DNSSEC `named.conf` options.

`named` periodically searches the key directory for keys matching the zone; if the keys' metadata indicates that any change should be made to the zone - such as adding, removing, or revoking a key - then that action is carried out. By default, the key directory is checked for changes every 60 minutes; this period can be adjusted with `dnssec-loadkeys-interval`, up to a maximum of 24 hours. The `rndc loadkeys` command forces `named` to check for key updates immediately.

If keys are present in the key directory the first time the zone is loaded, the zone is signed immediately, without waiting for an `rndc sign` or `rndc loadkeys` command. Those commands can still be used when there are unscheduled key changes.

When new keys are added to a zone, the TTL is set to match that of any existing DNSKEY RRset. If there is no existing DNSKEY RRset, the TTL is set to the TTL specified when the key was created (using the `dnssec-keygen -L` option), if any, or to the SOA TTL.

To sign the zone using NSEC3 instead of NSEC, submit an NSEC3PARAM record via dynamic update prior to the scheduled publication and activation of the keys. The OPTOUT bit for the NSEC3 chain can be set in the flags field of the NSEC3PARAM record. The NSEC3PARAM record does not appear in the zone immediately, but it is stored for later reference. When the zone is signed and the NSEC3 chain is completed, the NSEC3PARAM record appears in the zone.

Using the `auto-dnssec` option requires the zone to be configured to allow dynamic updates, by adding an `allow-update` or `update-policy` statement to the zone configuration. If this has not been done, the configuration fails.

5.9.4 Private Type Records

The state of the signing process is signaled by private type records (with a default type value of 65534). When signing is complete, those records with a non-zero initial octet have a non-zero value for the final octet.

If the first octet of a private type record is non-zero, the record indicates either that the zone needs to be signed with the key matching the record, or that all signatures that match the record should be removed. Here are the meanings of the different values of the first octet:

- algorithm (octet 1)
- key id in network order (octet 2 and 3)
- removal flag (octet 4)
- complete flag (octet 5)

Only records flagged as "complete" can be removed via dynamic update; attempts to remove other private type records are silently ignored.

If the first octet is zero (this is a reserved algorithm number that should never appear in a DNSKEY record), the record indicates that changes to the NSEC3 chains are in progress. The rest of the record contains an NSEC3PARAM record, while the flag field tells what operation to perform based on the flag bits:

> 0x01 OPTOUT
>
> 0x80 CREATE
>
> 0x40 REMOVE
>
> 0x20 NONSEC

5.9.5 DNSKEY Rollovers

As with insecure-to-secure conversions, DNSSEC keyrolls can be done in two ways: using a dynamic DNS update, or via the `auto-dnssec` zone option.

5.9.6 Dynamic DNS Update Method

To perform key rollovers via a dynamic update, the `K*` files for the new keys must be added so that `named` can find them. The new DNSKEY RRs can then be added via dynamic update. `named` then causes the zone to be signed with the new keys; when the signing is complete, the private type records are updated so that the last octet is non-zero.

If this is for a KSK, the parent and any trust anchor repositories of the new KSK must be informed.

The maximum TTL in the zone must expire before removing the old DNSKEY. If it is a KSK that is being updated, the DS RRset in the parent must also be updated and its TTL allowed to expire. This ensures that all clients are able to verify at least one signature when the old DNSKEY is removed.

The old DNSKEY can be removed via UPDATE, taking care to specify the correct key. `named` cleans out any signatures generated by the old key after the update completes.

5.9.7 Automatic Key Rollovers

When a new key reaches its activation date (as set by `dnssec-keygen` or `dnssec-settime`), and if the `auto-dnssec` zone option is set to `maintain`, `named` automatically carries out the key rollover. If the key's algorithm has not previously been used to sign the zone, then the zone is fully signed as quickly as possible. However, if the new key replaces an existing key of the same algorithm, the zone is re-signed incrementally, with signatures from the old key replaced with signatures from the new key as their signature validity periods expire. By default, this rollover completes in 30 days, after which it is safe to remove the old key from the DNSKEY RRset.

5.9.8 NSEC3PARAM Rollovers via UPDATE

The new NSEC3PARAM record can be added via dynamic update. When the new NSEC3 chain has been generated, the NSEC3PARAM flag field is set to zero. At that point, the old NSEC3PARAM record can be removed. The old chain is removed after the update request completes.

5.9.9 Converting From NSEC to NSEC3

Add a `nsec3param` option to your `dnssec-policy` and run `rndc reconfig`.

Or use `nsupdate` to add an NSEC3PARAM record.

In both cases, the NSEC3 chain is generated and the NSEC3PARAM record is added before the NSEC chain is destroyed.

5.9.10 Converting From NSEC3 to NSEC

To do this, remove the `nsec3param` option from the `dnssec-policy` and run `rndc reconfig`.

Or use `nsupdate` to remove all NSEC3PARAM records with a zero flag field. The NSEC chain is generated before the NSEC3 chain is removed.

5.9.11 Converting From Secure to Insecure

To convert a signed zone to unsigned using dynamic DNS, delete all the DNSKEY records from the zone apex using `nsupdate`. All signatures, NSEC or NSEC3 chains, and associated NSEC3PARAM records are removed automatically. This takes place after the update request completes.

This requires the `dnssec-secure-to-insecure` option to be set to `yes` in `named.conf`.

In addition, if the `auto-dnssec maintain` zone statement is used, it should be removed or changed to `allow` instead; otherwise it will re-sign.

5.9.12 Periodic Re-signing

In any secure zone which supports dynamic updates, `named` periodically re-signs RRsets which have not been re-signed as a result of some update action. The signature lifetimes are adjusted to spread the re-sign load over time rather than all at once.

5.9.13 NSEC3 and OPTOUT

`named` only supports creating new NSEC3 chains where all the NSEC3 records in the zone have the same OPTOUT state. `named` supports UPDATES to zones where the NSEC3 records in the chain have mixed OPTOUT state. `named` does not support changing the OPTOUT state of an individual NSEC3 record; if the OPTOUT state of an individual NSEC3 needs to be changed, the entire chain must be changed.

5.10 Dynamic Trust Anchor Management

BIND is able to maintain DNSSEC trust anchors using **RFC 5011** key management. This feature allows `named` to keep track of changes to critical DNSSEC keys without any need for the operator to make changes to configuration files.

5.10.1 Validating Resolver

To configure a validating resolver to use **RFC 5011** to maintain a trust anchor, configure the trust anchor using a `trust-anchors` statement and the `initial-key` keyword. Information about this can be found in *trust-anchors Statement Definition and Usage*.

5.10.2 Authoritative Server

To set up an authoritative zone for **RFC 5011** trust anchor maintenance, generate two (or more) key signing keys (KSKs) for the zone. Sign the zone with one of them; this is the "active" KSK. All KSKs which do not sign the zone are "stand-by" keys.

Any validating resolver which is configured to use the active KSK as an **RFC 5011**-managed trust anchor takes note of the stand-by KSKs in the zone's DNSKEY RRset, and stores them for future reference. The resolver rechecks the zone periodically; after 30 days, if the new key is still there, the key is accepted by the resolver as a valid trust anchor for the zone. Anytime after this 30-day acceptance timer has completed, the active KSK can be revoked, and the zone can be "rolled over" to the newly accepted key.

The easiest way to place a stand-by key in a zone is to use the "smart signing" features of `dnssec-keygen` and `dnssec-signzone`. If a key exists with a publication date in the past, but an activation date which is unset or in the future, `dnssec-signzone -S` includes the DNSKEY record in the zone but does not sign with it:

```
$ dnssec-keygen -K keys -f KSK -P now -A now+2y example.net
$ dnssec-signzone -S -K keys example.net
```

To revoke a key, use the command `dnssec-revoke`. This adds the REVOKED bit to the key flags and regenerates the `K*.key` and `K*.private` files.

After revoking the active key, the zone must be signed with both the revoked KSK and the new active KSK. Smart signing takes care of this automatically.

Once a key has been revoked and used to sign the DNSKEY RRset in which it appears, that key is never again accepted as a valid trust anchor by the resolver. However, validation can proceed using the new active key, which was accepted by the resolver when it was a stand-by key.

See **RFC 5011** for more details on key rollover scenarios.

When a key has been revoked, its key ID changes, increasing by 128 and wrapping around at 65535. So, for example, the key "`Kexample.com.+005+10000`" becomes "`Kexample.com.+005+10128`".

If two keys have IDs exactly 128 apart and one is revoked, the two key IDs will collide, causing several problems. To prevent this, `dnssec-keygen` does not generate a new key if another key which may collide is present. This checking only occurs if the new keys are written to the same directory that holds all other keys in use for that zone.

Older versions of BIND 9 did not have this protection. Exercise caution if using key revocation on keys that were generated by previous releases, or if using keys stored in multiple directories or on multiple machines.

It is expected that a future release of BIND 9 will address this problem in a different way, by storing revoked keys with their original unrevoked key IDs.

5.11 PKCS#11 (Cryptoki) Support

Public Key Cryptography Standard #11 (PKCS#11) defines a platform-independent API for the control of hardware security modules (HSMs) and other cryptographic support devices.

BIND 9 is known to work with three HSMs: the AEP Keyper, which has been tested with Debian Linux, Solaris x86, and Windows Server 2003; the Thales nShield, tested with Debian Linux; and the Sun SCA 6000 cryptographic acceleration board, tested with Solaris x86. In addition, BIND can be used with all current versions of SoftHSM, a software-based HSM simulator library produced by the OpenDNSSEC project.

PKCS#11 uses a "provider library": a dynamically loadable library which provides a low-level PKCS#11 interface to drive the HSM hardware. The PKCS#11 provider library comes from the HSM vendor, and it is specific to the HSM to be controlled.

There are two available mechanisms for PKCS#11 support in BIND 9: OpenSSL-based PKCS#11 and native PKCS#11. With OpenSSL-based PKCS#11, BIND uses a modified version of OpenSSL, which loads the provider library and operates the HSM indirectly; any cryptographic operations not supported by the HSM can be carried out by OpenSSL instead. Native PKCS#11 enables BIND to bypass OpenSSL completely; BIND loads the provider library itself, and uses the PKCS#11 API to drive the HSM directly.

5.11.1 Prerequisites

See the documentation provided by the HSM vendor for information about installing, initializing, testing, and troubleshooting the HSM.

5.11.2 Native PKCS#11

Native PKCS#11 mode only works with an HSM capable of carrying out *every* cryptographic operation BIND 9 may need. The HSM's provider library must have a complete implementation of the PKCS#11 API, so that all these functions are accessible. As of this writing, only the Thales nShield HSM and SoftHSMv2 can be used in this fashion. For other HSMs, including the AEP Keyper, Sun SCA 6000, and older versions of SoftHSM, use OpenSSL-based PKCS#11. (Note: Eventually, when more HSMs become capable of supporting native PKCS#11, it is expected that OpenSSL-based PKCS#11 will be deprecated.)

To build BIND with native PKCS#11, configure it as follows:

```
$ cd bind9
$ ./configure --enable-native-pkcs11 \
    --with-pkcs11=provider-library-path
```

This causes all BIND tools, including `named` and the `dnssec-*` and `pkcs11-*` tools, to use the PKCS#11 provider library specified in provider-library-path for cryptography. (The provider library path can be overridden using the `-E` argument in `named` and the `dnssec-*` tools, or the `-m` argument in the `pkcs11-*` tools.)

Building SoftHSMv2

SoftHSMv2, the latest development version of SoftHSM, is available from https://github.com/opendnssec/SoftHSMv2. It is a software library developed by the OpenDNSSEC project (https://www.opendnssec.org) which provides a PKCS#11 interface to a virtual HSM, implemented in the form of an SQLite3 database on the local filesystem. It provides less security than a true HSM, but it allows users to experiment with native PKCS#11 when an HSM is not available. SoftHSMv2 can be configured to use either OpenSSL or the Botan library to perform cryptographic functions, but when using it for native PKCS#11 in BIND, OpenSSL is required.

By default, the SoftHSMv2 configuration file is `prefix/etc/softhsm2.conf` (where `prefix` is configured at compile time). This location can be overridden by the SOFTHSM2_CONF environment variable. The SoftHSMv2 cryptographic store must be installed and initialized before using it with BIND.

```
$  cd SoftHSMv2
$  configure --with-crypto-backend=openssl --prefix=/opt/pkcs11/usr
$  make
$  make install
$  /opt/pkcs11/usr/bin/softhsm-util --init-token 0 --slot 0 --label softhsmv2
```

5.11.3 OpenSSL-based PKCS#11

OpenSSL-based PKCS#11 uses engine_pkcs11 OpenSSL engine from libp11 project.

For more information, see https://gitlab.isc.org/isc-projects/bind9/-/wikis/BIND-9-PKCS11

5.11.4 PKCS#11 Tools

BIND 9 includes a minimal set of tools to operate the HSM, including `pkcs11-keygen` to generate a new key pair within the HSM, `pkcs11-list` to list objects currently available, `pkcs11-destroy` to remove objects, and `pkcs11-tokens` to list available tokens.

In UNIX/Linux builds, these tools are built only if BIND 9 is configured with the `--with-pkcs11` option. (Note: If `--with-pkcs11` is set to `yes`, rather than to the path of the PKCS#11 provider, the tools are built but the provider is left undefined. Use the `-m` option or the PKCS11_PROVIDER environment variable to specify the path to the provider.)

5.11.5 Using the HSM

For OpenSSL-based PKCS#11, the runtime environment must first be set up so the OpenSSL and PKCS#11 libraries can be loaded:

```
$ export LD_LIBRARY_PATH=/opt/pkcs11/usr/lib:${LD_LIBRARY_PATH}
```

This causes `named` and other binaries to load the OpenSSL library from `/opt/pkcs11/usr/lib`, rather than from the default location. This step is not necessary when using native PKCS#11.

Some HSMs require other environment variables to be set. For example, when operating an AEP Keyper, the location of the "machine" file, which stores information about the Keyper for use by the provider library, must be specified. If the machine file is in `/opt/Keyper/PKCS11Provider/machine`, use:

```
$ export KEYPER_LIBRARY_PATH=/opt/Keyper/PKCS11Provider
```

Such environment variables must be set when running any tool that uses the HSM, including `pkcs11-keygen`, `pkcs11-list`, `pkcs11-destroy`, `dnssec-keyfromlabel`, `dnssec-signzone`, `dnssec-keygen`, and `named`.

HSM keys can now be created and used. In this case, we will create a 2048-bit key and give it the label "sample-ksk":

```
$ pkcs11-keygen -b 2048 -l sample-ksk
```

To confirm that the key exists:

```
$ pkcs11-list
Enter PIN:
object[0]: handle 2147483658 class 3 label[8] 'sample-ksk' id[0]
object[1]: handle 2147483657 class 2 label[8] 'sample-ksk' id[0]
```

Before using this key to sign a zone, we must create a pair of BIND 9 key files. The `dnssec-keyfromlabel` utility does this. In this case, we are using the HSM key "sample-ksk" as the key-signing key for "example.net":

```
$ dnssec-keyfromlabel -l sample-ksk -f KSK example.net
```

The resulting K*.key and K*.private files can now be used to sign the zone. Unlike normal K* files, which contain both public and private key data, these files contain only the public key data, plus an identifier for the private key which remains stored within the HSM. Signing with the private key takes place inside the HSM.

To generate a second key in the HSM for use as a zone-signing key, follow the same procedure above, using a different keylabel, a smaller key size, and omitting `-f KSK` from the `dnssec-keyfromlabel` arguments:

```
$ pkcs11-keygen -b 1024 -l sample-zsk
$ dnssec-keyfromlabel -l sample-zsk example.net
```

Alternatively, a conventional on-disk key can be generated using `dnssec-keygen`:

```
$ dnssec-keygen example.net
```

This provides less security than an HSM key, but since HSMs can be slow or cumbersome to use for security reasons, it may be more efficient to reserve HSM keys for use in the less frequent key-signing operation. The zone-signing key can be rolled more frequently, if desired, to compensate for a reduction in key security. (Note: When using native PKCS#11, there is no speed advantage to using on-disk keys, as cryptographic operations are done by the HSM.)

Now the zone can be signed. Please note that, if the -S option is not used for `dnssec-signzone`, the contents of both `K*.key` files must be added to the zone master file before signing it.

```
$ dnssec-signzone -S example.net
Enter PIN:
Verifying the zone using the following algorithms:
NSEC3RSASHA1.
Zone signing complete:
Algorithm: NSEC3RSASHA1: ZSKs: 1, KSKs: 1 active, 0 revoked, 0 stand-by
example.net.signed
```

5.11.6 Specifying the Engine on the Command Line

When using OpenSSL-based PKCS#11, the "engine" to be used by OpenSSL can be specified in `named` and all of the BIND `dnssec-*` tools by using the `-E <engine>` command line option. If BIND 9 is built with the `--with-pkcs11` option, this option defaults to "pkcs11". Specifying the engine is generally not necessary unless a different OpenSSL engine is used.

To disable use of the "pkcs11" engine - for troubleshooting purposes, or because the HSM is unavailable - set the engine to the empty string. For example:

```
$ dnssec-signzone -E '' -S example.net
```

This causes `dnssec-signzone` to run as if it were compiled without the `--with-pkcs11` option.

When built with native PKCS#11 mode, the "engine" option has a different meaning: it specifies the path to the PKCS#11 provider library. This may be useful when testing a new provider library.

5.11.7 Running `named` With Automatic Zone Re-signing

For `named` to dynamically re-sign zones using HSM keys, and/or to sign new records inserted via nsupdate, `named` must have access to the HSM PIN. In OpenSSL-based PKCS#11, this is accomplished by placing the PIN into the `openssl.cnf` file (in the above examples, `/opt/pkcs11/usr/ssl/openssl.cnf`).

The location of the openssl.cnf file can be overridden by setting the `OPENSSL_CONF` environment variable before running `named`.

Here is a sample `openssl.cnf`:

```
openssl_conf = openssl_def
[ openssl_def ]
engines = engine_section
[ engine_section ]
pkcs11 = pkcs11_section
[ pkcs11_section ]
PIN = <PLACE PIN HERE>
```

This also allows the `dnssec-\*` tools to access the HSM without PIN entry. (The `pkcs11-\*` tools access the HSM directly, not via OpenSSL, so a PIN is still required to use them.)

In native PKCS#11 mode, the PIN can be provided in a file specified as an attribute of the key's label. For example, if a key had the label `pkcs11:object=local-zsk;pin-source=/etc/hsmpin`, then the PIN would be read from the file `/etc/hsmpin`.

> **Warning:** Placing the HSM's PIN in a text file in this manner may reduce the security advantage of using an HSM. Use caution when configuring the system in this way.

5.12 Dynamically Loadable Zones (DLZ)

Dynamically Loadable Zones (DLZ) are an extension to BIND 9 that allows zone data to be retrieved directly from an external database. There is no required format or schema. DLZ drivers exist for several different database backends, including PostgreSQL, MySQL, and LDAP, and can be written for any other.

Historically, DLZ drivers had to be statically linked with the `named` binary and were turned on via a configure option at compile time (for example, `configure --with-dlz-ldap`). The drivers provided in the BIND 9 tarball in `contrib/dlz/drivers` are still linked this way.

In BIND 9.8 and higher, it is possible to link some DLZ modules dynamically at runtime, via the DLZ "dlopen" driver, which acts as a generic wrapper around a shared object implementing the DLZ API. The "dlopen" driver is linked into `named` by default, so configure options are no longer necessary when using these dynamically linkable drivers; they are still needed for the older drivers in `contrib/dlz/drivers`.

The DLZ module provides data to `named` in text format, which is then converted to DNS wire format by `named`. This conversion, and the lack of any internal caching, places significant limits on the query performance of DLZ modules. Consequently, DLZ is not recommended for use on high-volume servers. However, it can be used in a hidden primary

configuration, with secondaries retrieving zone updates via AXFR. Note, however, that DLZ has no built-in support for DNS notify; secondary servers are not automatically informed of changes to the zones in the database.

5.12.1 Configuring DLZ

A DLZ database is configured with a `dlz` statement in `named.conf`:

```
dlz example {
database "dlopen driver.so args";
search yes;
};
```

This specifies a DLZ module to search when answering queries; the module is implemented in `driver.so` and is loaded at runtime by the dlopen DLZ driver. Multiple `dlz` statements can be specified; when answering a query, all DLZ modules with `search` set to `yes` are queried to see whether they contain an answer for the query name. The best available answer is returned to the client.

The `search` option in the above example can be omitted, because `yes` is the default value.

If `search` is set to `no`, this DLZ module is *not* searched for the best match when a query is received. Instead, zones in this DLZ must be separately specified in a zone statement. This allows users to configure a zone normally using standard zone-option semantics, but specify a different database backend for storage of the zone's data. For example, to implement NXDOMAIN redirection using a DLZ module for backend storage of redirection rules:

```
dlz other {
      database "dlopen driver.so args";
      search no;
};

zone "." {
      type redirect;
      dlz other;
};
```

5.12.2 Sample DLZ Driver

For guidance in the implementation of DLZ modules, the directory `contrib/dlz/example` contains a basic dynamically linkable DLZ module - i.e., one which can be loaded at runtime by the "dlopen" DLZ driver. The example sets up a single zone, whose name is passed to the module as an argument in the `dlz` statement:

```
dlz other {
      database "dlopen driver.so example.nil";
};
```

In the above example, the module is configured to create a zone "example.nil", which can answer queries and AXFR requests and accept DDNS updates. At runtime, prior to any updates, the zone contains an SOA, NS, and a single A record at the apex:

```
example.nil.   3600    IN     SOA    example.nil. hostmaster.example.nil. (
                       123 900 600 86400 3600
               )
example.nil.   3600    IN     NS     example.nil.
example.nil.   1800    IN     A      10.53.0.1
```

The sample driver can retrieve information about the querying client and alter its response on the basis of this information. To demonstrate this feature, the example driver responds to queries for "source-addr.`zonename`>/TXT" with the source address of the query. Note, however, that this record will *not* be included in AXFR or ANY responses. Normally, this feature is used to alter responses in some other fashion, e.g., by providing different address records for a particular name depending on the network from which the query arrived.

Documentation of the DLZ module API can be found in `contrib/dlz/example/README`. This directory also contains the header file `dlz_minimal.h`, which defines the API and should be included by any dynamically linkable DLZ module.

5.13 Dynamic Database (DynDB)

Dynamic Database, or DynDB, is an extension to BIND 9 which, like DLZ (see *Dynamically Loadable Zones (DLZ)*), allows zone data to be retrieved from an external database. Unlike DLZ, a DynDB module provides a full-featured BIND zone database interface. Where DLZ translates DNS queries into real-time database lookups, resulting in relatively poor query performance, and is unable to handle DNSSEC-signed data due to its limited API, a DynDB module can pre-load an in-memory database from the external data source, providing the same performance and functionality as zones served natively by BIND.

A DynDB module supporting LDAP has been created by Red Hat and is available from https://pagure.io/bind-dyndb-ldap.

A sample DynDB module for testing and developer guidance is included with the BIND source code, in the directory `bin/tests/system/dyndb/driver`.

5.13.1 Configuring DynDB

A DynDB database is configured with a `dyndb` statement in `named.conf`:

```
dyndb example "driver.so" {
    parameters
};
```

The file `driver.so` is a DynDB module which implements the full DNS database API. Multiple `dyndb` statements can be specified, to load different drivers or multiple instances of the same driver. Zones provided by a DynDB module are added to the view's zone table, and are treated as normal authoritative zones when BIND responds to queries. Zone configuration is handled internally by the DynDB module.

The parameters are passed as an opaque string to the DynDB module's initialization routine. Configuration syntax differs depending on the driver.

5.13.2 Sample DynDB Module

For guidance in the implementation of DynDB modules, the directory `bin/tests/system/dyndb/driver` contains a basic DynDB module. The example sets up two zones, whose names are passed to the module as arguments in the `dyndb` statement:

```
dyndb sample "sample.so" { example.nil. arpa. };
```

In the above example, the module is configured to create a zone, "example.nil", which can answer queries and AXFR requests and accept DDNS updates. At runtime, prior to any updates, the zone contains an SOA, NS, and a single A record at the apex:

```
example.nil.    86400    IN    SOA    example.nil. example.nil. (
                                            0 28800 7200 604800 86400
                                    )
example.nil.    86400    IN    NS     example.nil.
example.nil.    86400    IN    A      127.0.0.1
```

When the zone is updated dynamically, the DynDB module determines whether the updated RR is an address (i.e., type A or AAAA); if so, it automatically updates the corresponding PTR record in a reverse zone. Note that updates are not stored permanently; all updates are lost when the server is restarted.

5.14 Catalog Zones

A "catalog zone" is a special DNS zone that contains a list of other zones to be served, along with their configuration parameters. Zones listed in a catalog zone are called "member zones." When a catalog zone is loaded or transferred to a secondary server which supports this functionality, the secondary server creates the member zones automatically. When the catalog zone is updated (for example, to add or delete member zones, or change their configuration parameters), those changes are immediately put into effect. Because the catalog zone is a normal DNS zone, these configuration changes can be propagated using the standard AXFR/IXFR zone transfer mechanism.

Catalog zones' format and behavior are specified as an Internet draft for interoperability among DNS implementations. The latest revision of the DNS catalog zones draft can be found here: https://datatracker.ietf.org/doc/draft-toorop-dnsop-dns-catalog-zones/ .

5.14.1 Principle of Operation

Normally, if a zone is to be served by a secondary server, the `named.conf` file on the server must list the zone, or the zone must be added using `rndc addzone`. In environments with a large number of secondary servers, and/or where the zones being served are changing frequently, the overhead involved in maintaining consistent zone configuration on all the secondary servers can be significant.

A catalog zone is a way to ease this administrative burden: it is a DNS zone that lists member zones that should be served by secondary servers. When a secondary server receives an update to the catalog zone, it adds, removes, or reconfigures member zones based on the data received.

To use a catalog zone, it must first be set up as a normal zone on both the primary and secondary servers that are configured to use it. It must also be added to a `catalog-zones` list in the `options` or `view` statement in `named.conf`. This is comparable to the way a policy zone is configured as a normal zone and also listed in a `response-policy` statement.

To use the catalog zone feature to serve a new member zone:

- Set up the member zone to be served on the primary as normal. This can be done by editing `named.conf` or by running `rndc addzone`.

- Add an entry to the catalog zone for the new member zone. This can be done by editing the catalog zone's zone file and running `rndc reload`, or by updating the zone using `nsupdate`.

The change to the catalog zone is propagated from the primary to all secondaries using the normal AXFR/IXFR mechanism. When the secondary receives the update to the catalog zone, it detects the entry for the new member zone, creates an instance of that zone on the secondary server, and points that instance to the `masters` specified in the catalog zone data. The newly created member zone is a normal secondary zone, so BIND immediately initiates a transfer of zone contents from the primary. Once complete, the secondary starts serving the member zone.

Removing a member zone from a secondary server requires only deleting the member zone's entry in the catalog zone; the change to the catalog zone is propagated to the secondary server using the normal AXFR/IXFR transfer mechanism. The secondary server, on processing the update, notices that the member zone has been removed, stops serving the zone,

and removes it from its list of configured zones. However, removing the member zone from the primary server must be done by editing the configuration file or running `rndc delzone`.

5.14.2 Configuring Catalog Zones

Catalog zones are configured with a `catalog-zones` statement in the `options` or `view` section of `named.conf`. For example:

```
catalog-zones {
    zone "catalog.example"
        default-masters { 10.53.0.1; }
        in-memory no
        zone-directory "catzones"
        min-update-interval 10;
};
```

This statement specifies that the zone `catalog.example` is a catalog zone. This zone must be properly configured in the same view. In most configurations, it would be a secondary zone.

The options following the zone name are not required, and may be specified in any order.

default-masters This option defines the default primaries for member zones listed in a catalog zone, and can be overridden by options within a catalog zone. If no such options are included, then member zones transfer their contents from the servers listed in this option.

in-memory This option, if set to `yes`, causes member zones to be stored only in memory. This is functionally equivalent to configuring a secondary zone without a `file` option. The default is `no`; member zones' content is stored locally in a file whose name is automatically generated from the view name, catalog zone name, and member zone name.

zone-directory This option causes local copies of member zones' zone files to be stored in the specified directory, if `in-memory` is not set to `yes`. The default is to store zone files in the server's working directory. A non-absolute pathname in `zone-directory` is assumed to be relative to the working directory.

min-update-interval This option sets the minimum interval between updates to catalog zones, in seconds. If an update to a catalog zone (for example, via IXFR) happens less than `min-update-interval` seconds after the most recent update, the changes are not carried out until this interval has elapsed. The default is 5 seconds.

Catalog zones are defined on a per-view basis. Configuring a non-empty `catalog-zones` statement in a view automatically turns on `allow-new-zones` for that view. This means that `rndc addzone` and `rndc delzone` also work in any view that supports catalog zones.

5.14.3 Catalog Zone Format

A catalog zone is a regular DNS zone; therefore, it must have a single `SOA` and at least one `NS` record.

A record stating the version of the catalog zone format is also required. If the version number listed is not supported by the server, then a catalog zone may not be used by that server.

```
catalog.example.    IN SOA . . 2016022901 900 600 86400 1
catalog.example.    IN NS nsexample.
version.catalog.example.    IN TXT "1"
```

Note that this record must have the domain name `version.catalog-zone-name`. The data stored in a catalog zone is indicated by the domain name label immediately before the catalog zone domain.

Catalog zone options can be set either globally for the whole catalog zone or for a single member zone. Global options override the settings in the configuration file, and member zone options override global options.

Global options are set at the apex of the catalog zone, e.g.:

```
masters.catalog.example.    IN AAAA 2001:db8::1
```

BIND currently supports the following options:

- A simple `masters` definition:

```
masters.catalog.example.    IN A 192.0.2.1
```

  This option defines a primary server for the member zones, which can be either an A or AAAA record. If multiple primaries are set, the order in which they are used is random.

- A `masters` with a TSIG key defined:

```
label.masters.catalog.example.    IN A 192.0.2.2
label.masters.catalog.example.    IN TXT "tsig_key_name"
```

  This option defines a primary server for the member zone with a TSIG key set. The TSIG key must be configured in the configuration file. `label` can be any valid DNS label.

- `allow-query` and `allow-transfer` ACLs:

```
allow-query.catalog.example.   IN APL 1:10.0.0.1/24
allow-transfer.catalog.example.    IN APL !1:10.0.0.1/32 1:10.0.0.0/24
```

  These options are the equivalents of `allow-query` and `allow-transfer` in a zone declaration in the `named.conf` configuration file. The ACL is processed in order; if there is no match to any rule, the default policy is to deny access. For the syntax of the APL RR, see **RFC 3123**.

A member zone is added by including a `PTR` resource record in the `zones` sub-domain of the catalog zone. The record label is a `SHA-1` hash of the member zone name in wire format. The target of the PTR record is the member zone name. For example, to add the member zone `domain.example`:

```
5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN PTR domain.example.
```

The hash is necessary to identify options for a specific member zone. The member zone-specific options are defined the same way as global options, but in the member zone subdomain:

```
masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN A 192.0.2.2
label.masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN AAAA␣
→2001:db8::2
label.masters.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN TXT
→"tsig_key"
allow-query.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN APL␣
→1:10.0.0.0/24
```

Options defined for a specific zone override the global options defined in the catalog zone. These in turn override the global options defined in the `catalog-zones` statement in the configuration file.

Note that none of the global records for an option are inherited if any records are defined for that option for the specific zone. For example, if the zone had a `masters` record of type A but not AAAA, it would *not* inherit the type AAAA record from the global option.

5.15 IPv6 Support in BIND 9

BIND 9 fully supports all currently defined forms of IPv6 name-to-address and address-to-name lookups. It also uses IPv6 addresses to make queries when running on an IPv6-capable system.

For forward lookups, BIND 9 supports only AAAA records. **RFC 3363** deprecated the use of A6 records, and client-side support for A6 records was accordingly removed from BIND 9. However, authoritative BIND 9 name servers still load zone files containing A6 records correctly, answer queries for A6 records, and accept zone transfer for a zone containing A6 records.

For IPv6 reverse lookups, BIND 9 supports the traditional "nibble" format used in the `ip6.arpa` domain, as well as the older, deprecated `ip6.int` domain. Older versions of BIND 9 supported the "binary label" (also known as "bitstring") format, but support of binary labels has been completely removed per **RFC 3363**. Many applications in BIND 9 do not understand the binary label format at all anymore, and return an error if one is given. In particular, an authoritative BIND 9 name server will not load a zone file containing binary labels.

5.15.1 Address Lookups Using AAAA Records

The IPv6 AAAA record is a parallel to the IPv4 A record, and, unlike the deprecated A6 record, specifies the entire IPv6 address in a single record. For example:

```
$ORIGIN example.com.
host          3600    IN      AAAA    2001:db8::1
```

Use of IPv4-in-IPv6 mapped addresses is not recommended. If a host has an IPv4 address, use an A record, not a AAAA, with `::ffff:192.168.42.1` as the address.

5.15.2 Address-to-Name Lookups Using Nibble Format

When looking up an address in nibble format, the address components are simply reversed, just as in IPv4, and `ip6. arpa.` is appended to the resulting name. For example, the following commands produce a reverse name lookup for a host with address `2001:db8::1`:

```
$ORIGIN 0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0  14400    IN     PTR    (
                   host.example.com. )
```

BIND 9 SECURITY CONSIDERATIONS

6.1 Access Control Lists

Access Control Lists (ACLs) are address match lists that can be set up and nicknamed for future use in `allow-notify`, `allow-query`, `allow-query-on`, `allow-recursion`, `blackhole`, `allow-transfer`, `match-clients`, etc.

ACLs give users finer control over who can access the name server, without cluttering up configuration files with huge lists of IP addresses.

It is a *good idea* to use ACLs, and to control access. Limiting access to the server by outside parties can help prevent spoofing and denial of service (DoS) attacks against the server.

ACLs match clients on the basis of up to three characteristics: 1) The client's IP address; 2) the TSIG or SIG(0) key that was used to sign the request, if any; and 3) an address prefix encoded in an EDNS Client-Subnet option, if any.

Here is an example of ACLs based on client addresses:

```
// Set up an ACL named "bogusnets" that blocks
// RFC1918 space and some reserved space, which is
// commonly used in spoofing attacks.
acl bogusnets {
    0.0.0.0/8;  192.0.2.0/24; 224.0.0.0/3;
    10.0.0.0/8; 172.16.0.0/12; 192.168.0.0/16;
};

// Set up an ACL called our-nets. Replace this with the
// real IP numbers.
acl our-nets { x.x.x.x/24; x.x.x.x/21; };
options {
  ...
  ...
  allow-query { our-nets; };
  allow-recursion { our-nets; };
  ...
  blackhole { bogusnets; };
  ...
};

zone "example.com" {
  type primary;
  file "m/example.com";
  allow-query { any; };
};
```

This allows authoritative queries for `example.com` from any address, but recursive queries only from the networks specified in `our-nets`, and no queries at all from the networks specified in `bogusnets`.

In addition to network addresses and prefixes, which are matched against the source address of the DNS request, ACLs may include `key` elements, which specify the name of a TSIG or SIG(0) key.

When BIND 9 is built with GeoIP support, ACLs can also be used for geographic access restrictions. This is done by specifying an ACL element of the form: `geoip db database field value`.

The `field` parameter indicates which field to search for a match. Available fields are `country`, `region`, `city`, `continent`, `postal` (postal code), `metro` (metro code), `area` (area code), `tz` (timezone), `isp`, `asnum`, and `domain`.

`value` is the value to search for within the database. A string may be quoted if it contains spaces or other special characters. An `asnum` search for autonomous system number can be specified using the string "ASNNNN" or the integer NNNN. If a `country` search is specified with a string that is two characters long, it must be a standard ISO-3166-1 two-letter country code; otherwise, it is interpreted as the full name of the country. Similarly, if `region` is the search term and the string is two characters long, it is treated as a standard two-letter state or province abbreviation; otherwise, it is treated as the full name of the state or province.

The `database` field indicates which GeoIP database to search for a match. In most cases this is unnecessary, because most search fields can only be found in a single database. However, searches for `continent` or `country` can be answered from either the `city` or `country` databases, so for these search types, specifying a `database` forces the query to be answered from that database and no other. If a `database` is not specified, these queries are first answered from the `city` database if it is installed, and then from the `country` database if it is installed. Valid database names are `country`, `city`, `asnum`, `isp`, and `domain`.

Some example GeoIP ACLs:

```
geoip country US;
geoip country JP;
geoip db country country Canada;
geoip region WA;
geoip city "San Francisco";
geoip region Oklahoma;
geoip postal 95062;
geoip tz "America/Los_Angeles";
geoip org "Internet Systems Consortium";
```

ACLs use a "first-match" logic rather than "best-match"; if an address prefix matches an ACL element, then that ACL is considered to have matched even if a later element would have matched more specifically. For example, the ACL `{ 10/8; !10.0.0.1; }` would actually match a query from 10.0.0.1, because the first element indicates that the query should be accepted, and the second element is ignored.

When using "nested" ACLs (that is, ACLs included or referenced within other ACLs), a negative match of a nested ACL tells the containing ACL to continue looking for matches. This enables complex ACLs to be constructed, in which multiple client characteristics can be checked at the same time. For example, to construct an ACL which allows a query only when it originates from a particular network *and* only when it is signed with a particular key, use:

```
allow-query { !{ !10/8; any; }; key example; };
```

Within the nested ACL, any address that is *not* in the 10/8 network prefix is rejected, which terminates the processing of the ACL. Any address that *is* in the 10/8 network prefix is accepted, but this causes a negative match of the nested ACL, so the containing ACL continues processing. The query is accepted if it is signed by the key `example`, and rejected otherwise. The ACL, then, only matches when *both* conditions are true.

6.2 `Chroot` and `Setuid`

On Unix servers, it is possible to run BIND in a *chrooted* environment (using the `chroot()` function) by specifying the `-t` option for `named`. This can help improve system security by placing BIND in a "sandbox," which limits the damage done if a server is compromised.

Another useful feature in the Unix version of BIND is the ability to run the daemon as an unprivileged user (`-u` user). We suggest running as an unprivileged user when using the `chroot` feature.

Here is an example command line to load BIND in a `chroot` sandbox, `/var/named`, and to run `named setuid` to user 202:

```
/usr/local/sbin/named -u 202 -t /var/named
```

6.2.1 The `chroot` Environment

For a `chroot` environment to work properly in a particular directory (for example, `/var/named`), the environment must include everything BIND needs to run. From BIND's point of view, `/var/named` is the root of the filesystem; the values of options like `directory` and `pid-file` must be adjusted to account for this.

Unlike with earlier versions of BIND, `named` does *not* typically need to be compiled statically, nor do shared libraries need to be installed under the new root. However, depending on the operating system, it may be necessary to set up locations such as `/dev/zero`, `/dev/random`, `/dev/log`, and `/etc/localtime`.

6.2.2 Using the `setuid` Function

Prior to running the `named` daemon, use the `touch` utility (to change file access and modification times) or the `chown` utility (to set the user id and/or group id) on files where BIND should write.

Note: If the `named` daemon is running as an unprivileged user, it cannot bind to new restricted ports if the server is reloaded.

6.3 Dynamic Update Security

Access to the dynamic update facility should be strictly limited. In earlier versions of BIND, the only way to do this was based on the IP address of the host requesting the update, by listing an IP address or network prefix in the `allow-update` zone option. This method is insecure, since the source address of the update UDP packet is easily forged. Also note that if the IP addresses allowed by the `allow-update` option include the address of a secondary server which performs forwarding of dynamic updates, the primary can be trivially attacked by sending the update to the secondary, which forwards it to the primary with its own source IP address - causing the primary to approve it without question.

For these reasons, we strongly recommend that updates be cryptographically authenticated by means of transaction signatures (TSIG). That is, the `allow-update` option should list only TSIG key names, not IP addresses or network prefixes. Alternatively, the `update-policy` option can be used.

Some sites choose to keep all dynamically updated DNS data in a subdomain and delegate that subdomain to a separate zone. This way, the top-level zone containing critical data, such as the IP addresses of public web and mail servers, need not allow dynamic updates at all.

TROUBLESHOOTING

7.1 Common Problems

7.1.1 It's Not Working; How Can I Figure Out What's Wrong?

The best solution to installation and configuration issues is to take preventive measures by setting up logging files beforehand. The log files provide hints and information that can be used to identify anything that went wrong and fix the problem.

7.1.2 EDNS Compliance Issues

EDNS (Extended DNS) is a standard that was first specified in 1999. It is required for DNSSEC validation, DNS COOKIE options, and other features. There are broken and outdated DNS servers and firewalls still in use which misbehave when queried with EDNS; for example, they may drop EDNS queries rather than replying with FORMERR. BIND and other recursive name servers have traditionally employed workarounds in this situation, retrying queries in different ways and eventually falling back to plain DNS queries without EDNS.

Such workarounds cause unnecessary resolution delays, increase code complexity, and prevent deployment of new DNS features. In February 2019, all major DNS software vendors removed these workarounds; see https://dnsflagday.net/2019 for further details. This change was implemented in BIND as of release 9.14.0.

As a result, some domains may be non-resolvable without manual intervention. In these cases, resolution can be restored by adding `server` clauses for the offending servers, or by specifying `edns no` or `send-cookie no`, depending on the specific noncompliance.

To determine which `server` clause to use, run the following commands to send queries to the authoritative servers for the broken domain:

```
dig soa <zone> @<server> +dnssec
dig soa <zone> @<server> +dnssec +nocookie
dig soa <zone> @<server> +noedns
```

If the first command fails but the second succeeds, the server most likely needs `send-cookie no`. If the first two fail but the third succeeds, then the server needs EDNS to be fully disabled with `edns no`.

Please contact the administrators of noncompliant domains and encourage them to upgrade their broken DNS servers.

7.2 Incrementing and Changing the Serial Number

Zone serial numbers are just numbers — they are not date-related. However, many people set them to a number that represents a date, usually of the form YYYYMMDDRR. Occasionally they make a mistake and set the serial number to a date in the future, then try to correct it by setting it to the current date. This causes problems because serial numbers are used to indicate that a zone has been updated. If the serial number on the secondary server is lower than the serial number on the primary, the secondary server attempts to update its copy of the zone.

Setting the serial number to a lower number on the primary server than the one on the secondary server means that the secondary will not perform updates to its copy of the zone.

The solution to this is to add 2147483647 (2^31-1) to the number, reload the zone and make sure all secondaries have updated to the new zone serial number, then reset it to the desired number and reload the zone again.

7.3 Where Can I Get Help?

The BIND-users mailing list, at https://lists.isc.org/mailman/listinfo/bind-users, is an excellent resource for peer user support. In addition, ISC maintains a Knowledgebase of helpful articles at https://kb.isc.org.

Internet Systems Consortium (ISC) offers annual support agreements for BIND 9, ISC DHCP, and Kea DHCP. All paid support contracts include advance security notifications; some levels include service level agreements (SLAs), premium software features, and increased priority on bug fixes and feature requests.

Please contact info@isc.org or visit https://www.isc.org/contact/ for more information.

RELEASE NOTES

Contents

8.1 Introduction

BIND 9.16 (Extended Support Version) is a stable branch of BIND. This document summarizes significant changes since the last production release on that branch. Please see the CHANGES file for a more detailed list of changes and bug fixes.

8.2 Note on Version Numbering

As of BIND 9.13/9.14, BIND has adopted the "odd-unstable/even-stable" release numbering convention. BIND 9.16 contains new features that were added during the BIND 9.15 development process. Henceforth, the 9.16 branch will be limited to bug fixes, and new feature development will proceed in the unstable 9.17 branch.

8.3 Supported Platforms

See the *Supported Platforms* section in the *BIND Resource Requirements* chapter.

8.4 Download

The latest versions of BIND 9 software can always be found at https://www.isc.org/download/. There you will find additional information about each release, source code, and pre-compiled versions for Microsoft Windows operating systems.

8.5 Notes for BIND 9.16.28

8.5.1 New Features

- Add a new configuration option `reuseport` to disable load balancing on sockets in situations where processing of Response Policy Zones (RPZ), Catalog Zones, or large zone transfers can cause service disruptions. See the BIND 9 ARM for more detail. [GL #3249]

8.5.2 Bug Fixes

- Invalid `dnssec-policy` definitions, where the defined keys did not cover both KSK and ZSK roles for a given algorithm, were being accepted. These are now checked, and the `dnssec-policy` is rejected if both roles are not present for all algorithms in use. [GL #3142]

- Handling of TCP write timeouts has been improved to track the timeout for each TCP write separately, leading to a faster connection teardown in case the other party is not reading the data. [GL #3200]

8.6 Notes for BIND 9.16.27

8.6.1 Security Fixes

- The rules for acceptance of records into the cache have been tightened to prevent the possibility of poisoning if forwarders send records outside the configured bailiwick. (CVE-2021-25220)

  ISC would like to thank Xiang Li, Baojun Liu, and Chaoyi Lu from Network and Information Security Lab, Tsinghua University, and Changgen Zou from Qi An Xin Group Corp. for bringing this vulnerability to our attention. [GL #2950]

- TCP connections with `keep-response-order` enabled could leave the TCP sockets in the `CLOSE_WAIT` state when the client did not properly shut down the connection. (CVE-2022-0396) [GL #3112]

8.6.2 Feature Changes

- DEBUG(1)-level messages were added when starting and ending the BIND 9 task-exclusive mode that stops normal DNS operation (e.g. for reconfiguration, interface scans, and other events that require exclusive access to a shared resource). [GL #3137]

8.6.3 Bug Fixes

- The `max-transfer-time-out` and `max-transfer-idle-out` options were not implemented when the BIND 9 networking stack was refactored in 9.16. The missing functionality has been re-implemented and outgoing zone transfers now time out properly when not progressing. **[GL #1897]**

- TCP connections could hang indefinitely if the other party did not read sent data, causing the TCP write buffers to fill. This has been fixed by adding a "write" timer. Connections that are hung while writing now time out after the `tcp-idle-timeout` period has elapsed. **[GL #3132]**

- The statistics counter representing the current number of clients awaiting recursive resolution results (`RecursClients`) could be miscalculated in certain resolution scenarios, potentially causing the value of the counter to drop below zero. This has been fixed. **[GL #3147]**

8.7 Notes for BIND 9.16.26

8.7.1 Feature Changes

- The DLZ API has been updated: EDNS Client-Subnet (ECS) options sent by a client are now included in the client information sent to DLZ modules when processing queries. **[GL #3082]**

8.7.2 Bug Fixes

- Previously, `recvmmsg` support was enabled in libuv 1.35.0 and 1.36.0, but not in libuv versions 1.37.0 or greater, reducing the maximum query-response performance. This has been fixed. **[GL #3095]**

- A failed view configuration during a `named` reconfiguration procedure could cause inconsistencies in BIND internal structures, causing a crash or other unexpected errors. This has been fixed. **[GL #3060]**

- Previously, `named` logged a "quota reached" message when it hit its hard quota on the number of connections. That message was accidentally removed but has now been restored. **[GL #3125]**

- Build errors were introduced in some DLZ modules due to an incomplete change in the previous release. This has been fixed. **[GL #3111]**

8.8 Notes for BIND 9.16.25

8.8.1 Feature Changes

- Overall memory use by `named` has been optimized and reduced, especially on systems with many CPU cores. The default memory allocator has been switched from `internal` to `external`. A new command-line option `-M internal` allows `named` to be started with the old internal memory allocator. **[GL #2398]**

8.8.2 Bug Fixes

- On FreeBSD, TCP connections leaked a small amount of heap memory, leading to an eventual out-of-memory problem. This has been fixed. [GL #3051]

- If signatures created by the ZSK were expired and the ZSK private key was offline, the signatures were not replaced. This behavior has been amended to replace the expired signatures with new signatures created using the KSK. [GL #3049]

- Under certain circumstances, the signed version of an inline-signed zone could be dumped to disk without the serial number of the unsigned version of the zone. This prevented resynchronization of the zone contents after `named` restarted, if the unsigned zone file was modified while `named` was not running. This has been fixed. [GL #3071]

8.9 Notes for BIND 9.16.24

8.9.1 Feature Changes

- Previously, when an incoming TCP connection could not be accepted because the client closed the connection early, an error message of `TCP connection failed: socket is not connected` was logged. This message has been changed to `Accepting TCP connection failed: socket is not connected`. The severity level at which this type of message is logged has also been changed from `error` to `info` for the following triggering events: `socket is not connected`, `quota reached`, and `soft quota reached`. [GL #2700]

- `dnssec-dsfromkey` no longer generates DS records from revoked keys. [GL #853]

8.9.2 Bug Fixes

- Removing a configured `catalog-zone` clause from the configuration, running `rndc reconfig`, then bringing back the removed `catalog-zone` clause and running `rndc reconfig` again caused `named` to crash. This has been fixed. [GL #1608]

8.10 Notes for BIND 9.16.23

8.10.1 Bug Fixes

- Reloading a catalog zone which referenced a missing/deleted member zone triggered a runtime check failure, causing `named` to exit prematurely. This has been fixed. [GL #2308]

8.11 Notes for BIND 9.16.22

8.11.1 Security Fixes

- The `lame-ttl` option controls how long `named` caches certain types of broken responses from authoritative servers (see the security advisory for details). This caching mechanism could be abused by an attacker to significantly degrade resolver performance. The vulnerability has been mitigated by changing the default value of `lame-ttl` to `0` and overriding any explicitly set value with `0`, effectively disabling this mechanism altogether. ISC's testing has determined that doing that has a negligible impact on resolver performance while also preventing

abuse. Administrators may observe more traffic towards servers issuing certain types of broken responses than in previous BIND 9 releases, depending on client query patterns. (CVE-2021-25219)

ISC would like to thank Kishore Kumar Kothapalli of Infoblox for bringing this vulnerability to our attention. **[GL #2899]**

8.11.2 Feature Changes

- The use of native PKCS#11 for Public-Key Cryptography in BIND 9 has been deprecated in favor of the engine_pkcs11 OpenSSL engine from the OpenSC project. The `--with-native-pkcs11` configuration option will be removed in the next major BIND 9 release. The option to use the engine_pkcs11 OpenSSL engine is already available in BIND 9; please see the *ARM section on PKCS#11* for details. **[GL #2691]**

- Old-style Dynamically Loadable Zones (DLZ) drivers that had to be enabled in `named` at build time have been marked as deprecated in favor of new-style DLZ modules. Old-style DLZ drivers will be removed in the next major BIND 9 release. **[GL #2814]**

- The `map` zone file format has been marked as deprecated and will be removed in the next major BIND 9 release. **[GL #2882]**

- `named` and `named-checkconf` now exit with an error when a single port configured for `query-source`, `transfer-source`, `notify-source`, `parental-source`, and/or their respective IPv6 counterparts clashes with a global listening port. This configuration has not been supported since BIND 9.16.0, but no error was reported until now (even though sending UDP messages such as NOTIFY failed). **[GL #2888]**

- `named` and `named-checkconf` now issue a warning when there is a single port configured for `query-source`, `transfer-source`, `notify-source`, `parental-source`, and/or for their respective IPv6 counterparts. **[GL #2888]**

8.11.3 Bug Fixes

- A recent change introduced in BIND 9.16.21 inadvertently broke backward compatibility for the `check-names master ...` and `check-names slave ...` options, causing them to be silently ignored. This has been fixed and these options now work properly again. **[GL #2911]**

- When new IP addresses were set up by the operating system during `named` startup, it could fail to listen for TCP connections on the newly added interfaces. **[GL #2852]**

8.12 Notes for BIND 9.16.21

8.12.1 New Features

- Support for HTTPS and SVCB record types has been added. (This does not include ADDITIONAL section processing for these record types, only basic support for RR type parsing and printing.) **[GL #1132]**

8.12.2 Feature Changes

- When `dnssec-signzone` signs a zone using a successor key whose predecessor is still published, it now only refreshes signatures for RRsets which have an invalid signature, an expired signature, or a signature which expires within the provided cycle interval. This allows `dnssec-signzone` to gradually replace signatures in a zone whose ZSK is being rolled over (similarly to what `auto-dnssec maintain;` does). [GL #1551]

8.12.3 Bug Fixes

- A recent change to the internal memory structure of zone databases inadvertently neglected to update the MAPAPI value for zone files in `map` format. This caused version 9.16.20 of `named` to attempt to load files into memory that were no longer compatible, triggering an assertion failure on startup. The MAPAPI value has now been updated, so `named` rejects outdated files when encountering them. [GL #2872]

- Zone files in `map` format whose size exceeded 2 GB failed to load. This has been fixed. [GL #2878]

- `named` was unable to run as a Windows Service under certain circumstances. This has been fixed. [GL #2837]

- Stale data in the cache could cause `named` to send non-minimized queries despite QNAME minimization being enabled. This has been fixed. [GL #2665]

- When a DNSSEC-signed zone which only has a single signing key available is migrated to `dnssec-policy`, that key is now treated as a Combined Signing Key (CSK). [GL #2857]

- When a dynamic zone was made available in another view using the `in-view` statement, running `rndc freeze` always reported an `already frozen` error even though the zone was successfully frozen. This has been fixed. [GL #2844]

8.13 Notes for BIND 9.16.20

8.13.1 Security Fixes

- Fixed an assertion failure that occurred in `named` when it attempted to send a UDP packet that exceeded the MTU size, if Response Rate Limiting (RRL) was enabled. (CVE-2021-25218) [GL #2856]

- `named` failed to check the opcode of responses when performing zone refreshes, stub zone updates, and UPDATE forwarding. This could lead to an assertion failure under certain conditions and has been addressed by rejecting responses whose opcode does not match the expected value. [GL #2762]

8.13.2 Feature Changes

- Testing revealed that setting the thread affinity for various types of `named` threads led to inconsistent recursive performance, as sometimes multiple sets of threads competed over a single resource.

  Due to the above, `named` no longer sets thread affinity. This causes a slight dip of around 5% in authoritative performance, but recursive performance is now consistently improved. [GL #2822]

- CDS and CDNSKEY records can now be published in a zone without the requirement that they exactly match an existing DNSKEY record, as long as the zone is signed with an algorithm represented in the CDS or CDNSKEY record. This allows a clean rollover from one DNS provider to another when using a multiple-signer DNSSEC configuration. [GL #2710]

8.13.3 Bug Fixes

- Authentication of `rndc` messages could fail if a `controls` statement was configured with multiple key algorithms for the same listener. This has been fixed. **[GL #2756]**

8.14 Notes for BIND 9.16.19

8.14.1 New Features

- Using a new configuration option, `parental-agents`, each zone can now be associated with a list of servers that can be used to check the DS RRset in the parent zone. This enables automatic KSK rollovers. **[GL #1126]**

8.14.2 Feature Changes

- IP fragmentation has been disabled for outgoing UDP sockets. Errors triggered by sending DNS messages larger than the specified path MTU are properly handled by sending empty DNS replies with the `TC` (TrunCated) bit set, which forces DNS clients to fall back to TCP. **[GL #2790]**

8.14.3 Bug Fixes

- The code managing **RFC 5011** trust anchors created an invalid placeholder keydata record upon a refresh failure, which prevented the database of managed keys from subsequently being read back. This has been fixed. **[GL #2686]**

- Signed, insecure delegation responses prepared by `named` either lacked the necessary NSEC records or contained duplicate NSEC records when both wildcard expansion and CNAME chaining were required to prepare the response. This has been fixed. **[GL #2759]**

- If `nsupdate` sends an SOA request and receives a REFUSED response, it now fails over to the next available server. **[GL #2758]**

- A bug that caused the NSEC3 salt to be changed on every restart for zones using KASP has been fixed. **[GL #2725]**

- The configuration-checking code failed to account for the inheritance rules of the `dnssec-policy` option. This has been fixed. **[GL #2780]**

- The fix for **[GL #1875]** inadvertently introduced a deadlock: when locking key files for reading and writing, the `in-view` logic was not considered. This has been fixed. **[GL #2783]**

- A race condition could occur where two threads were competing for the same set of key file locks, leading to a deadlock. This has been fixed. **[GL #2786]**

8.15 Notes for BIND 9.16.18

8.15.1 Bug Fixes

- When preparing DNS responses, `named` could replace the letters `W` (uppercase) and `w` (lowercase) with `\000`. This has been fixed. **[GL #2779]**

- The configuration-checking code failed to account for the inheritance rules of the `key-directory` option. As a side effect of this flaw, the code detecting `key-directory` conflicts for zones using KASP incorrectly reported unique key directories as being reused. This has been fixed. [GL #2778]

8.16 Notes for BIND 9.16.17

8.16.1 Feature Changes

- After the network manager was introduced to `named` to handle incoming traffic, it was discovered that recursive performance had degraded compared to previous BIND 9 versions. This has now been fixed by processing internal tasks inside network manager worker threads, preventing resource contention among two sets of threads. [GL #2638]

- Zone dumping tasks are now run on separate asynchronous thread pools. This change prevents zone dumping from blocking network I/O. [GL #2732]

- `inline-signing` was incorrectly described as being inherited from the `options`/`view` levels and was incorrectly accepted at those levels without effect. This has been fixed; `named.conf` files with `inline-signing` at those levels no longer load. [GL #2536]

8.16.2 Bug Fixes

- The calculation of the estimated IXFR transaction size in `dns_journal_iter_init()` was invalid. This resulted in excessive AXFR-style IXFR responses. [GL #2685]

- Fixed an assertion failure that could occur if stale data was used to answer a query, and then a prefetch was triggered after the query was restarted (for example, to follow a CNAME). [GL #2733]

- If a query was answered with stale data on a server with DNS64 enabled, an assertion could occur if a non-stale answer arrived afterward. This has been fixed. [GL #2731]

- Fixed an error which caused the `IP_DONTFRAG` socket option to be enabled instead of disabled, leading to errors when sending oversized UDP packets. [GL #2746]

- Zones which are configured in multiple views, with different values set for `dnssec-policy` and with identical values set for `key-directory`, are now detected and treated as a configuration error. [GL #2463]

- A race condition could occur when reading and writing key files for zones using KASP and configured in multiple views. This has been fixed. [GL #1875]

8.17 Notes for BIND 9.16.16

8.17.1 Feature Changes

- DNSSEC responses containing NSEC3 records with iteration counts greater than 150 are now treated as insecure. [GL #2445]

- The maximum supported number of NSEC3 iterations that can be configured for a zone has been reduced to 150. [GL #2642]

- The default value of the `max-ixfr-ratio` option was changed to `unlimited`, for better backwards compatibility in the stable release series. [GL #2671]

- Zones that want to transition from secure to insecure mode without becoming bogus in the process must now have their `dnssec-policy` changed first to `insecure`, rather than `none`. After the DNSSEC records have been removed from the zone, the `dnssec-policy` can be set to `none` or removed from the configuration. Setting the `dnssec-policy` to `insecure` causes CDS and CDNSKEY DELETE records to be published. **[GL #2645]**

- The implementation of the ZONEMD RR type has been updated to match **RFC 8976**. **[GL #2658]**

- The `draft-vandijk-dnsop-nsec-ttl` IETF draft was implemented: NSEC(3) TTL values are now set to the minimum of the SOA MINIMUM value or the SOA TTL. **[GL #2347]**

8.17.2 Bug Fixes

- It was possible for corrupt journal files generated by an earlier version of `named` to cause problems after an upgrade. This has been fixed. **[GL #2670]**

- TTL values in cache dumps were reported incorrectly when `stale-cache-enable` was set to `yes`. This has been fixed. **[GL #389] [GL #2289]**

- A deadlock could occur when multiple `rndc addzone`, `rndc delzone`, and/or `rndc modzone` commands were invoked simultaneously for different zones. This has been fixed. **[GL #2626]**

- `named` and `named-checkconf` did not report an error when multiple zones with the `dnssec-policy` option set were using the same zone file. This has been fixed. **[GL #2603]**

- If `dnssec-policy` was active and a private key file was temporarily offline during a rekey event, `named` could incorrectly introduce replacement keys and break a signed zone. This has been fixed. **[GL #2596]**

- When generating zone signing keys, KASP now also checks for key ID conflicts among newly created keys, rather than just between new and existing ones. **[GL #2628]**

8.18 Notes for BIND 9.16.15

8.18.1 Security Fixes

- A malformed incoming IXFR transfer could trigger an assertion failure in `named`, causing it to quit abnormally. (CVE-2021-25214)

  ISC would like to thank Greg Kuechle of SaskTel for bringing this vulnerability to our attention. **[GL #2467]**

- `named` crashed when a DNAME record placed in the ANSWER section during DNAME chasing turned out to be the final answer to a client query. (CVE-2021-25215)

  ISC would like to thank Siva Kakarla for bringing this vulnerability to our attention. **[GL #2540]**

- When a server's configuration set the `tkey-gssapi-keytab` or `tkey-gssapi-credential` option, a specially crafted GSS-TSIG query could cause a buffer overflow in the ISC implementation of SPNEGO (a protocol enabling negotiation of the security mechanism used for GSSAPI authentication). This flaw could be exploited to crash `named` binaries compiled for 64-bit platforms, and could enable remote code execution when `named` was compiled for 32-bit platforms. (CVE-2021-25216)

  This vulnerability was reported to us as ZDI-CAN-13347 by Trend Micro Zero Day Initiative. **[GL #2604]**

8.18.2 Feature Changes

- The ISC implementation of SPNEGO was removed from BIND 9 source code. Instead, BIND 9 now always uses the SPNEGO implementation provided by the system GSSAPI library when it is built with GSSAPI support. All major contemporary Kerberos/GSSAPI libraries contain an implementation of the SPNEGO mechanism. **[GL #2607]**

- The default value for the `stale-answer-client-timeout` option was changed from `1800` (ms) to `off`. The default value may be changed again in future releases as this feature matures. **[GL #2608]**

8.18.3 Bug Fixes

- TCP idle and initial timeouts were being incorrectly applied: only the `tcp-initial-timeout` was applied on the whole connection, even if the connection were still active, which could prevent a large zone transfer from being sent back to the client. The default setting for `tcp-initial-timeout` was 30 seconds, which meant that any TCP connection taking more than 30 seconds was abruptly terminated. This has been fixed. **[GL #2583]**

- When `stale-answer-client-timeout` was set to a positive value and recursion for a client query completed when `named` was about to look for a stale answer, an assertion could fail in `query_respond()`, resulting in a crash. This has been fixed. **[GL #2594]**

- If zone journal files written by BIND 9.16.11 or earlier were present when BIND was upgraded to BIND 9.16.13 or BIND 9.16.14, the zone file for that zone could have been inadvertently rewritten with the current zone contents. This caused the original zone file structure (e.g. comments, `$INCLUDE` directives) to be lost, although the zone data itself was preserved. **[GL #2623]**

- After upgrading to BIND 9.16.13, journal files for trust anchor databases (e.g. `managed-keys.bind.jnl`) could be left in a corrupt state. (Other zone journal files were not affected.) This has been fixed. If a corrupt journal file is detected, `named` can now recover from it. **[GL #2600]**

- When sending queries over TCP, `dig` now properly handles `+tries=1 +retry=0` by not retrying the connection when the remote server closes the connection prematurely. **[GL #2490]**

- CDS/CDNSKEY DELETE records are now removed when a zone transitions from a secure to an insecure state. `named-checkzone` also no longer reports an error when such records are found in an unsigned zone. **[GL #2517]**

- Zones using KASP could not be thawed after they were frozen using `rndc freeze`. This has been fixed. **[GL #2523]**

- After `rndc checkds -checkds` or `rndc dnssec -rollover` is used, `named` now immediately attempts to reconfigure zone keys. This change prevents unnecessary key rollover delays. **[GL #2488]**

- Previously, a memory leak could occur when `named` failed to bind a UDP socket to a network interface. This has been fixed. **[GL #2575]**

8.19 Notes for BIND 9.16.14

Note: The BIND 9.16.14 release was withdrawn after a backporting bug was discovered during pre-release testing. ISC would like to acknowledge the assistance of Natan Segal of Bluecat Networks.

8.20 Notes for BIND 9.16.13

8.20.1 New Features

- A new `purge-keys` option has been added to `dnssec-policy`. It sets the period of time that key files are retained after becoming obsolete due to a key rollover; the default is 90 days. This feature can be disabled by setting `purge-keys` to 0. **[GL #2408]**

8.20.2 Feature Changes

- When serve-stale is enabled and stale data is available, `named` now returns stale answers upon encountering any unexpected error in the query resolution process. This may happen, for example, if the `fetches-per-server` or `fetches-per-zone` limits are reached. In this case, `named` attempts to answer DNS requests with stale data, but does not start the `stale-refresh-time` window. **[GL #2434]**

8.20.3 Bug Fixes

- Zone journal (`.jnl`) files created by versions of `named` prior to 9.16.12 were no longer compatible; this could cause problems when upgrading if journal files were not synchronized first. This has been corrected: older journal files can now be read when starting up. When an old-style journal file is detected, it is updated to the new format immediately after loading.

  Note that journals created by the current version of `named` are not usable by versions prior to 9.16.12. Before downgrading to a prior release, users are advised to ensure that all dynamic zones have been synchronized using `rndc sync -clean`.

  A journal file's format can be changed manually by running `named-journalprint -d` (downgrade) or `named-journalprint -u` (upgrade). Note that this *must not* be done while `named` is running. **[GL #2505]**

- `named` crashed when it was allowed to serve stale answers and `stale-answer-client-timeout` was triggered without any (stale) data available in the cache to answer the query. **[GL #2503]**

- If an outgoing packet exceeded `max-udp-size`, `named` dropped it instead of sending back a proper response. To prevent this problem, the `IP_DONTFRAG` option is no longer set on UDP sockets, which has been happening since BIND 9.16.11. **[GL #2466]**

- NSEC3 records were not immediately created when signing a dynamic zone using `dnssec-policy` with `nsec3param`. This has been fixed. **[GL #2498]**

- A memory leak occurred when `named` was reconfigured after adding an inline-signed zone with `auto-dnssec maintain` enabled. This has been fixed. **[GL #2041]**

- An invalid direction field (not one of `N`, `S`, `E`, `W`) in a LOC record resulted in an INSIST failure when a zone file containing such a record was loaded. **[GL #2499]**

8.21 Notes for BIND 9.16.12

8.21.1 Security Fixes

- When `tkey-gssapi-keytab` or `tkey-gssapi-credential` was configured, a specially crafted GSS-TSIG query could cause a buffer overflow in the ISC implementation of SPNEGO (a protocol enabling negotiation of the security mechanism to use for GSSAPI authentication). This flaw could be exploited to crash `named`. Theoretically, it also enabled remote code execution, but achieving the latter is very difficult in real-world conditions. (CVE-2020-8625)

  This vulnerability was responsibly reported to us as ZDI-CAN-12302 by Trend Micro Zero Day Initiative. **[GL #2354]**

8.21.2 New Features

- When a secondary server receives a large incremental zone transfer (IXFR), it can have a negative impact on query performance while the incremental changes are applied to the zone. To address this, `named` can now limit the size of IXFR responses it sends in response to zone transfer requests. If an IXFR response would be larger than an AXFR of the entire zone, it will send an AXFR response instead.

  This behavior is controlled by the `max-ixfr-ratio` option - a percentage value representing the ratio of IXFR size to the size of a full zone transfer. The default is `100%`. **[GL #1515]**

- A new option, `stale-answer-client-timeout`, has been added to improve `named`'s behavior with respect to serving stale data. The option defines the amount of time `named` waits before attempting to answer the query with a stale RRset from cache. If a stale answer is found, `named` continues the ongoing fetches, attempting to refresh the RRset in cache until the `resolver-query-timeout` interval is reached.

  The default value is `1800` (in milliseconds) and the maximum value is limited to `resolver-query-timeout` minus one second. A value of `0` causes any available cached RRset to immediately be returned while still triggering a refresh of the data in cache.

  This new behavior can be disabled by setting `stale-answer-client-timeout` to `off` or `disabled`. The new option has no effect if `stale-answer-enable` is disabled. **[GL #2247]**

8.21.3 Feature Changes

- As part of an ongoing effort to use **RFC 8499** terminology, `primaries` can now be used as a synonym for `masters` in `named.conf`. Similarly, `notify primary-only` can now be used as a synonym for `notify master-only`. The output of `rndc zonestatus` now uses `primary` and `secondary` terminology. **[GL #1948]**

- The default value of `max-stale-ttl` has been changed from 12 hours to 1 day and the default value of `stale-answer-ttl` has been changed from 1 second to 30 seconds, following **RFC 8767** recommendations. **[GL #2248]**

- The SONAMEs for BIND 9 libraries now include the current BIND 9 version number, in an effort to tightly couple internal libraries with a specific release. This change makes the BIND 9 release process both simpler and more consistent while also unequivocally preventing BIND 9 binaries from silently loading wrong versions of shared libraries (or multiple versions of the same shared library) at startup. **[GL #2387]**

- When `check-names` is in effect, A records below an `_spf`, `_spf_rate`, or `_spf_verify` label (which are employed by the `exists` SPF mechanism defined in **RFC 7208** section 5.7/appendix D.1) are no longer reported as warnings/errors. **[GL #2377]**

8.21.4 Bug Fixes

- `named` failed to start when its configuration included a zone with a non-builtin `allow-update` ACL attached. **[GL #2413]**

- Previously, `dnssec-keyfromlabel` crashed when operating on an ECDSA key. This has been fixed. **[GL #2178]**

- KASP incorrectly set signature validity to the value of the DNSKEY signature validity. This has been fixed. **[GL #2383]**

- When migrating to KASP, BIND 9 considered keys with the `Inactive` and/or `Delete` timing metadata to be possible active keys. This has been fixed. **[GL #2406]**

- Fix the "three is a crowd" key rollover bug in KASP. When keys rolled faster than the time required to finish the rollover procedure, the successor relation equation failed because it assumed only two keys were taking part in a rollover. This could lead to premature removal of predecessor keys. BIND 9 now implements a recursive successor relation, as described in the paper "Flexible and Robust Key Rollover" (Equation (2)). **[GL #2375]**

- Performance of the DNSSEC verification code (used by `dnssec-signzone`, `dnssec-verify`, and mirror zones) has been improved. **[GL #2073]**

8.22 Notes for BIND 9.16.11

8.22.1 Feature Changes

- The new networking code introduced in BIND 9.16 (netmgr) was overhauled in order to make it more stable, testable, and maintainable. **[GL #2321]**

- Earlier releases of BIND versions 9.16 and newer required the operating system to support load-balanced sockets in order for `named` to be able to achieve high performance (by distributing incoming queries among multiple threads). However, the only operating systems currently known to support load-balanced sockets are Linux and FreeBSD 12, which means both UDP and TCP performance were limited to a single thread on other systems. As of BIND 9.16.11, `named` attempts to distribute incoming queries among multiple threads on systems which lack support for load-balanced sockets (except Windows). **[GL #2137]**

- It is now possible to transition a zone from secure to insecure mode without making it bogus in the process; changing to `dnssec-policy none;` also causes CDS and CDNSKEY DELETE records to be published, to signal that the entire DS RRset at the parent must be removed, as described in **RFC 8078**. **[GL #1750]**

- When using the `unixtime` or `date` method to update the SOA serial number, `named` and `dnssec-signzone` silently fell back to the `increment` method to prevent the new serial number from being smaller than the old serial number (using serial number arithmetics). `dnssec-signzone` now prints a warning message, and `named` logs a warning, when such a fallback happens. **[GL #2058]**

8.22.2 Bug Fixes

- Multiple threads could attempt to destroy a single RBTDB instance at the same time, resulting in an unpredictable but low-probability assertion failure in `free_rbtdb()`. This has been fixed. **[GL #2317]**

- `named` no longer attempts to assign threads to CPUs outside the CPU affinity set. Thanks to Ole Bjørn Hessen. **[GL #2245]**

- When reconfiguring `named`, removing `auto-dnssec` did not turn off DNSSEC maintenance. This has been fixed. **[GL #2341]**

- The report of intermittent BIND assertion failures triggered in `lib/dns/resolver.c:dns_name_issubdomain()` has now been closed without further action. Our initial response to this was to add diagnostic logging instead of terminating `named`, anticipating that we would receive further useful troubleshooting input. This workaround first appeared in BIND releases 9.17.5 and 9.16.7. However, since those releases were published, there have been no new reports of assertion failures matching this issue, but also no further diagnostic input, so we have closed the issue. [GL #2091]

8.23 Notes for BIND 9.16.10

8.23.1 New Features

- NSEC3 support was added to KASP. A new option for `dnssec-policy`, `nsec3param`, can be used to set the desired NSEC3 parameters. NSEC3 salt collisions are automatically prevented during resalting. [GL #1620]

8.23.2 Feature Changes

- The default value of `max-recursion-queries` was increased from 75 to 100. Since the queries sent towards root and TLD servers are now included in the count (as a result of the fix for CVE-2020-8616), `max-recursion-queries` has a higher chance of being exceeded by non-attack queries, which is the main reason for increasing its default value. [GL #2305]

- The default value of `nocookie-udp-size` was restored back to 4096 bytes. Since `max-udp-size` is the upper bound for `nocookie-udp-size`, this change relieves the operator from having to change `nocookie-udp-size` together with `max-udp-size` in order to increase the default EDNS buffer size limit. `nocookie-udp-size` can still be set to a value lower than `max-udp-size`, if desired. [GL #2250]

8.23.3 Bug Fixes

- Handling of missing DNS COOKIE responses over UDP was tightened by falling back to TCP. [GL #2275]

- The CNAME synthesized from a DNAME was incorrectly followed when the QTYPE was CNAME or ANY. [GL #2280]

- Building with native PKCS#11 support for AEP Keyper has been broken since BIND 9.16.6. This has been fixed. [GL #2315]

8.24 Notes for BIND 9.16.9

8.24.1 New Features

- A new configuration option, `stale-refresh-time`, has been introduced. It allows a stale RRset to be served directly from cache for a period of time after a failed lookup, before a new attempt to refresh it is made. [GL #2066]

8.24.2 Bug Fixes

- `named` could crash with an assertion failure if a TCP connection were closed while a request was still being processed. **[GL #2227]**

- `named` acting as a resolver could incorrectly treat signed zones with no DS record at the parent as bogus. Such zones should be treated as insecure. This has been fixed. **[GL #2236]**

- After a Negative Trust Anchor (NTA) is added, BIND performs periodic checks to see if it is still necessary. If BIND encountered a failure while creating a query to perform such a check, it attempted to dereference a `NULL` pointer, resulting in a crash. **[GL #2244]**

- A problem obtaining glue records could prevent a stub zone from functioning properly, if the authoritative server for the zone were configured for minimal responses. **[GL #1736]**

- `UV_EOF` is no longer treated as a `TCP4RecvErr` or a `TCP6RecvErr`. **[GL #2208]**

8.25 Notes for BIND 9.16.8

8.25.1 New Features

- Add a new `rndc` command, `rndc dnssec -rollover`, which triggers a manual rollover for a specific key. **[GL #1749]**

- Add a new `rndc` command, `rndc dumpdb -expired`, which dumps the cache database, including expired RRsets that are awaiting cleanup, to the `dump-file` for diagnostic purposes. **[GL #1870]**

8.25.2 Feature Changes

- DNS Flag Day 2020: The default EDNS buffer size has been changed from 4096 to 1232 bytes. According to measurements done by multiple parties, this should not cause any operational problems as most of the Internet "core" is able to cope with IP message sizes between 1400-1500 bytes; the 1232 size was picked as a conservative minimal number that could be changed by the DNS operator to an estimated path MTU minus the estimated header space. In practice, the smallest MTU witnessed in the operational DNS community is 1500 octets, the maximum Ethernet payload size, so a useful default for maximum DNS/UDP payload size on reliable networks would be 1432 bytes. **[GL #2183]**

8.25.3 Bug Fixes

- `named` reported an invalid memory size when running in an environment that did not properly report the number of available memory pages and/or the size of each memory page. **[GL #2166]**

- With multiple forwarders configured, `named` could fail the `REQUIRE(msg->state == (-1))` assertion in `lib/dns/message.c`, causing it to crash. This has been fixed. **[GL #2124]**

- `named` erroneously performed continuous key rollovers for KASP policies that used algorithm Ed25519 or Ed448 due to a mismatch between created key size and expected key size. **[GL #2171]**

- Updating contents of an RPZ zone which contained names spelled using varying letter case could cause some processing rules in that RPZ zone to be erroneously ignored. **[GL #2169]**

8.26 Notes for BIND 9.16.7

8.26.1 New Features

- Add a new `rndc` command, `rndc dnssec -checkds`, which signals to `named` that a DS record for a given zone or key has been published or withdrawn from the parent. This command replaces the time-based `parent-registration-delay` configuration option. [GL #1613]

- Log when `named` adds a CDS/CDNSKEY to the zone. [GL #1748]

8.26.2 Bug Fixes

- In rare circumstances, `named` would exit with an assertion failure when the number of nodes stored in the red-black tree exceeded the maximum allowed size of the internal hash table. [GL #2104]

- Silence spurious system log messages for an EPROTO(71) error code that was seen on older operating systems, where unhandled ICMPv6 errors resulted in a generic protocol error being returned instead of a more specific error code. [GL #1928]

- With query name minimization enabled, `named` failed to resolve `ip6.arpa.` names that had extra labels to the left of the IPv6 part. For example, when `named` attempted query name minimization on a name like `A.B.1.2.3.4.(...).ip6.arpa.`, it stopped at the leftmost IPv6 label, i.e. `1.2.3.4.(...).ip6.arpa.`, without considering the extra labels (`A.B`). That caused a query loop when resolving the name: if `named` received NXDOMAIN answers, then the same query was repeatedly sent until the number of queries sent reached the value of the `max-recursion-queries` configuration option. [GL #1847]

- Parsing of LOC records was made more strict by rejecting a sole period (`.`) and/or `m` as a value. These changes prevent zone files using such values from being loaded. Handling of negative altitudes which are not integers was also corrected. [GL #2074]

- Several problems found by OSS-Fuzz were fixed. (None of these are security issues.) [GL !3953] [GL !3975]

8.27 Notes for BIND 9.16.6

8.27.1 Security Fixes

- It was possible to trigger an assertion failure by sending a specially crafted large TCP DNS message. This was disclosed in CVE-2020-8620.

  ISC would like to thank Emanuel Almeida of Cisco Systems, Inc. for bringing this vulnerability to our attention. [GL #1996]

- `named` could crash after failing an assertion check in certain query resolution scenarios where QNAME minimization and forwarding were both enabled. To prevent such crashes, QNAME minimization is now always disabled for a given query resolution process, if forwarders are used at any point. This was disclosed in CVE-2020-8621.

  ISC would like to thank Joseph Gullo for bringing this vulnerability to our attention. [GL #1997]

- It was possible to trigger an assertion failure when verifying the response to a TSIG-signed request. This was disclosed in CVE-2020-8622.

  ISC would like to thank Dave Feldman, Jeff Warren, and Joel Cunningham of Oracle for bringing this vulnerability to our attention. [GL #2028]

- When BIND 9 was compiled with native PKCS#11 support, it was possible to trigger an assertion failure in code determining the number of bits in the PKCS#11 RSA public key with a specially crafted packet. This was disclosed in CVE-2020-8623.

  ISC would like to thank Lyu Chiy for bringing this vulnerability to our attention. [GL #2037]

- `update-policy` rules of type `subdomain` were incorrectly treated as `zonesub` rules, which allowed keys used in `subdomain` rules to update names outside of the specified subdomains. The problem was fixed by making sure `subdomain` rules are again processed as described in the ARM. This was disclosed in CVE-2020-8624.

  ISC would like to thank Joop Boonen of credativ GmbH for bringing this vulnerability to our attention. [GL #2055]

8.27.2 New Features

- A new configuration option `stale-cache-enable` has been introduced to enable or disable keeping stale answers in cache. [GL #1712]

8.27.3 Feature Changes

- BIND's cache database implementation has been updated to use a faster hash function with better distribution. In addition, the effective `max-cache-size` (configured explicitly, defaulting to a value based on system memory or set to `unlimited`) now pre-allocates fixed-size hash tables. This prevents interruption to query resolution when the hash table sizes need to be increased. [GL #1775]

- Resource records received with 0 TTL are no longer kept in the cache to be used for stale answers. [GL #1829]

8.27.4 Bug Fixes

- Wildcard RPZ passthru rules could incorrectly be overridden by other rules that were loaded from RPZ zones which appeared later in the `response-policy` statement. This has been fixed. [GL #1619]

- The IPv6 Duplicate Address Detection (DAD) mechanism could inadvertently prevent `named` from binding to new IPv6 interfaces, by causing multiple route socket messages to be sent for each IPv6 address. `named` monitors for new interfaces to `bind()` to when it is configured to listen on `any` or on a specific range of addresses. New IPv6 interfaces can be in a "tentative" state before they are fully available for use. When DAD is in use, two messages are emitted by the route socket: one when the interface first appears and then a second one when it is fully "up." An attempt by `named` to `bind()` to the new interface prematurely would fail, causing it thereafter to ignore that address/interface. The problem was worked around by setting the `IP_FREEBIND` option on the socket and trying to `bind()` to each IPv6 address again if the first `bind()` call for that address failed with `EADDRNOTAVAIL`. [GL #2038]

- Addressed an error in recursive clients stats reporting which could cause underflow, and even negative statistics. There were occasions when an incoming query could trigger a prefetch for some eligible RRset, and if the prefetch code were executed before recursion, no increment in recursive clients stats would take place. Conversely, when processing the answers, if the recursion code were executed before the prefetch, the same counter would be decremented without a matching increment. [GL #1719]

- The introduction of KASP support inadvertently caused the second field of `sig-validity-interval` to always be calculated in hours, even in cases when it should have been calculated in days. This has been fixed. (Thanks to Tony Finch.) [GL !3735]

- LMDB locking code was revised to make `rndc reconfig` work properly on FreeBSD and with LMDB >= 0.9.26. [GL #1976]

8.28 Notes for BIND 9.16.5

8.28.1 New Features

- New `rndc` command `rndc dnssec -status` shows the current DNSSEC policy and keys in use, the key states, and rollover status. **[GL #1612]**

8.28.2 Bug Fixes

- A race condition could occur if a TCP socket connection was closed while `named` was waiting for a recursive response. The attempt to send a response over the closing connection triggered an assertion failure in the function `isc__nm_tcpdns_send()`. **[GL #1937]**

- A race condition could occur when `named` attempted to use a UDP interface that was shutting down. This triggered an assertion failure in `uv__udp_finish_close()`. **[GL #1938]**

- Fix assertion failure when server was under load and root zone had not yet been loaded. **[GL #1862]**

- `named` could crash when cleaning dead nodes in `lib/dns/rbtdb.c` that were being reused. **[GL #1968]**

- `named` crashed on shutdown when a new `rndc` connection was received during shutdown. This has been fixed. **[GL #1747]**

- The DS RRset returned by `dns_keynode_dsset()` was used in a non-thread-safe manner. This could result in an INSIST being triggered. **[GL #1926]**

- Properly handle missing `kyua` command so that `make check` does not fail unexpectedly when CMocka is installed, but Kyua is not. **[GL #1950]**

- The `primary` and `secondary` keywords, when used as parameters for `check-names`, were not processed correctly and were being ignored. **[GL #1949]**

- `rndc dnstap -roll <value>` did not limit the number of saved files to `<value>`. **[GL !3728]**

- The validator could fail to accept a properly signed RRset if an unsupported algorithm appeared earlier in the DNSKEY RRset than a supported algorithm. It could also stop if it detected a malformed public key. **[GL #1689]**

- The `blackhole` ACL was inadvertently disabled for client queries. Blocked IP addresses were not used for upstream queries but queries from those addresses could still be answered. **[GL #1936]**

8.29 Notes for BIND 9.16.4

8.29.1 Security Fixes

- It was possible to trigger an assertion when attempting to fill an oversized TCP buffer. This was disclosed in CVE-2020-8618. **[GL #1850]**

- It was possible to trigger an INSIST failure when a zone with an interior wildcard label was queried in a certain pattern. This was disclosed in CVE-2020-8619. **[GL #1111] [GL #1718]**

8.29.2 New Features

- Documentation was converted from DocBook to reStructuredText. The BIND 9 ARM is now generated using Sphinx and published on Read the Docs. Release notes are no longer available as a separate document accompanying a release. [GL #83]

- `named` and `named-checkzone` now reject master zones that have a DS RRset at the zone apex. Attempts to add DS records at the zone apex via UPDATE will be logged but otherwise ignored. DS records belong in the parent zone, not at the zone apex. [GL #1798]

- `dig` and other tools can now print the Extended DNS Error (EDE) option when it appears in a request or a response. [GL #1835]

8.29.3 Feature Changes

- The default value of `max-stale-ttl` has changed from 1 week to 12 hours. This option controls how long `named` retains expired RRsets in cache as a potential mitigation mechanism, should there be a problem with one or more domains. Note that cache content retention is independent of whether stale answers are used in response to client queries (`stale-answer-enable yes|no` and `rndc serve-stale on|off`). Serving of stale answers when the authoritative servers are not responding must be explicitly enabled, whereas the retention of expired cache content takes place automatically on all versions of BIND 9 that have this feature available. [GL #1877]

> **Warning:** This change may be significant for administrators who expect that stale cache content will be automatically retained for up to 1 week. Add option `max-stale-ttl 1w;` to `named.conf` to keep the previous behavior of `named`.

- `listen-on-v6 { any; }` creates a separate socket for each interface. Previously, just one socket was created on systems conforming to RFC 3493 and RFC 3542. This change was introduced in BIND 9.16.0, but it was accidentally omitted from documentation. [GL #1782]

8.29.4 Bug Fixes

- When fully updating the NSEC3 chain for a large zone via IXFR, a temporary loss of performance could be experienced on the secondary server when answering queries for nonexistent data that required DNSSEC proof of non-existence (in other words, queries that required the server to find and to return NSEC3 data). The unnecessary processing step that was causing this delay has now been removed. [GL #1834]

- `named` could crash with an assertion failure if the name of a database node was looked up while the database was being modified. [GL #1857]

- A possible deadlock in `lib/isc/unix/socket.c` was fixed. [GL #1859]

- Previously, `named` did not destroy some mutexes and conditional variables in netmgr code, which caused a memory leak on FreeBSD. This has been fixed. [GL #1893]

- A data race in `lib/dns/resolver.c:log_formerr()` that could lead to an assertion failure was fixed. [GL #1808]

- Previously, `provide-ixfr no;` failed to return up-to-date responses when the serial number was greater than or equal to the current serial number. [GL #1714]

- A bug in dnssec-policy keymgr was fixed, where the check for the existence of a given key's successor would incorrectly return `true` if any other key in the keyring had a successor. [GL #1845]

- With dnssec-policy, when creating a successor key, the "goal" state of the current active key (the predecessor) was not changed and thus never removed from the zone. [GL #1846]

- `named-checkconf -p` could include spurious text in `server-addresses` statements due to an uninitialized DSCP value. This has been fixed. [GL #1812]

- The ARM has been updated to indicate that the TSIG session key is generated when named starts, regardless of whether it is needed. [GL #1842]

8.30 Notes for BIND 9.16.3

8.30.1 Security Fixes

- To prevent exhaustion of server resources by a maliciously configured domain, the number of recursive queries that can be triggered by a request before aborting recursion has been further limited. Root and top-level domain servers are no longer exempt from the `max-recursion-queries` limit. Fetches for missing name server address records are limited to 4 for any domain. This issue was disclosed in CVE-2020-8616. [GL #1388]

- Replaying a TSIG BADTIME response as a request could trigger an assertion failure. This was disclosed in CVE-2020-8617. [GL #1703]

8.30.2 Known Issues

- BIND crashes on startup when linked against libuv 1.36. This issue is related to `recvmmsg()` support in libuv, which was first included in libuv 1.35. The problem was addressed in libuv 1.37, but the relevant libuv code change requires a special flag to be set during library initialization in order for `recvmmsg()` support to be enabled. This BIND release sets that special flag when required, so `recvmmsg()` support is now enabled when BIND is compiled against either libuv 1.35 or libuv 1.37+; libuv 1.36 is still not usable with BIND. [GL #1761] [GL #1797]

8.30.3 Feature Changes

- BIND 9 no longer sets receive/send buffer sizes for UDP sockets, relying on system defaults instead. [GL #1713]

- The default rwlock implementation has been changed back to the native BIND 9 rwlock implementation. [GL #1753]

- The native PKCS#11 EdDSA implementation has been updated to PKCS#11 v3.0 and thus made operational again. Contributed by Aaron Thompson. [GL !3326]

- The OpenSSL ECDSA implementation has been updated to support PKCS#11 via OpenSSL engine (see engine_pkcs11 from libp11 project). [GL #1534]

- The OpenSSL EdDSA implementation has been updated to support PKCS#11 via OpenSSL engine. Please note that an EdDSA-capable OpenSSL engine is required and thus this code is only a proof-of-concept for the time being. Contributed by Aaron Thompson. [GL #1763]

- Message IDs in inbound AXFR transfers are now checked for consistency. Log messages are emitted for streams with inconsistent message IDs. [GL #1674]

- The zone timers are now exported to the statistics channel. For the primary zones, only the loaded time is exported. For the secondary zones, the exported timers also include expire and refresh times. Contributed by Paul Frieden, Verizon Media. [GL #1232]

8.30.4 Bug Fixes

- A bug in dnstap initialization could prevent some dnstap data from being logged, especially on recursive resolvers. **[GL #1795]**

- When running on a system with support for Linux capabilities, `named` drops root privileges very soon after system startup. This was causing a spurious log message, `unable to set effective uid to 0: Operation not permitted`, which has now been silenced. **[GL #1042] [GL #1090]**

- When `named-checkconf -z` was run, it would sometimes incorrectly set its exit code. It reflected only the status of the last view found; any errors found for other configured views were not reported. Thanks to Graham Clinch. **[GL #1807]**

- When built without LMDB support, `named` failed to restart after a zone with a double quote (") in its name was added with `rndc addzone`. Thanks to Alberto Fernández. **[GL #1695]**

8.31 Notes for BIND 9.16.2

8.31.1 Security Fixes

- DNS rebinding protection was ineffective when BIND 9 is configured as a forwarding DNS server. Found and responsibly reported by Tobias Klein.:gl:*#1574*

8.31.2 Known Issues

- We have received reports that in some circumstances, receipt of an IXFR can cause the processing of queries to slow significantly. Some of these were related to RPZ processing, which has been fixed in this release (see below). Others appear to occur where there are NSEC3-related changes (such as an operator changing the NSEC3 salt used in the hash calculation). These are being investigated. **[GL #1685]**

8.31.3 Feature Changes

- The previous DNSSEC sign statistics used lots of memory. The number of keys to track is reduced to four per zone, which should be enough for 99% of all signed zones. **[GL #1179]**

8.31.4 Bug Fixes

- When an RPZ policy zone was updated via zone transfer and a large number of records was deleted, `named` could become nonresponsive for a short period while deleted names were removed from the RPZ summary database. This database cleanup is now done incrementally over a longer period of time, reducing such delays. **[GL #1447]**

- When trying to migrate an already-signed zone from `auto-dnssec maintain` to one based on `dnssec-policy`, the existing keys were immediately deleted and replaced with new ones. As the key rollover timing constraints were not being followed, it was possible that some clients would not have been able to validate responses until all old DNSSEC information had timed out from caches. BIND now looks at the time metadata of the existing keys and incorporates it into its DNSSEC policy operation. **[GL #1706]**

8.32 Notes for BIND 9.16.1

8.32.1 Known Issues

- UDP network ports used for listening can no longer simultaneously be used for sending traffic. An example configuration which triggers this issue would be one which uses the same address:port pair for `listen-on(-v6)` statements as for `notify-source(-v6)` or `transfer-source(-v6)`. While this issue affects all operating systems, it only triggers log messages (e.g. "unable to create dispatch for reserved port") on some of them. There are currently no plans to make such a combination of settings work again.

8.32.2 Feature Changes

- The system-provided POSIX Threads read-write lock implementation is now used by default instead of the native BIND 9 implementation. Please be aware that glibc versions 2.26 through 2.29 had a bug that could cause BIND 9 to deadlock. A fix was released in glibc 2.30, and most current Linux distributions have patched or updated glibc, with the notable exception of Ubuntu 18.04 (Bionic) which is a work in progress. If you are running on an affected operating system, compile BIND 9 with `--disable-pthread-rwlock` until a fixed version of glibc is available. **[GL !3125]**

8.32.3 Bug Fixes

- Fixed re-signing issues with inline zones which resulted in records being re-signed late or not at all.

8.33 Notes for BIND 9.16.0

Note: this section only lists changes from BIND 9.14 (the previous stable branch of BIND).

8.33.1 New Features

- A new asynchronous network communications system based on `libuv` is now used by `named` for listening for incoming requests and responding to them. This change will make it easier to improve performance and implement new protocol layers (for example, DNS over TLS) in the future. **[GL #29]**

- The new `dnssec-policy` option allows the configuration of a key and signing policy (KASP) for zones. This option enables `named` to generate new keys as needed and automatically roll both ZSK and KSK keys. (Note that the syntax for this statement differs from the DNSSEC policy used by `dnssec-keymgr`.) **[GL #1134]**

- In order to clarify the configuration of DNSSEC keys, the `trusted-keys` and `managed-keys` statements have been deprecated, and the new `trust-anchors` statement should now be used for both types of key.

  When used with the keyword `initial-key`, `trust-anchors` has the same behavior as `managed-keys`, i.e., it configures a trust anchor that is to be maintained via **RFC 5011**.

  When used with the new keyword `static-key`, `trust-anchors` has the same behavior as `trusted-keys`, i.e., it configures a permanent trust anchor that will not automatically be updated. (This usage is not recommended for the root key.) **[GL #6]**

- Two new keywords have been added to the `trust-anchors` statement: `initial-ds` and `static-ds`. These allow the use of trust anchors in DS format instead of DNSKEY format. DS format allows trust anchors to be configured for keys that have not yet been published; this is the format used by IANA when announcing future root keys.

As with the `initial-key` and `static-key` keywords, `initial-ds` configures a dynamic trust anchor to be maintained via **RFC 5011**, and `static-ds` configures a permanent trust anchor. **[GL #6] [GL #622]**

- `dig`, `mdig` and `delv` can all now take a `+yaml` option to print output in a detailed YAML format. **[GL #1145]**

- `dig` now has a new command line option: `+[no]unexpected`. By default, `dig` won't accept a reply from a source other than the one to which it sent the query. Add the `+unexpected` argument to enable it to process replies from unexpected sources. [RT #44978]

- `dig` now accepts a new command line option, `+[no]expandaaaa`, which causes the IPv6 addresses in AAAA records to be printed in full 128-bit notation rather than the default **RFC 5952** format. **[GL #765]**

- Statistics channel groups can now be toggled. **[GL #1030]**

8.33.2 Feature Changes

- When static and managed DNSSEC keys were both configured for the same name, or when a static key was used to configure a trust anchor for the root zone and `dnssec-validation` was set to the default value of `auto`, automatic **RFC 5011** key rollovers would be disabled. This combination of settings was never intended to work, but there was no check for it in the parser. This has been corrected, and it is now a fatal configuration error. **[GL #868]**

- DS and CDS records are now generated with SHA-256 digests only, instead of both SHA-1 and SHA-256. This affects the default output of `dnssec-dsfromkey`, the `dsset` files generated by `dnssec-signzone`, the DS records added to a zone by `dnssec-signzone` based on `keyset` files, the CDS records added to a zone by `named` and `dnssec-signzone` based on "sync" timing parameters in key files, and the checks performed by `dnssec-checkds`. **[GL #1015]**

- `named` will now log a warning if a static key is configured for the root zone. **[GL #6]**

- A SipHash 2-4 based DNS Cookie (**RFC 7873**) algorithm has been added and made default. Old non-default HMAC-SHA based DNS Cookie algorithms have been removed, and only the default AES algorithm is being kept for legacy reasons. This change has no operational impact in most common scenarios. **[GL #605]**

  If you are running multiple DNS servers (different versions of BIND 9 or DNS servers from multiple vendors) responding from the same IP address (anycast or load-balancing scenarios), make sure that all the servers are configured with the same DNS Cookie algorithm and same Server Secret for the best performance.

- The information from the `dnssec-signzone` and `dnssec-verify` commands is now printed to standard output. The standard error output is only used to print warnings and errors, and in case the user requests the signed zone to be printed to standard output with the `-f -` option. A new configuration option `-q` has been added to silence all output on standard output except for the name of the signed zone. **[GL #1151]**

- The DNSSEC validation code has been refactored for clarity and to reduce code duplication. **[GL #622]**

- Compile-time settings enabled by the `--with-tuning=large` option for `configure` are now in effect by default. Previously used default compile-time settings can be enabled by passing `--with-tuning=small` to `configure`. **[GL !2989]**

- JSON-C is now the only supported library for enabling JSON support for BIND statistics. The `configure` option has been renamed from `--with-libjson` to `--with-json-c`. Set the `PKG_CONFIG_PATH` environment variable accordingly to specify a custom path to the `json-c` library, as the new `configure` option does not take the library installation path as an optional argument. **[GL #855]**

- `./configure` no longer sets `--sysconfdir` to `/etc` or `--localstatedir` to `/var` when `--prefix` is not specified and the aforementioned options are not specified explicitly. Instead, Autoconf's defaults of `$prefix/etc` and `$prefix/var` are respected. **[GL #658]**

8.33.3 Removed Features

- The `dnssec-enable` option has been obsoleted and no longer has any effect. DNSSEC responses are always enabled if signatures and other DNSSEC data are present. [GL #866]

- DNSSEC Lookaside Validation (DLV) is now obsolete. The `dnssec-lookaside` option has been marked as deprecated; when used in `named.conf`, it will generate a warning but will otherwise be ignored. All code enabling the use of lookaside validation has been removed from the validator, `delv`, and the DNSSEC tools. [GL #7]

- The `cleaning-interval` option has been removed. [GL !1731]

8.34 License

BIND 9 is open source software licensed under the terms of the Mozilla Public License, version 2.0 (see the `COPYING` file for the full text).

Those wishing to discuss license compliance may contact ISC at https://www.isc.org/contact/.

8.35 End of Life

BIND 9.16 (Extended Support Version) will be supported until at least December, 2023. See https://kb.isc.org/docs/aa-00896 for details of ISC's software support policy.

8.36 Thank You

Thank you to everyone who assisted us in making this release possible.

DNSSEC GUIDE

9.1 Preface

9.1.1 Organization

This document provides introductory information on how DNSSEC works, how to configure BIND 9 to support some common DNSSEC features, and some basic troubleshooting tips. The chapters are organized as follows:

Introduction covers the intended audience for this document, assumed background knowledge, and a basic introduction to the topic of DNSSEC.

Getting Started covers various requirements before implementing DNSSEC, such as software versions, hardware capacity, network requirements, and security changes.

Validation walks through setting up a validating resolver, and gives both more information on the validation process and some examples of tools to verify that the resolver is properly validating answers.

Signing explains how to set up a basic signed authoritative zone, details the relationship between a child and a parent zone, and discusses ongoing maintenance tasks.

Basic DNSSEC Troubleshooting provides some tips on how to analyze and diagnose DNSSEC-related problems.

Advanced Discussions covers several topics, including key generation, key storage, key management, NSEC and NSEC3, and some disadvantages of DNSSEC.

Recipes provides several working examples of common DNSSEC solutions, with step-by-step details.

Commonly Asked Questions lists some commonly asked questions and answers about DNSSEC.

9.1.2 Acknowledgements

This document was originally authored by Josh Kuo of DeepDive Networking. He can be reached at josh.kuo@gmail.com.

Thanks to the following individuals (in no particular order) who have helped in completing this document: Jeremy C. Reed, Heidi Schempf, Stephen Morris, Jeff Osborn, Vicky Risk, Jim Martin, Evan Hunt, Mark Andrews, Michael McNally, Kelli Blucher, Chuck Aurora, Francis Dupont, Rob Nagy, Ray Bellis, Matthijs Mekking, and Suzanne Goldlust.

Special thanks goes to Cricket Liu and Matt Larson for their selflessness in knowledge sharing.

Thanks to all the reviewers and contributors, including John Allen, Jim Young, Tony Finch, Timothe Litt, and Dr. Jeffry A. Spain.

The sections on key rollover and key timing metadata borrowed heavily from the Internet Engineering Task Force draft titled "DNSSEC Key Timing Considerations" by S. Morris, J. Ihren, J. Dickinson, and W. Mekking, subsequently published as **RFC 7583**.

Icons made by Freepik and SimpleIcon from Flaticon, licensed under Creative Commons BY 3.0.

9.2 Introduction

9.2.1 Who Should Read this Guide?

This guide is intended as an introduction to DNSSEC for the DNS administrator who is already comfortable working with the existing BIND and DNS infrastructure. He or she might be curious about DNSSEC, but may not have had the time to investigate DNSSEC, to learn whether DNSSEC should be a part of his or her environment, and understand what it means to deploy it in the field.

This guide provides basic information on how to configure DNSSEC using BIND 9.16.0 or later. Most of the information and examples in this guide also apply to versions of BIND later than 9.9.0, but some of the key features described here were only introduced in version 9.16.0. Readers are assumed to have basic working knowledge of the Domain Name System (DNS) and related network infrastructure, such as concepts of TCP/IP. In-depth knowledge of DNS and TCP/IP is not required. The guide assumes no prior knowledge of DNSSEC or related technology such as public key cryptography.

9.2.2 Who May Not Want to Read this Guide?

If you are already operating a DNSSEC-signed zone, you may not learn much from the first half of this document, and you may want to start with *Advanced Discussions*. If you want to learn about details of the protocol extension, such as data fields and flags, or the new record types, this document can help you get started but it does not include all the technical details.

If you are experienced in DNSSEC, you may find some of the concepts in this document to be overly simplified for your taste, and some details are intentionally omitted at times for ease of illustration.

If you administer a large or complex BIND environment, this guide may not provide enough information for you, as it is intended to provide only basic, generic working examples.

If you are a top-level domain (TLD) operator, or administer zones under signed TLDs, this guide can help you get started, but it does not provide enough details to serve all of your needs.

If your DNS environment uses DNS products other than (or in addition to) BIND, this document may provide some background or overlapping information, but you should check each product's vendor documentation for specifics.

Finally, deploying DNSSEC on internal or private networks is not covered in this document, with the exception of a brief discussion in *DNSSEC on Private Networks*.

9.2.3 What is DNSSEC?

The Domain Name System (DNS) was designed in a day and age when the Internet was a friendly and trusting place. The protocol itself provides little protection against malicious or forged answers. DNS Security Extensions (DNSSEC) addresses this need, by adding digital signatures into DNS data so that each DNS response can be verified for integrity (the answer did not change during transit) and authenticity (the data came from the true source, not an impostor). In the ideal world, when DNSSEC is fully deployed, every single DNS answer can be validated and trusted.

DNSSEC does not provide a secure tunnel; it does not encrypt or hide DNS data. It operates independently of an existing Public Key Infrastructure (PKI). It does not need SSL certificates or shared secrets. It was designed with backwards compatibility in mind, and can be deployed without impacting "old" unsecured domain names.

DNSSEC is deployed on the three major components of the DNS infrastructure:

- *Recursive Servers*: People use recursive servers to lookup external domain names such as `www.example.com`. Operators of recursive servers need to enable DNSSEC validation. With validation enabled, recursive servers carry out additional tasks on each DNS response they receive to ensure its authenticity.

- *Authoritative Servers*: People who publish DNS data on their name servers need to sign that data. This entails creating additional resource records, and publishing them to parent domains where necessary. With DNSSEC enabled, authoritative servers respond to queries with additional DNS data, such as digital signatures and keys, in addition to the standard answers.

- *Applications*: This component lives on every client machine, from web servers to smart phones. This includes resolver libraries on different operating systems, and applications such as web browsers.

In this guide, we focus on the first two components, Recursive Servers and Authoritative Servers, and only lightly touch on the third component. We look at how DNSSEC works, how to configure a validating resolver, how to sign DNS zone data, and other operational tasks and considerations.

9.2.4 What Does DNSSEC Add to DNS?

Note: Public Key Cryptography works on the concept of a pair of keys: one made available to the world publicly, and one kept in secrecy privately. Not surprisingly, they are known as a public key and a private key. If you are not familiar with the concept, think of it as a cleverly designed lock, where one key locks and one key unlocks. In DNSSEC, we give out the unlocking public key to the rest of the world, while keeping the locking key private. To learn how this is used to secure DNS messages, see *How Are Answers Verified?*.

DNSSEC introduces eight new resource record types:

- RRSIG (digital resource record signature)

- DNSKEY (public key)

- DS (parent-child)

- NSEC (proof of nonexistence)

- NSEC3 (proof of nonexistence)

- NSEC3PARAM (proof of nonexistence)

- CDS (child-parent signaling)

- CDNSKEY (child-parent signaling)

This guide does not go deep into the anatomy of each resource record type; the details are left for the reader to research and explore. Below is a short introduction on each of the new record types:

- *RRSIG*: With DNSSEC enabled, just about every DNS answer (A, PTR, MX, SOA, DNSKEY, etc.) comes with at least one resource record signature, or RRSIG. These signatures are used by recursive name servers, also known as validating resolvers, to verify the answers received. To learn how digital signatures are generated and used, see *How Are Answers Verified?*.

- *DNSKEY*: DNSSEC relies on public-key cryptography for data authenticity and integrity. There are several keys used in DNSSEC, some private, some public. The public keys are published to the world as part of the zone data, and they are stored in the DNSKEY record type.

  In general, keys in DNSSEC are used for one or both of the following roles: as a Zone Signing Key (ZSK), used to protect all zone data; or as a Key Signing Key (KSK), used to protect the zone's keys. A key that is used for both roles is referred to as a Combined Signing Key (CSK). We talk about keys in more detail in *DNSSEC Keys*.

- *DS*: One of the critical components of DNSSEC is that the parent zone can "vouch" for its child zone. The DS record is verifiable information (generated from one of the child's public keys) that a parent zone publishes about its child as part of the chain of trust. To learn more about the Chain of Trust, see *Chain of Trust*.

- *NSEC, NSEC3, NSEC3PARAM*: These resource records all deal with a very interesting problem: proving that something does not exist. We look at these record types in more detail in *Proof of Non-Existence (NSEC and NSEC3)*.

- *CDS, CDNSKEY*: The CDS and CDNSKEY resource records apply to operational matters and are a way to signal to the parent zone that the DS records it holds for the child zone should be updated. This is covered in more detail in *The CDS and CDNSKEY Resource Records*.

9.2.5 How Does DNSSEC Change DNS Lookup?

Traditional (insecure) DNS lookup is simple: a recursive name server receives a query from a client to lookup a name like `www.isc.org`. The recursive name server tracks down the authoritative name server(s) responsible, sends the query to one of the authoritative name servers, and waits for it to respond with the answer.

With DNSSEC validation enabled, a validating recursive name server (a.k.a. a *validating resolver*) asks for additional resource records in its query, hoping the remote authoritative name servers respond with more than just the answer to the query, but some proof to go along with the answer as well. If DNSSEC responses are received, the validating resolver performs cryptographic computation to verify the authenticity (the origin of the data) and integrity (that the data was not altered during transit) of the answers, and even asks the parent zone as part of the verification. It repeats this process of get-key, validate, ask-parent, and its parent, and its parent, all the way until the validating resolver reaches a key that it trusts. In the ideal, fully deployed world of DNSSEC, all validating resolvers only need to trust one key: the root key.

The 12-Step DNSSEC Validation Process (Simplified)

The following example shows the 12 steps of the DNSSEC validating process at a very high level, looking up the name `www.isc.org`:

1. Upon receiving a DNS query from a client to resolve `www.isc.org`, the validating resolver follows standard DNS protocol to track down the name server for `isc.org`, and sends it a DNS query to ask for the A record of `www.isc.org`. But since this is a DNSSEC-enabled resolver, the outgoing query has a bit set indicating it wants DNSSEC answers, hoping the name server that receives it is DNSSEC-enabled and can honor this secure request.

2. The `isc.org` name server is DNSSEC-enabled, so it responds with both the answer (in this case, an A record) and a digital signature for verification purposes.

3. The validating resolver requires cryptographic keys to be able to verify the digital signature, so it asks the `isc.org` name server for those keys.

4. The `isc.org` name server responds with the cryptographic keys (and digital signatures of the keys) used to generate the digital signature that was sent in #2. At this point, the validating resolver can use this information to verify the answers received in #2.

   Let's take a quick break here and look at what we've got so far... how can our server trust this answer? If a clever attacker had taken over the `isc.org` name server(s), or course she would send matching keys and signatures. We need to ask someone else to have confidence that we are really talking to the real `isc.org` name server. This is a critical part of DNSSEC: at some point, the DNS administrators at `isc.org` uploaded some cryptographic information to its parent, `.org`, maybe through a secure web form, maybe through an email exchange, or perhaps in person. In any event, at some point some verifiable information about the child (`isc.org`) was sent to the parent (`.org`) for safekeeping.

5. The validating resolver asks the parent (`.org`) for the verifiable information it keeps on its child, `isc.org`.

6. Verifiable information is sent from the `.org` server. At this point, the validating resolver compares this to the answer it received in #4; if the two of them match, it proves the authenticity of `isc.org`.
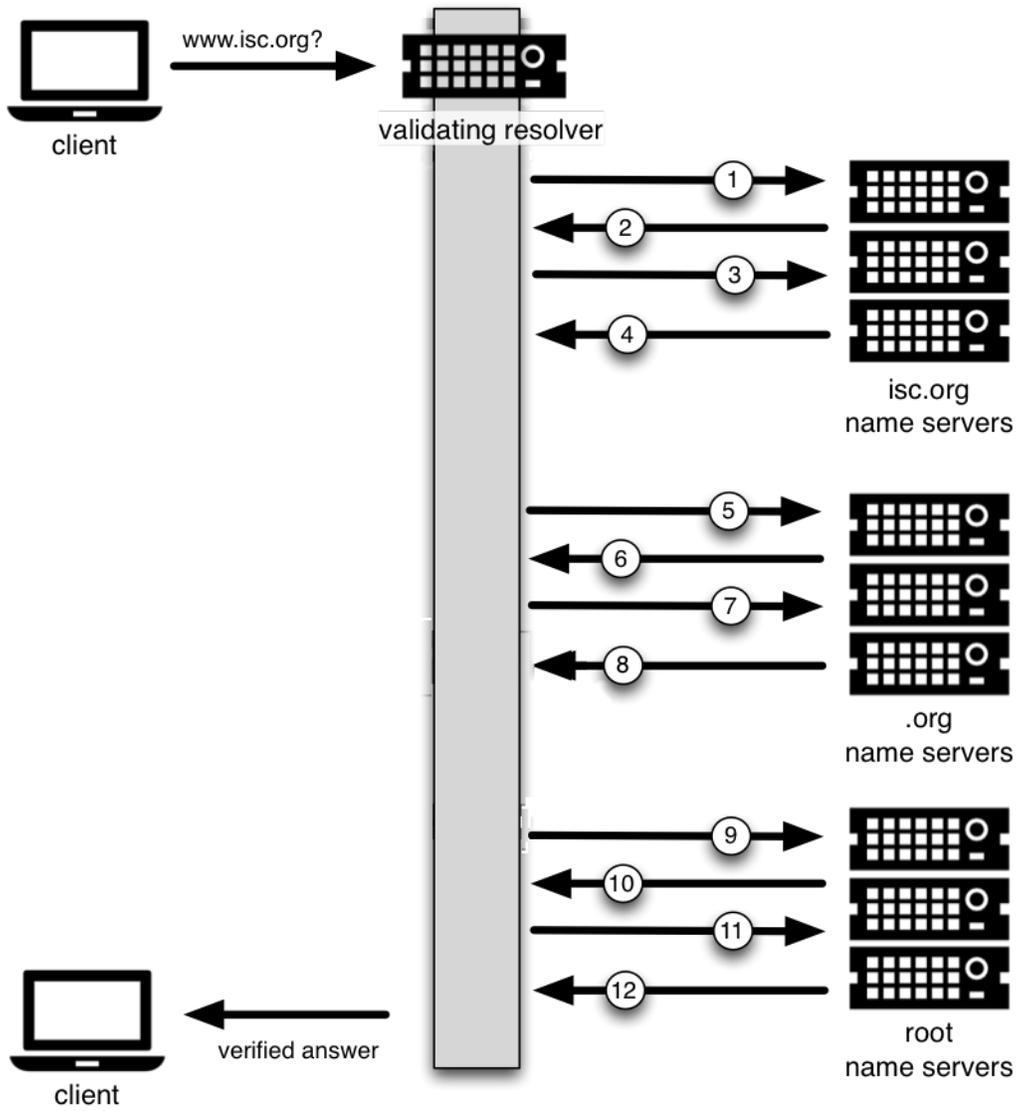
Let's examine this process. You might be thinking to yourself, what if the clever attacker that took over `isc.org` also compromised the `.org` servers? Of course all this information would match! That's why we turn our attention now to the `.org` server, interrogate it for its cryptographic keys, and move one level up to `.org`'s parent, root.

7. The validating resolver asks the `.org` authoritative name server for its cryptographic keys, to verify the answers received in #6.

8. The `.org` name server responds with the answer (in this case, keys and signatures). At this point, the validating resolver can verify the answers received in #6.

9. The validating resolver asks root (`.org`'s parent) for the verifiable information it keeps on its child, `.org`.

10. The root name server sends back the verifiable information it keeps on `.org`. The validating resolver uses this information to verify the answers received in #8.

    So at this point, both `isc.org` and `.org` check out. But what about root? What if this attacker is really clever and somehow tricked us into thinking she's the root name server? Of course she would send us all matching information! So we repeat the interrogation process and ask for the keys from the root name server.

11. The validating resolver asks the root name server for its cryptographic keys to verify the answer(s) received in #10.

12. The root name server sends its keys; at this point, the validating resolver can verify the answer(s) received in #10.

Chain of Trust

But what about the root server itself? Who do we go to verify root's keys? There's no parent zone for root. In security, you have to trust someone, and in the perfectly protected world of DNSSEC (we talk later about the current imperfect state and ways to work around it), each validating resolver would only have to trust one entity, that is, the root name server. The validating resolver already has the root key on file (we discuss later how we got the root key file). So after the answer in #12 is received, the validating resolver compares it to the key it already has on file. Providing one of the keys in the answer matches the one on file, we can trust the answer from root. Thus we can trust `.org`, and thus we can trust `isc.org`. This is known as the "chain of trust" in DNSSEC.

We revisit this 12-step process again later in *How Does DNSSEC Change DNS Lookup (Revisited)?* with more technical details.

9.2.6 Why is DNSSEC Important? (Why Should I Care?)

You might be thinking to yourself: all this DNSSEC stuff sounds wonderful, but why should I care? Below are some reasons why you may want to consider deploying DNSSEC:

1. *Being a good netizen*: By enabling DNSSEC validation (as described in *Validation*) on your DNS servers, you're protecting your users and yourself a little more by checking answers returned to you; by signing your zones (as described in *Signing*), you are making it possible for other people to verify your zone data. As more people adopt DNSSEC, the Internet as a whole becomes more secure for everyone.

2. *Compliance*: You may not even get a say in implementing DNSSEC, if your organization is subject to compliance standards that mandate it. For example, the US government set a deadline in 2008 to have all `.gov` subdomains signed by December 2009[1]. So if you operate a subdomain in `.gov`, you must implement DNSSEC to be compliant. ICANN also requires that all new top-level domains support DNSSEC.

3. *Enhanced Security*: Okay, so the big lofty goal of "let's be good" doesn't appeal to you, and you don't have any compliance standards to worry about. Here is a more practical reason why you should consider DNSSEC: in the event of a DNS-based security breach, such as cache poisoning or domain hijacking, after all the financial and brand damage done to your domain name, you might be placed under scrutiny for any preventive measure that could have been put in place. Think of this like having your website only available via HTTP but not HTTPS.

---

[1] The Office of Management and Budget (OMB) for the US government published a memo in 2008, requesting all `.gov` subdomains to be DNSSEC-signed by December 2009. This explains why `.gov` is the most-deployed DNSSEC domain currently, with around 90% of subdomains signed.

4. *New Features*: DNSSEC brings not only enhanced security, but also a whole new suite of features. Once DNS can be trusted completely, it becomes possible to publish SSL certificates in DNS, or PGP keys for fully automatic cross-platform email encryption, or SSH fingerprints…. New features are still being developed, but they all rely on a trustworthy DNS infrastructure. To take a peek at these next-generation DNS features, check out *Introduction to DANE*.

9.2.7 How Does DNSSEC Change My Job as a DNS Administrator?

With this protocol extension, some of the things you were used to in DNS have changed. As the DNS administrator, you have new maintenance tasks to perform on a regular basis (as described in *Maintenance Tasks*); when there is a DNS resolution problem, you have new troubleshooting techniques and tools to use (as described in *Basic DNSSEC Troubleshooting*). BIND 9 tries its best to make these things as transparent and seamless as possible. In this guide, we try to use configuration examples that result in the least amount of work for BIND 9 DNS administrators.

9.3 Getting Started

9.3.1 Software Requirements

BIND Version

Most configuration examples given in this document require BIND version 9.16.0 or newer (although many do work with all versions of BIND later than 9.9). To check the version of `named` you have installed, use the `-v` switch as shown below:

```
# named -v
BIND 9.16.0 (Stable Release) <id:6270e602ea>
```

Some configuration examples are added in BIND version 9.17 and backported to 9.16. For example, NSEC3 configuration requires BIND version 9.16.9.

We recommend you run the latest stable version to get the most complete DNSSEC configuration, as well as the latest security fixes.

DNSSEC Support in BIND

All versions of BIND 9 since BIND 9.7 can support DNSSEC, as currently deployed in the global DNS, so the BIND software you are running most likely already supports DNSSEC. Run the command `named -V` to see what flags it was built with. If it was built with OpenSSL (`--with-openssl`), then it supports DNSSEC. Below is an example of the output from running `named -V`:

```
$ named -V
BIND 9.16.0 (Stable Release) <id:6270e602ea>
running on Linux x86_64 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1+deb9u4 (2019-07-19)
built by make with defaults
compiled by GCC 6.3.0 20170516
compiled with OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019
linked to OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019
compiled with libxml2 version: 2.9.4
linked to libxml2 version: 20904
compiled with json-c version: 0.12.1
linked to json-c version: 0.12.1
```

(continues on next page)

```
compiled with zlib version: 1.2.8
linked to zlib version: 1.2.8
threads support is enabled

default paths:
  named configuration:   /usr/local/etc/named.conf
  rndc configuration:    /usr/local/etc/rndc.conf
  DNSSEC root key:       /usr/local/etc/bind.keys
  nsupdate session key:  /usr/local/var/run/named/session.key
  named PID file:        /usr/local/var/run/named/named.pid
  named lock file:       /usr/local/var/run/named/named.lock
```

If the BIND 9 software you have does not support DNSSEC, you should upgrade it. (It has not been possible to build BIND without DNSSEC support since BIND 9.13, released in 2018.) As well as missing out on DNSSEC support, you are also missing a number of security fixes made to the software in recent years.

System Entropy

To deploy DNSSEC to your authoritative server, you need to generate cryptographic keys. The amount of time it takes to generate the keys depends on the source of randomness, or entropy, on your systems. On some systems (especially virtual machines) with insufficient entropy, it may take much longer than one cares to wait to generate keys.

There are software packages, such as `haveged` for Linux, that provide additional entropy for a system. Once installed, they significantly reduce the time needed to generate keys.

The more entropy there is, the better pseudo-random numbers you get, and the stronger the keys that are generated. If you want or need high-quality random numbers, take a look at *Hardware Security Modules (HSMs)* for some of the hardware-based solutions.

9.3.2 Hardware Requirements

Recursive Server Hardware

Enabling DNSSEC validation on a recursive server makes it a *validating resolver*. The job of a validating resolver is to fetch additional information that can be used to computationally verify the answer set. Below are the areas that should be considered for possible hardware enhancement for a validating resolver:

1. *CPU*: a validating resolver executes cryptographic functions on many of the answers returned, which usually leads to increased CPU usage, unless your recursive server has built-in hardware to perform cryptographic computations.

2. *System memory*: DNSSEC leads to larger answer sets and occupies more memory space.

3. *Network interfaces*: although DNSSEC does increase the amount of DNS traffic overall, it is unlikely that you need to upgrade your network interface card (NIC) on the name server unless you have some truly outdated hardware.

One factor to consider is the destinations of your current DNS traffic. If your current users spend a lot of time visiting `.gov` websites, you should expect a jump in all of the above categories when validation is enabled, because `.gov` is more than 90% signed. This means that more than 90% of the time, your validating resolver will be doing what is described in *How Does DNSSEC Change DNS Lookup?*. However, if your users only care about resources in the `.com` domain, which, as of mid-2020, is under 1.5% signed[2], your recursive name server is unlikely to experience a significant load increase after enabling DNSSEC validation.

---

[2] https://rick.eng.br/dnssecstat

Authoritative Server Hardware

On the authoritative server side, DNSSEC is enabled on a zone-by-zone basis. When a zone is DNSSEC-enabled, it is also known as "signed." Below are the areas to consider for possible hardware enhancements for an authoritative server with signed zones:

1. *CPU*: a DNSSEC-signed zone requires periodic re-signing, which is a cryptographic function that is CPU-intensive. If your DNS zone is dynamic or changes frequently, that also adds to higher CPU loads.

2. *System storage*: A signed zone is definitely larger than an unsigned zone. How much larger? See *Your Zone, Before and After DNSSEC* for a comparison example. Roughly speaking, you should expect your zone file to grow by at least three times, and frequently more.

3. *System memory*: Larger DNS zone files take up not only more storage space on the file system, but also more space when they are loaded into system memory.

4. *Network interfaces*: While your authoritative name servers will begin sending back larger responses, it is unlikely that you need to upgrade your network interface card (NIC) on the name server unless you have some truly outdated hardware.

One factor to consider, but over which you really have no control, is the number of users who query your domain name who themselves have DNSSEC enabled. It was estimated in late 2014 that roughly 10% to 15% of the Internet DNS queries were DNSSEC-aware. Estimates by APNIC suggest that in 2020 about one-third of all queries are validating queries, although the percentage varies widely on a per-country basis. This means that more DNS queries for your domain will take advantage of the additional security features, which will result in increased system load and possibly network traffic.

9.3.3 Network Requirements

From a network perspective, DNS and DNSSEC packets are very similar; DNSSEC packets are just bigger, which means DNS is more likely to use TCP. You should test for the following two items to make sure your network is ready for DNSSEC:

1. *DNS over TCP*: Verify network connectivity over TCP port 53, which may mean updating firewall policies or Access Control Lists (ACL) on routers. See *Wait… DNS Uses TCP?* for more details.

2. *Large UDP packets*: Some network equipment, such as firewalls, may make assumptions about the size of DNS UDP packets and incorrectly reject DNS traffic that appears "too big." Verify that the responses your name server generates are being seen by the rest of the world: see *What's EDNS All About (And Why Should I Care)?* for more details.

9.3.4 Operational Requirements

Parent Zone

Before starting your DNSSEC deployment, check with your parent zone administrators to make sure they support DNSSEC. This may or may not be the same entity as your registrar. As you will see later in *Working With the Parent Zone*, a crucial step in DNSSEC deployment is establishing the parent-child trust relationship. If your parent zone does not yet support DNSSEC, contact that administrator to voice your concerns.

Security Requirements

Some organizations may be subject to stricter security requirements than others. Check to see if your organization requires stronger cryptographic keys be generated and stored, and how often keys need to be rotated. The examples presented in this document are not intended for high-value zones. We cover some of these security considerations in *Advanced Discussions*.

9.4 Validation

9.4.1 Easy-Start Guide for Recursive Servers

This section provides the basic information needed to set up a working DNSSEC-aware recursive server, also known as a validating resolver. A validating resolver performs validation for each remote response received, following the chain of trust to verify that the answers it receives are legitimate, through the use of public key cryptography and hashing functions.

Enabling DNSSEC Validation

So how do we turn on DNSSEC validation? It turns out that you may not need to reconfigure your name server at all, since the most recent versions of BIND 9 - including packages and distributions - have shipped with DNSSEC validation enabled by default. Before making any configuration changes, check whether you already have DNSSEC validation enabled by following the steps described in *So You Think You Are Validating (How To Test A Recursive Server)*.

In earlier versions of BIND, including 9.11-ESV, DNSSEC validation must be explicitly enabled. To do this, you only need to add one line to the `options` section of your configuration file:

```
options {
    ...
    dnssec-validation auto;
    ...
};
```

Restart `named` or run `rndc reconfig`, and your recursive server is now happily validating each DNS response. If this does not work for you, and you have already verified DNSSEC support as described in *DNSSEC Support in BIND*, you may have some other network-related configurations that need to be adjusted. Take a look at *Network Requirements* to make sure your network is ready for DNSSEC.

Effects of Enabling DNSSEC Validation

Once DNSSEC validation is enabled, any DNS response that does not pass the validation checks results in a failure to resolve the domain name (often a SERVFAIL status seen by the client). If everything has been configured properly, this is the correct result; it means that an end user has been protected against a malicious attack.

However, if there is a DNSSEC configuration issue (sometimes outside of the administrator's control), a specific name or sometimes entire domains may "disappear" from the DNS, and become unreachable through that resolver. For the end user, the issue may manifest itself as name resolution being slow or failing altogether; some parts of a URL not loading; or the web browser returning an error message indicating that the page cannot be displayed. For example, if root name servers were misconfigured with the wrong information about `.org`, it could cause all validation for `.org` domains to fail. To end users, it would appear that all `.org` web sites were out of service[3]. Should you encounter DNSSEC-related problems, don't be tempted to disable validation; there is almost certainly a solution that leaves validation enabled. A basic troubleshooting guide can be found in *Basic DNSSEC Troubleshooting*.

---

[3] Of course, something like this could happen for reasons other than DNSSEC: for example, the root publishing the wrong addresses for the `.org` nameservers.

9.4.2 So You Think You Are Validating (How To Test A Recursive Server)

Now that you have reconfigured your recursive server and restarted it, how do you know that your recursive name server is actually verifying each DNS query? There are several ways to check, and we've listed a few of them below.

Using Web-Based Tools to Verify

For most people, the simplest way to check if a recursive name server is indeed validating DNS queries is to use one of the many web-based tools available.

Configure your client computer to use the newly reconfigured recursive server for DNS resolution; then use one of these web-based tests to confirm that it is in fact validating DNS responses.

- Internet.nl
- DNSSEC Resolver Test (uni-due.de)
- DNSSEC or Not (VeriSign)

Using `dig` to Verify

Web-based DNSSEC-verification tools often employ JavaScript. If you don't trust the JavaScript magic that the web-based tools rely on, you can take matters into your own hands and use a command-line DNS tool to check your validating resolver yourself.

While `nslookup` is popular, partly because it comes pre-installed on most systems, it is not DNSSEC-aware. `dig`, on the other hand, fully supports the DNSSEC standard and comes as a part of BIND. If you do not have `dig` already installed on your system, install it by downloading it from ISC's website. ISC provides pre-compiled Windows versions on its website.

`dig` is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name servers that were queried. Most seasoned DNS administrators use `dig` to troubleshoot DNS problems because of its flexibility, ease of use, and clarity of output.

The example below shows how to use `dig` to query the name server 10.53.0.1 for the A record for `ftp.isc.org` when DNSSEC validation is enabled (i.e. the default). The address 10.53.0.1 is only used as an example; replace it with the actual address or host name of your recursive name server.

```
$ dig @10.53.0.1 ftp.isc.org. A +dnssec +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.1 ftp.isc.org a +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48742
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 29a9705c2160b08c010000005e67a4a102b9ae079c1b24c8 (good)
;; QUESTION SECTION:
;ftp.isc.org.        IN A

;; ANSWER SECTION:
ftp.isc.org.        300 IN A 149.20.1.49
ftp.isc.org.        300 IN RRSIG A 13 3 300 (
                20200401191851 20200302184340 27566 isc.org.
```

(continues on next page)

```
                    e9Vkb6/6aHMQk/t23Im71ioiDUhB06sncsduoW9+Asl4
                    L3TZtpLvZ5+zudTJC2coI4D/D9AXte1cD6FV6iS6PQ== )

;; Query time: 452 msec
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 14:30:57 GMT 2020
;; MSG SIZE  rcvd: 187
```

The important detail in this output is the presence of the `ad` flag in the header. This signifies that BIND has retrieved all related DNSSEC information related to the target of the query (`ftp.isc.org`) and that the answer received has passed the validation process described in *How Are Answers Verified?*. We can have confidence in the authenticity and integrity of the answer, that `ftp.isc.org` really points to the IP address 149.20.1.49, and that it was not a spoofed answer from a clever attacker.

Unlike earlier versions of BIND, the current versions of BIND always request DNSSEC records (by setting the `do` bit in the query they make to upstream servers), regardless of DNSSEC settings. However, with validation disabled, the returned signature is not checked. This can be seen by explicitly disabling DNSSEC validation. To do this, add the line `dnssec-validation no;` to the "options" section of the configuration file, i.e.:

```
options {
    ...
    dnssec-validation no;
    ...
};
```

If the server is restarted (to ensure a clean cache) and the same `dig` command executed, the result is very similar:

```
$ dig @10.53.0.1 ftp.isc.org. A +dnssec +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.1 ftp.isc.org a +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39050
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: a8dc9d1b9ec45e75010000005e67a8a69399741fdbe126f2 (good)
;; QUESTION SECTION:
;ftp.isc.org.        IN A

;; ANSWER SECTION:
ftp.isc.org.        300 IN A 149.20.1.49
ftp.isc.org.        300 IN RRSIG A 13 3 300 (
                    20200401191851 20200302184340 27566 isc.org.
                    e9Vkb6/6aHMQk/t23Im71ioiDUhB06sncsduoW9+Asl4
                    L3TZtpLvZ5+zudTJC2coI4D/D9AXte1cD6FV6iS6PQ== )

;; Query time: 261 msec
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 14:48:06 GMT 2020
;; MSG SIZE  rcvd: 187
```

However, this time there is no `ad` flag in the header. Although `dig` is still returning the DNSSEC-related resource records, it is not checking them, and thus cannot vouch for the authenticity of the answer. If you do carry out this test, remember to re-enable DNSSEC validation (by removing the `dnssec-validation no;` line from the configuration

file) before continuing.

9.4.3 Verifying Protection From Bad Domain Names

It is also important to make sure that DNSSEC is protecting your network from domain names that fail to validate; such failures could be caused by attacks on your system, attempting to get it to accept false DNS information. Validation could fail for a number of reasons: maybe the answer doesn't verify because it's a spoofed response; maybe the signature was a replayed network attack that has expired; or maybe the child zone has been compromised along with its keys, and the parent zone's information tells us that things don't add up. There is a domain name specifically set up to fail DNSSEC validation, `www.dnssec-failed.org`.

With DNSSEC validation enabled (the default), an attempt to look up that name fails:

```
$ dig @10.53.0.1 www.dnssec-failed.org. A

; <<>> DiG 9.16.0 <<>> @10.53.0.1 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 22667
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 69c30831448545870100000005e67bb57f5f90ff2688e455d (good)
;; QUESTION SECTION:
;www.dnssec-failed.org.      IN  A

;; Query time: 2763 msec
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 16:07:51 GMT 2020
;; MSG SIZE  rcvd: 78
```

On the other hand, if DNSSEC validation is disabled (by adding the statement `dnssec-validation no;` to the `options` clause in the configuration file), the lookup succeeds:

```
$ dig @10.53.0.1 www.dnssec-failed.org. A

; <<>> DiG 9.16.0 <<>> @10.53.0.1 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54704
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 251eee58208917f90100000005e67bb6829f6dabc5ae6b7b9 (good)
;; QUESTION SECTION:
;www.dnssec-failed.org.      IN A

;; ANSWER SECTION:
www.dnssec-failed.org.  7200    IN  A   68.87.109.242
www.dnssec-failed.org.  7200    IN  A   69.252.193.191

;; Query time: 439 msec
```

```
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 16:08:08 GMT 2020
;; MSG SIZE  rcvd: 110
```

Do not be tempted to disable DNSSEC validation just because some names are failing to resolve. Remember, DNSSEC protects your DNS lookup from hacking. The next section describes how to quickly check whether the failure to success-fully look up a name is due to a validation failure.

How Do I Know I Have a Validation Problem?

Since all DNSSEC validation failures result in a general `SERVFAIL` message, how do we know if it was really a validation error? Fortunately, there is a flag in `dig`, (`+cd`, for "checking disabled") which tells the server to disable DNSSEC validation. If you receive a `SERVFAIL` message, re-run the query a second time and set the `+cd` flag. If the query succeeds with `+cd`, but ends in `SERVFAIL` without it, you know you are dealing with a validation problem. So using the previous example of `www.dnssec-failed.org` and with DNSSEC validation enabled in the resolver:

```
$ dig @10.53.0.1 www.dnssec-failed.org A +cd

; <<>> DiG 9.16.0 <<>> @10.53.0.1 www.dnssec-failed.org. A +cd
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62313
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 73ca1be3a74dd2cf010000005e67c8c8e6df64b519cd87fd (good)
;; QUESTION SECTION:
;www.dnssec-failed.org.      IN  A

;; ANSWER SECTION:
www.dnssec-failed.org.  7197    IN  A   68.87.109.242
www.dnssec-failed.org.  7197    IN  A   69.252.193.191

;; Query time: 0 msec
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 17:05:12 GMT 2020
;; MSG SIZE  rcvd: 110
```

For more information on troubleshooting, please see *Basic DNSSEC Troubleshooting*.

9.4.4 Validation Easy Start Explained

In *Easy-Start Guide for Recursive Servers*, we used one line of configuration to turn on DNSSEC validation: the act of chasing down signatures and keys, making sure they are authentic. Now we are going to take a closer look at what DNSSEC validation actually does, and some other options.

dnssec-validation

```
options {
    dnssec-validation auto;
};
```

This "auto" line enables automatic DNSSEC trust anchor configuration using the `managed-keys` feature. In this case, no manual key configuration is needed. There are three possible choices for the `dnssec-validation` option:

- *yes*: DNSSEC validation is enabled, but a trust anchor must be manually configured. No validation actually takes place until at least one trusted key has been manually configured.

- *no*: DNSSEC validation is disabled, and the recursive server behaves in the "old-fashioned" way of performing insecure DNS lookups.

- *auto*: DNSSEC validation is enabled, and a default trust anchor (included as part of BIND 9) for the DNS root zone is used. This is the default; BIND automatically does this if there is no `dnssec-validation` line in the configuration file.

Let's discuss the difference between *yes* and *auto*. If set to *yes*, the trust anchor must be manually defined and maintained using the `trust-anchors` statement (with either the `static-key` or `static-ds` modifier) in the configuration file; if set to *auto* (the default, and as shown in the example), then no further action should be required as BIND includes a copy[4] of the root key. When set to *auto*, BIND automatically keeps the keys (also known as trust anchors, discussed in *Trust Anchors*) up-to-date without intervention from the DNS administrator.

We recommend using the default *auto* unless there is a good reason to require a manual trust anchor. To learn more about trust anchors, please refer to *Trusted Keys and Managed Keys*.

How Does DNSSEC Change DNS Lookup (Revisited)?

Now you've enabled validation on your recursive name server and verified that it works. What exactly changed? In *How Does DNSSEC Change DNS Lookup?* we looked at a very high-level, simplified version of the 12 steps of the DNSSEC validation process. Let's revisit that process now and see what your validating resolver is doing in more detail. Again, as an example we are looking up the A record for the domain name `www.isc.org` (see *The 12-Step DNSSEC Validation Process (Simplified)*):

1. The validating resolver queries the `isc.org` name servers for the A record of `www.isc.org`. This query has the `DNSSEC OK (do)` bit set to 1, notifying the remote authoritative server that DNSSEC answers are desired.

2. Since the zone `isc.org` is signed, and its name servers are DNSSEC-aware, it responds with the answer to the A record query plus the RRSIG for the A record.

3. The validating resolver queries for the DNSKEY for `isc.org`.

4. The `isc.org` name server responds with the DNSKEY and RRSIG records. The DNSKEY is used to verify the answers received in #2.

5. The validating resolver queries the parent (`.org`) for the DS record for `isc.org`.

6. The `.org` name server is also DNSSEC-aware, so it responds with the DS and RRSIG records. The DS record is used to verify the answers received in #4.

7. The validating resolver queries for the DNSKEY for `.org`.

8. The `.org` name server responds with its DNSKEY and RRSIG. The DNSKEY is used to verify the answers received in #6.

9. The validating resolver queries the parent (root) for the DS record for `.org`.

---

[4] BIND technically includes two copies of the root key: one is in `bind.keys.h` and is built into the executable, and one is in `bind.keys` as a `trust-anchors` statement. The two copies of the key are identical.

10. The root name server, being DNSSEC-aware, responds with DS and RRSIG records. The DS record is used to verify the answers received in #8.

11. The validating resolver queries for the DNSKEY for root.

12. The root name server responds with its DNSKEY and RRSIG. The DNSKEY is used to verify the answers received in #10.

After step #12, the validating resolver takes the DNSKEY received and compares it to the key or keys it has configured, to decide whether the received key can be trusted. We talk about these locally configured keys, or trust anchors, in *Trust Anchors*.

With DNSSEC, every response includes not just the answer, but a digital signature (RRSIG) as well, so the validating resolver can verify the answer received. That is what we look at in the next section, *How Are Answers Verified?*.

How Are Answers Verified?

Note: Keep in mind, as you read this section, that although words like "encryption" and "decryption" are used here from time to time, DNSSEC does not provide privacy. Public key cryptography is used to verify data *authenticity* (who sent it) and data *integrity* (it did not change during transit), but any eavesdropper can still see DNS requests and responses in clear text, even when DNSSEC is enabled.

So how exactly are DNSSEC answers verified? Let's first see how verifiable information is generated. On the authoritative server, each DNS record (or message) is run through a hash function, and this hashed value is then encrypted by a private key. This encrypted hash value is the digital signature.
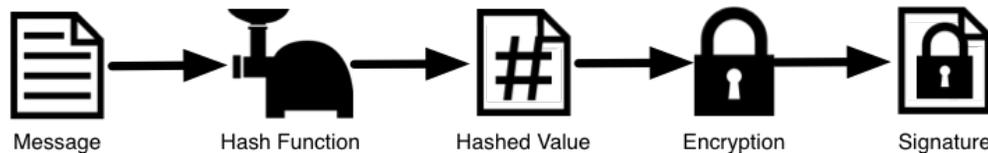
Fig. 1: Signature Generation

When the validating resolver queries for the resource record, it receives both the plain-text message and the digital signature(s). The validating resolver knows the hash function used (it is listed in the digital signature record itself), so it can take the plain-text message and run it through the same hash function to produce a hashed value, which we'll call hash value X. The validating resolver can also obtain the public key (published as DNSKEY records), decrypt the digital signature, and get back the original hashed value produced by the authoritative server, which we'll call hash value Y. If hash values X and Y are identical, and the time is correct (more on what this means below), the answer is verified, meaning this answer came from the authoritative server (authenticity), and the content remained intact during transit (integrity).

Take the A record `ftp.isc.org`, for example. The plain text is:

```
ftp.isc.org.     4 IN A  149.20.1.49
```

The digital signature portion is:

```
ftp.isc.org.      300 IN RRSIG A 13 3 300 (
             20200401191851 20200302184340 27566 isc.org.
             e9Vkb6/6aHMQk/t23Im71ioiDUhB06sncsduoW9+Asl4
             L3TZtpLvZ5+zudTJC2coI4D/D9AXte1cD6FV6iS6PQ== )
```

When a validating resolver queries for the A record `ftp.isc.org`, it receives both the A record and the RRSIG record. It runs the A record through a hash function (in this example, SHA256 as indicated by the number 13, signifying
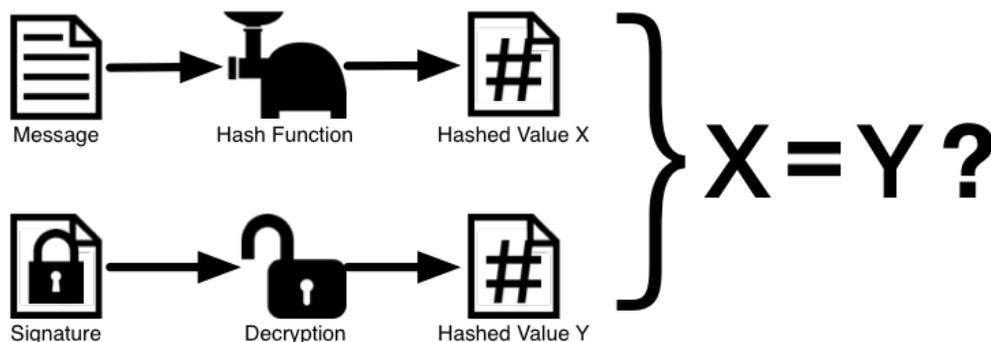
Fig. 2: Signature Verification

ECDSAP256SHA256) and produces hash value X. The resolver also fetches the appropriate DNSKEY record to decrypt the signature, and the result of the decryption is hash value Y.

But wait, there's more! Just because X equals Y doesn't mean everything is good. We still have to look at the time. Remember we mentioned a little earlier that we need to check if the time is correct? Look at the two timestamps in our example above:

- Signature Expiration: 20200401191851

- Signature Inception: 20200302184340

This tells us that this signature was generated UTC March 2nd, 2020, at 6:43:40 PM (20200302184340), and it is good until UTC April 1st, 2020, 7:18:51 PM (20200401191851). The validating resolver's current system time needs to fall between these two timestamps. If it does not, the validation fails, because it could be an attacker replaying an old captured answer set from the past, or feeding us a crafted one with incorrect future timestamps.

If the answer passes both the hash value check and the timestamp check, it is validated and the authenticated data (`ad`) bit is set, and the response is sent to the client; if it does not verify, a SERVFAIL is returned to the client.

9.4.5 Trust Anchors

A trust anchor is a key that is placed into a validating resolver, so that the validator can verify the results of a given request with a known or trusted public key (the trust anchor). A validating resolver must have at least one trust anchor installed to perform DNSSEC validation.

9.4.6 How Trust Anchors are Used

In the section *How Does DNSSEC Change DNS Lookup (Revisited)?*, we walked through the 12 steps of the DNSSEC lookup process. At the end of the 12 steps, a critical comparison happens: the key received from the remote server and the key we have on file are compared to see if we trust it. The key we have on file is called a trust anchor, sometimes also known as a trust key, trust point, or secure entry point.

The 12-step lookup process describes the DNSSEC lookup in the ideal world, where every single domain name is signed and properly delegated, and where each validating resolver only needs to have one trust anchor - that is, the root's public key. But there is no restriction that the validating resolver must only have one trust anchor. In fact, in the early stages of DNSSEC adoption, it was not unusual for a validating resolver to have more than one trust anchor.

For instance, before the root zone was signed (in July 2010), some validating resolvers that wished to validate domain names in the `.gov` zone needed to obtain and install the key for `.gov`. A sample lookup process for `www.fbi.gov` at that time would have been eight steps rather than 12:
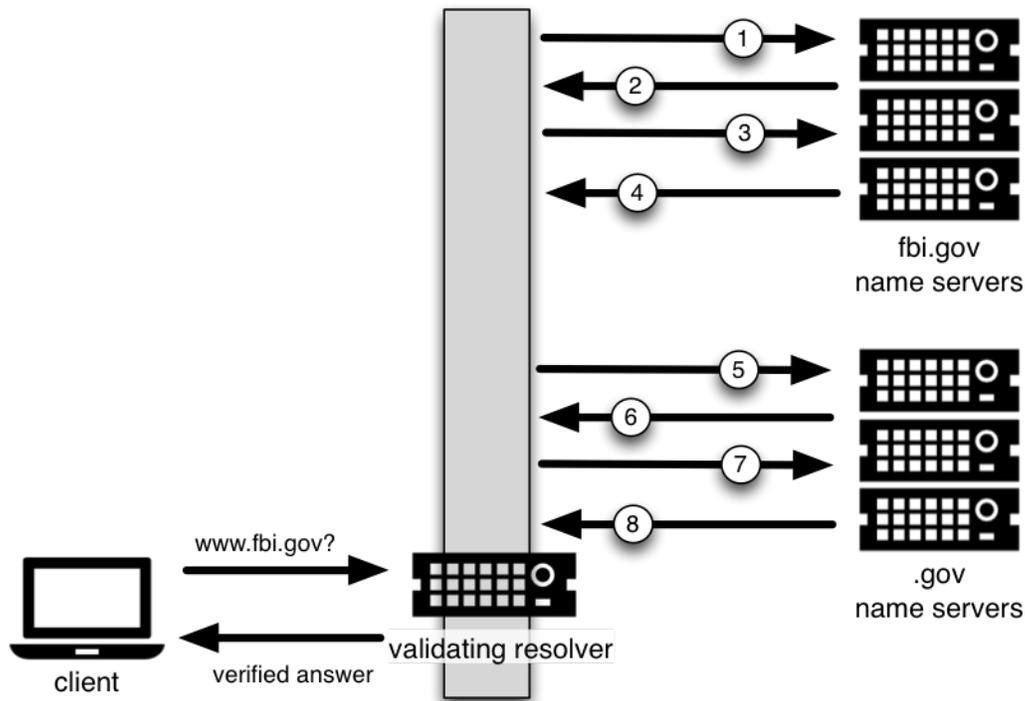
1. The validating resolver queried `fbi.gov` name server for the A record of `www.fbi.gov`.

2. The FBI's name server responded with the answer and its RRSIG.

3. The validating resolver queried the FBI's name server for its DNSKEY.

4. The FBI's name server responded with the DNSKEY and its RRSIG.

5. The validating resolver queried a `.gov` name server for the DS record of `fbi.gov`.

6. The `.gov` name server responded with the DS record and the associated RRSIG for `fbi.gov`.

7. The validating resolver queried the `.gov` name server for its DNSKEY.

8. The `.gov` name server responded with its DNSKEY and the associated RRSIG.

This all looks very similar, except it's shorter than the 12 steps that we saw earlier. Once the validating resolver receives the DNSKEY file in #8, it recognizes that this is the manually configured trusted key (trust anchor), and never goes to the root name servers to ask for the DS record for `.gov`, or ask the root name servers for their DNSKEY.

In fact, whenever the validating resolver receives a DNSKEY, it checks to see if this is a configured trusted key to decide whether it needs to continue chasing down the validation chain.

Trusted Keys and Managed Keys

Since the resolver is validating, we must have at least one key (trust anchor) configured. How did it get here, and how do we maintain it?

If you followed the recommendation in *Easy-Start Guide for Recursive Servers*, by setting `dnssec-validation` to *auto*, there is nothing left to do. BIND already includes a copy of the root key (in the file `bind.keys`), and automatically updates it when the root key changes.[5] It looks something like this:

```
trust-anchors {
        # This key (20326) was published in the root zone in 2017.
        . initial-key 257 3 8 "AwEAAaz/
→tAm8yTn4Mfeh5eyI96WSVexTBAvkMgJzkKTOiW1vkIbzxeF3
                +/4RgWOq7HrxRixHlFlExOLAJr5emLvN7SWXgnLh4+B5xQlNVz8Og8kv
                ArMtNROxVQuCaSnIDdD5LKyWbRd2n9WGe2R8PzgCmr3EgVLrjyBxWezF
                0jLHwVN8efS3rCj/EWgvIWgb9tarpVUDK/b58Da+sqqls3eNbuv7pr+e
                oZG+SrDK6nWeL3c6H5Apxz7LjVc1uTIdsIXxuOLYA4/ilBmSVIzuDWfd
                RUfhHdY6+cn8HFRm+2hM8AnXGXws9555KrUB5qihylGa8subX2Nn6UwN
                R1AkUTV74bU=";
};
```

You can, of course, decide to manage this key manually yourself.   First, you need to make sure that `dnssec-validation` is set to *yes* rather than *auto*:

```
options {
    dnssec-validation yes;
};
```

Then, download the root key manually from a trustworthy source, such as https://www.isc.org/bind-keys. Finally, take the root key you manually downloaded and put it into a `trust-anchors` statement as shown below:

```
trust-anchors {
        # This key (20326) was published in the root zone in 2017.
        . static-key 257 3 8 "AwEAAaz/tAm8yTn4Mfeh5eyI96WSVexTBAvkMgJzkKTOiW1vkIbzxeF3
                +/4RgWOq7HrxRixHlFlExOLAJr5emLvN7SWXgnLh4+B5xQlNVz8Og8kv
                ArMtNROxVQuCaSnIDdD5LKyWbRd2n9WGe2R8PzgCmr3EgVLrjyBxWezF
                0jLHwVN8efS3rCj/EWgvIWgb9tarpVUDK/b58Da+sqqls3eNbuv7pr+e
                oZG+SrDK6nWeL3c6H5Apxz7LjVc1uTIdsIXxuOLYA4/ilBmSVIzuDWfd
                RUfhHdY6+cn8HFRm+2hM8AnXGXws9555KrUB5qihylGa8subX2Nn6UwN
                R1AkUTV74bU=";
};
```

While this `trust-anchors` statement and the one in the `bind.keys` file appear similar, the definition of the key in `bind.keys` has the `initial-key` modifier, whereas in the statement in the configuration file, that is replaced by `static-key`. There is an important difference between the two: a key defined with `static-key` is always trusted until it is deleted from the configuration file. With the `initial-key` modified, keys are only trusted once: for as long as it takes to load the managed key database and start the key maintenance process. Thereafter, BIND uses the managed keys database (`managed-keys.bind.jnl`) as the source of key information.

> **Warning:**   Remember, if you choose to manage the keys on your own, whenever the key changes (which, for most zones, happens on a periodic basis), the configuration needs to be updated manually.   Failure to do so will result in breaking nearly all DNS queries for the subdomain of the key.   So if you are manually managing `.gov`, all domain

---

[5] The root zone was signed in July 2010 and, as at the time of this writing (mid-2020), the key has been changed once, in October 2018. The intention going forward is to roll the key once every five years.

names in the `.gov` space may become unresolvable; if you are manually managing the root key, you could break all DNS requests made to your recursive name server.

Explicit management of keys was common in the early days of DNSSEC, when neither the root zone nor many top-level domains were signed. Since then, over 90% of the top-level domains have been signed, including all the largest ones. Unless you have a particular need to manage keys yourself, it is best to use the BIND defaults and let the software manage the root key.

9.4.7 What's EDNS All About (And Why Should I Care)?

EDNS Overview

Traditional DNS responses are typically small in size (less than 512 bytes) and fit nicely into a small UDP packet. The Extension mechanism for DNS (EDNS, or EDNS(0)) offers a mechanism to send DNS data in larger packets over UDP. To support EDNS, both the DNS server and the network need to be properly prepared to support the larger packet sizes and multiple fragments.

This is important for DNSSEC, since the `+do` bit that signals DNSSEC-awareness is carried within EDNS, and DNSSEC responses are larger than traditional DNS ones. If DNS servers and the network environment cannot support large UDP packets, it will cause retransmission over TCP, or the larger UDP responses will be discarded. Users will likely experience slow DNS resolution or be unable to resolve certain names at all.

Note that EDNS applies regardless of whether you are validating DNSSEC, because BIND has DNSSEC enabled by default.

Please see *Network Requirements* for more information on what DNSSEC expects from the network environment.

EDNS on DNS Servers

For many years, BIND has had EDNS enabled by default, and the UDP packet size is set to a maximum of 4096 bytes. The DNS administrator should not need to perform any reconfiguration. You can use `dig` to verify that your server supports EDNS and see the UDP packet size it allows with this `dig` command:

```
$ dig @10.53.0.1 www.isc.org. A +dnssec +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.1 ftp.isc.org a +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48742
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 29a9705c2160b08c010000005e67a4a102b9ae079c1b24c8 (good)
;; QUESTION SECTION:
;ftp.isc.org.        IN A

;; ANSWER SECTION:
ftp.isc.org.        300 IN A 149.20.1.49
ftp.isc.org.        300 IN RRSIG A 13 3 300 (
                20200401191851 20200302184340 27566 isc.org.
                e9Vkb6/6aHMQk/t23Im71ioiDUhB06sncsduoW9+Asl4
```

(continues on next page)

```
                L3TZtpLvZ5+zudTJC2coI4D/D9AXte1cD6FV6iS6PQ== )

;; Query time: 452 msec
;; SERVER: 10.53.0.1#53(10.53.0.1)
;; WHEN: Tue Mar 10 14:30:57 GMT 2020
;; MSG SIZE  rcvd: 187
```

There is a helpful testing tool available (provided by DNS-OARC) that you can use to verify resolver behavior regarding EDNS support: https://www.dns-oarc.net/oarc/services/replysizetest/ .

Once you've verified that your name servers have EDNS enabled, that should be the end of the story, right? Unfortunately, EDNS is a hop-by-hop extension to DNS. This means the use of EDNS is negotiated between each pair of hosts in a DNS resolution process, which in turn means if one of your upstream name servers (for instance, your ISP's recursive name server that your name server forwards to) does not support EDNS, you may experience DNS lookup failures or be unable to perform DNSSEC validation.

Support for Large Packets on Network Equipment

If both your recursive name server and your ISP's name servers support EDNS, we are all good here, right? Not so fast. Since these large packets have to traverse the network, the network infrastructure itself must allow them to pass.

When data is physically transmitted over a network, it has to be broken down into chunks. The size of the data chunk is known as the Maximum Transmission Unit (MTU), and it can differ from network to network. IP fragmentation occurs when a large data packet needs to be broken down into chunks smaller than the MTU; these smaller chunks then need to be reassembled back into the large data packet at their destination. IP fragmentation is not necessarily a bad thing, and it most likely occurs on your network today.

Some network equipment, such as a firewall, may make assumptions about DNS traffic. One of these assumptions may be how large each DNS packet is. When a firewall sees a larger DNS packet than it expects, it may either reject the large packet or drop its fragments because the firewall thinks it's an attack. This configuration probably didn't cause problems in the past, since traditional DNS packets are usually pretty small in size. However, with DNSSEC, these configurations need to be updated, since DNSSEC traffic regularly exceeds 1500 bytes (a common MTU value). If the configuration is not updated to support a larger DNS packet size, it often results in the larger packets being rejected, and to the end user it looks like the queries go unanswered. Or in the case of fragmentation, only a part of the answer makes it to the validating resolver, and your validating resolver may need to re-ask the question again and again, creating the appearance for end users that the DNS/network is slow.

While you are updating the configuration on your network equipment, make sure TCP port 53 is also allowed for DNS traffic.

Wait... DNS Uses TCP?

Yes. DNS uses TCP port 53 as a fallback mechanism, when it cannot use UDP to transmit data. This has always been the case, even long before the arrival of DNSSEC. Traditional DNS relies on TCP port 53 for operations such as zone transfer. The use of DNSSEC, or DNS with IPv6 records such as AAAA, increases the chance that DNS data will be transmitted via TCP.

Due to the increased packet size, DNSSEC may fall back to TCP more often than traditional (insecure) DNS. If your network blocks or filters TCP port 53 today, you may already experience instability with DNS resolution, before even deploying DNSSEC.

9.5 Signing

9.5.1 Easy-Start Guide for Signing Authoritative Zones

This section provides the basic information needed to set up a DNSSEC-enabled authoritative name server. A DNSSEC-enabled (or "signed") zone contains additional resource records that are used to verify the authenticity of its zone information.

To convert a traditional (insecure) DNS zone to a secure one, we need to create some additional records (DNSKEY, RRSIG, and NSEC or NSEC3), and upload verifiable information (such as a DS record) to the parent zone to complete the chain of trust. For more information about DNSSEC resource records, please see *What Does DNSSEC Add to DNS?*.

Note: In this chapter, we assume all configuration files, key files, and zone files are stored in `/etc/bind`, and most examples show commands run as the root user. This may not be ideal, but the point is not to distract from what is important here: learning how to sign a zone. There are many best practices for deploying a more secure BIND installation, with techniques such as jailed process and restricted user privileges, but those are not covered in this document. We trust you, a responsible DNS administrator, to take the necessary precautions to secure your system.

For the examples below, we work with the assumption that there is an existing insecure zone `example.com` that we are converting to a secure zone.

Enabling Automated DNSSEC Zone Maintenance and Key Generation

To sign a zone, add the following statement to its `zone` clause in the BIND 9 configuration file:

```
options {
    directory "/etc/bind";
    recursion no;
    ...
};

zone "example.com" in {
    ...
    dnssec-policy default;
    ...
};
```

The `dnssec-policy` statement causes the zone to be signed and turns on automatic maintenance for the zone. This includes re-signing the zone as signatures expire and replacing keys on a periodic basis. The value `default` selects the default policy, which contains values suitable for most situations. We cover the creation of a custom policy in *Creating a Custom DNSSEC Policy*, but for the moment we are accepting the default values.

When the configuration file is updated, tell `named` to reload the configuration file by running `rndc reconfig`:

```
# rndc reconfig
```

And that's it - BIND signs your zone.

At this point, before you go away and merrily add `dnssec-policy` statements to all your zones, we should mention that, like a number of other BIND configuration options, its scope depends on where it is placed. In the example above, we placed it in a `zone` clause, so it applied only to the zone in question. If we had placed it in a `view` clause, it would have applied to all zones in the view; and if we had placed it in the `options` clause, it would have applied to all zones served by this instance of BIND.

Verification

The BIND 9 reconfiguration starts the process of signing the zone. First, it generates a key for the zone and includes it in the published zone. The log file shows messages such as these:

```
07-Apr-2020 16:02:55.045 zone example.com/IN (signed): reconfiguring zone keys
07-Apr-2020 16:02:55.045 reloading configuration succeeded
07-Apr-2020 16:02:55.046 keymgr: DNSKEY example.com/ECDSAP256SHA256/10376 (CSK)↵
→created for policy default
07-Apr-2020 16:02:55.046 Fetching example.com/ECDSAP256SHA256/10376 (CSK) from key↵
→repository.
07-Apr-2020 16:02:55.046 DNSKEY example.com/ECDSAP256SHA256/10376 (CSK) is now↵
→published
07-Apr-2020 16:02:55.046 DNSKEY example.com/ECDSAP256SHA256/10376 (CSK) is now active
07-Apr-2020 16:02:55.048 zone example.com/IN (signed): next key event: 07-Apr-2020↵
→18:07:55.045
```

It then starts signing the zone. How long this process takes depends on the size of the zone, the speed of the server, and how much activity is taking place. We can check what is happening by using `rndc`, entering the command:

```
# rndc signing -list example.com
```

While the signing is in progress, the output is something like:

```
Signing with key 10376/ECDSAP256SHA256
```

and when it is finished:

```
Done signing with key 10376/ECDSAP256SHA256
```

When the second message appears, the zone is signed.

Before moving on to the next step of coordinating with the parent zone, let's make sure everything looks good using `delv`. We want to simulate what a validating resolver will check, by telling `delv` to use a specific trust anchor.

First, we need to make a copy of the key created by BIND. This is in the directory you set with the `directory` statement in your configuration file's `options` clause, and is named something like `Kexample.com.+013.10376.key`:

```
# cp /etc/bind/Kexample.com.+013+10376.key /tmp/example.key
```

The original key file looks like this (with the actual key shortened for ease of display, and comments omitted):

```
# cat /etc/bind/Kexample.com.+013+10376.key

...
example.com. 3600 IN DNSKEY 257 3 13 6saiq99qDB...dqp+o0dw==
```

We want to edit the copy to be in the `trust-anchors` format, so that it looks like this:

```
# cat /tmp/example.key
trust-anchors {
    example.com. static-key 257 3 13 "6saiq99qDB...dqp+o0dw==";
};
```

Now we can run the `delv` command and instruct it to use this trusted-key file to validate the answer it receives from the authoritative name server 192.168.1.13:

```
$ delv @192.168.1.13 -a /tmp/example.key +root=example.com example.com. SOA +multiline
; fully validated
example.com.        600 IN SOA ns1.example.com. admin.example.com. (
                2020040703 ; serial
                1800       ; refresh (30 minutes)
                900        ; retry (15 minutes)
                2419200    ; expire (4 weeks)
                300        ; minimum (5 minutes)
                )
example.com.        600 IN RRSIG SOA 13 2 600 (
                20200421150255 20200407140255 10376 example.com.
                jBsz92zwAcGMNV/yu167aKQZvFyC7BiQe1WEnlogdLTF
                oq4yBQumOhO5WX61LjA17l1DuLWcd/ASwlUZWFGCYQ== )
```

Uploading Information to the Parent Zone

Once everything is complete on our name server, we need to generate some information to be uploaded to the parent zone to complete the chain of trust. The format and the upload methods are actually dictated by your parent zone's administrator, so contact your registrar or parent zone administrator to find out what the actual format should be and how to deliver or upload the information to the parent zone.

What about your zone between the time you signed it and the time your parent zone accepts the upload? To the rest of the world, your zone still appears to be insecure, because if a validating resolver attempts to validate your domain name via your parent zone, your parent zone will indicate that you are not yet signed (as far as it knows). The validating resolver will then give up attempting to validate your domain name, and will fall back to the insecure DNS. Until you complete this final step with your parent zone, your zone remains insecure.

Note: Before uploading to your parent zone, verify that your newly signed zone has propagated to all of your name servers (usually via zone transfers). If some of your name servers still have unsigned zone data while the parent tells the world it should be signed, validating resolvers around the world cannot resolve your domain name.

Here are some examples of what you may upload to your parent zone, with the DNSKEY/DS data shortened for display. Note that no matter what format may be required, the end result is the parent zone publishing DS record(s) based on the information you upload. Again, contact your parent zone administrator(s) to find out the correct format for their system.

1. DS record format:

```
example.com. 3600 IN DS 10376 13 2 B92E22CAE0...33B8312EF0
```

2. DNSKEY format:

```
example.com. 3600 IN DNSKEY 257 3 13 6saiq99qDB...dqp+o0dw==
```

The DS record format may be generated from the DNSKEY using the `dnssec-dsfromkey` tool, which is covered in *DS Record Format*. For more details and examples on how to work with your parent zone, please see *Working With the Parent Zone*.

So... What Now?

Congratulations! Your zone is signed, your secondary servers have received the new zone data, and the parent zone has accepted your upload and published your DS record. Your zone is now officially DNSSEC-enabled. What happens next? That is basically it - BIND takes care of everything else. As for updating your zone file, you can continue to update it the same way as prior to signing your zone; the normal work flow of editing a zone file and using the `rndc` command to reload the zone still works as usual, and although you are editing the unsigned version of the zone, BIND generates the signed version automatically.

Curious as to what all these commands did to your zone file? Read on to *Your Zone, Before and After DNSSEC* and find out. If you are interested in how to roll this out to your existing primary and secondary name servers, check out *DNSSEC Signing* in the *Recipes* chapter.

9.5.2 Your Zone, Before and After DNSSEC

When we assigned the default DNSSEC policy to the zone, we provided the minimal amount of information to convert a traditional DNS zone into a DNSSEC-enabled zone. This is what the zone looked like before we started:

```
$ dig @192.168.1.13 example.com. AXFR +multiline +onesoa

; <<>> DiG 9.16.0 <<>> @192.168.1.13 example.com AXFR +multiline +onesoa
; (1 server found)
;; global options: +cmd
example.com.          600 IN SOA ns1.example.com. admin.example.com. (
                 2020040700 ; serial
                 1800       ; refresh (30 minutes)
                 900        ; retry (15 minutes)
                 2419200    ; expire (4 weeks)
                 300        ; minimum (5 minutes)
                 )
example.com.          600 IN NS ns1.example.com.
ftp.example.com.      600 IN A 192.168.1.200
ns1.example.com.      600 IN A 192.168.1.1
web.example.com.      600 IN CNAME www.example.com.
www.example.com.      600 IN A 192.168.1.100
```

Below shows the test zone `example.com` after reloading the server configuration. Clearly, the zone grew in size, and the number of records multiplied:

```
# dig @192.168.1.13 example.com. AXFR +multiline +onesoa

; <<>> DiG 9.16.0 <<>> @192.168.1.13 example.com AXFR +multiline +onesoa
; (1 server found)
;; global options: +cmd
example.com.          600 IN SOA ns1.example.com. admin.example.com. (
                 2020040703 ; serial
                 1800       ; refresh (30 minutes)
                 900        ; retry (15 minutes)
                 2419200    ; expire (4 weeks)
                 300        ; minimum (5 minutes)
                 )
example.com.          300 IN RRSIG NSEC 13 2 300 (
                 20200413050536 20200407140255 10376 example.com.
                 drtV1rJbo5OMi65OJtu7Jmg/thgpdTWrzr6O3Pzt12+B
                 oCxMAv3orWWYjfP2n9w5wj0rx2Mt2ev7MOOG8IOUCA== )
example.com.          300 IN NSEC ftp.example.com. NS SOA RRSIG NSEC DNSKEY TYPE65534
```

(continues on next page)

```
example.com.          600 IN RRSIG NS 13 2 600 (
                20200413130638 20200407140255 10376 example.com.
                2ipmzm1Ei6vfE9OLowPMsxLBCbjrCpWPgWJ0ekwZBbux
                MLffZOXn8clt0Ql2U9iCPdyoQryuJCiojHSE2d6nrw== )
example.com.          600 IN RRSIG SOA 13 2 600 (
                20200421150255 20200407140255 10376 example.com.
                jBsz92zwAcGMNV/yu167aKQZvFyC7BiQe1WEnlogdLTF
                oq4yBQumOhO5WX61LjA17l1DuLWcd/ASwlUZWFGCYQ== )
example.com.          0 IN RRSIG TYPE65534 13 2 0 (
                20200413050536 20200407140255 10376 example.com.
                Xjkom24N6qeCJjg9BMUfuWf+euLeZB169DHvLYZPZNlm
                GgM2czUDPio6VpQbUw6JE5DSNjuGjgpgXC5SipC42g== )
example.com.          3600 IN RRSIG DNSKEY 13 2 3600 (
                20200421150255 20200407140255 10376 example.com.
                maK75+28oUyDtci3V7wjTsuhgkLUZW+Q++q46Lea6bKn
                Xj77kXcLNogNdUOr5am/6O6cnPeJKJWsnmTLISm62g== )
example.com.          0 IN TYPE65534 \# 5 ( 0D28880001 )
example.com.          3600 IN DNSKEY 257 3 13 (
                6saiq99qDBb5b4G4cx13cPjFTrIvUs3NW44SvbbHorHb
                kXwOzeGAWyPORN+pwEV/LP9+FHAF/JzAJYdqp+o0dw==
                ) ; KSK; alg = ECDSAP256SHA256 ; key id = 10376
example.com.          600 IN NS ns1.example.com.
ftp.example.com.     600 IN RRSIG A 13 3 600 (
                20200413130638 20200407140255 10376 example.com.
                UYo1njeUA49VhKnPSS3JO4G+/Xd2PD4m3Vaacnd191yz
                BIoouEBAGPcrEM2BNrgR0op1EWSus9tG86SM1ZHGuQ== )
ftp.example.com.     300 IN RRSIG NSEC 13 3 300 (
                20200413130638 20200407140255 10376 example.com.
                rPADrAMAPIPSF3S45OSY8kXBTYMS3nrZg4Awj7qRL+/b
                sOKy6044MbIbjg+YWL69dBjKoTSeEGSCSt73uIxrYA== )
ftp.example.com.     300 IN NSEC ns1.example.com. A RRSIG NSEC
ftp.example.com.     600 IN A 192.168.1.200
ns1.example.com.     600 IN RRSIG A 13 3 600 (
                20200413130638 20200407140255 10376 example.com.
                Yeojg7qrJmxL6uLTnALwKU5byNldZ9Ggj5XjcbpPvujQ
                ocG/ovGBg6pdugXC9UxE39bCDl8dua1frjDcRCCZAA== )
ns1.example.com.     300 IN RRSIG NSEC 13 3 300 (
                20200413130638 20200407140255 10376 example.com.
                vukgQme6k7JwCf/mJOOzHXbE3fKtSro+Kc10T6dHMdsc
                oM1/oXioZvgBZ9cKrQhIAUt7r1KUnrUwM6Je36wWFA== )
ns1.example.com.     300 IN NSEC web.example.com. A RRSIG NSEC
ns1.example.com.     600 IN A 192.168.1.1
web.example.com.     600 IN RRSIG CNAME 13 3 600 (
                20200413130638 20200407140255 10376 example.com.
                JXi4WYypofD5geUowVqlqJyHzvcRnsvU/ONhTBaUCw5Y
                XtifKAXRHWrUL1HIwt37JYPLf5uYu90RfkWLj0GqTQ== )
web.example.com.     300 IN RRSIG NSEC 13 3 300 (
                20200413130638 20200407140255 10376 example.com.
                XF4Hsd58dalL+s6Qu99bG80PQyMf7ZrHEzDiEflRuykP
                DfBRuf34z27vj70LO1lp2ZiX4BB1ahcEK2ae9ASAmA== )
web.example.com.     300 IN NSEC www.example.com. CNAME RRSIG NSEC
web.example.com.     600 IN CNAME www.example.com.
www.example.com.     600 IN RRSIG A 13 3 600 (
                20200413050536 20200407140255 10376 example.com.
                mACKXrDOF5JMWqncSiQ3pYWA6abyGDJ4wgGCumjLXhPy
                0cMzJmKv2s7G6+tW3TsA6BK3UoMfv30oblY2Mnl4/A== )
www.example.com.     300 IN RRSIG NSEC 13 3 300 (
```

```
                20200413050536 20200407140255 10376 example.com.
                1YQ22odVt0TeP5gbNJwkvS684ipDmx6sEOsF0eCizhCv
                x8osuOATdlPjIEztt+rveaErZ2nsoLor5k1nQAHsbQ== )
www.example.com.    300 IN NSEC example.com. A RRSIG NSEC
www.example.com.    600 IN A 192.168.1.100
```

But this is a really messy way to tell if the zone is set up properly with DNSSEC. Fortunately, there are tools to help us with that. Read on to *How To Test Authoritative Zones* to learn more.

9.5.3 How To Test Authoritative Zones

So we've activated DNSSEC and uploaded some data to our parent zone. How do we know our zone is signed correctly? Here are a few ways to check.

Look for Key Data in Your Zone

One way to see if your zone is signed is to check for the presence of DNSKEY record types. In our example, we created a single key, and we expect to see it returned when we query for it.

```
$ dig @192.168.1.13 example.com. DNSKEY +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.6 example.com DNSKEY +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18637
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: efe186423313fb66010000005e8c997e99864f7d69ed7c11 (good)
;; QUESTION SECTION:
;example.com.        IN DNSKEY

;; ANSWER SECTION:
example.com.         3600 IN DNSKEY 257 3 13 (
                6saiq99qDBb5b4G4cx13cPjFTrIvUs3NW44SvbbHorHb
                kXwOzeGAWyPORN+pwEV/LP9+FHAF/JzAJYdqp+o0dw==
                ) ; KSK; alg = ECDSAP256SHA256 ; key id = 10376
```

Look for Signatures in Your Zone

Another way to see if your zone data is signed is to check for the presence of a signature. With DNSSEC, every record[6] now comes with at least one corresponding signature, known as an RRSIG.

```
$ dig @192.168.1.13 example.com. SOA +dnssec +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.6 example.com SOA +dnssec +multiline
```

---

[6] Well, almost every record: NS records and glue records for delegations do not have RRSIG records. If there are no delegations, then every record in your zone is signed and comes with its own RRSIG.

```
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45219
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 75adff4f4ce916b2010000005e8c99c0de47eabb7951b2f5 (good)
;; QUESTION SECTION:
;example.com.        IN SOA

;; ANSWER SECTION:
example.com.          600 IN SOA ns1.example.com. admin.example.com. (
                 2020040703 ; serial
                 1800       ; refresh (30 minutes)
                 900        ; retry (15 minutes)
                 2419200    ; expire (4 weeks)
                 300        ; minimum (5 minutes)
                 )
example.com.          600 IN RRSIG SOA 13 2 600 (
                 20200421150255 20200407140255 10376 example.com.
                 jBsz92zwAcGMNV/yu167aKQZvFyC7BiQe1WEnlogdLTF
                 oq4yBQumOhO5WX61LjA17l1DuLWcd/ASwlUZWFGCYQ== )
```

The serial number was automatically incremented from the old, unsigned version. `named` keeps track of the serial number of the signed version of the zone independently of the unsigned version. If the unsigned zone is updated with a new serial number that is higher than the one in the signed copy, then the signed copy is increased to match it; otherwise, the two are kept separate.

Examine the Zone File

Our original zone file `example.com.db` remains untouched, and `named` has generated three additional files automatically for us (shown below). The signed DNS data is stored in `example.com.db.signed` and in the associated journal file.

```
# cd /etc/bind
# ls
example.com.db  example.com.db.jbk  example.com.db.signed  example.com.db.signed.jnl
```

A quick description of each of the files:

- `.jbk`: a transient file used by `named`

- `.signed`: the signed version of the zone in raw format

- `.signed.jnl`: a journal file for the signed version of the zone

These files are stored in raw (binary) format for faster loading. To reveal the human-readable version, use `named-compilezone` as shown below. In the example below, we run the command on the raw format zone `example.com.db.signed` to produce a text version of the zone `example.com.text`:

```
# named-compilezone -f raw -F text -o example.com.text example.com example.com.db.
↪signed
zone example.com/IN: loaded serial 2014112008 (DNSSEC signed)
```

```
dump zone to example.com.text...done
OK
```

Check the Parent

Although this is not strictly related to whether the zone is signed, a critical part of DNSSEC is the trust relationship between the parent and the child. Just because we, the child, have all the correctly signed records in our zone does not mean it can be fully validated by a validating resolver, unless our parent's data agrees with ours. To check if our upload to the parent was successful, ask the parent name server for the DS record of our child zone; we should get back the DS record(s) containing the information we uploaded in *Uploading Information to the Parent Zone*:

```
$ dig example.com. DS

; <<>> DiG 9.16.0 <<>> example.com DS
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16954
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: db280d5b52576780010000005e8c9bf5b0d8de103d934e5d (good)
;; QUESTION SECTION:
;example.com.            IN   DS

;; ANSWER SECTION:
example.com.  61179 IN   DS   10376 13 2↵
→B92E22CAE0B41430EC38D3F7EDF1183C3A94F4D4748569250C15EE33B8312EF0
```

External Testing Tools

We recommend two tools, below: Verisign DNSSEC Debugger and DNSViz. Others can be found via a simple online search. These excellent online tools are an easy way to verify that your domain name is fully secured.

Verisign DNSSEC Debugger

URL: https://dnssec-debugger.verisignlabs.com/

This tool shows a nice summary of checks performed on your domain name. You can expand it to view more details for each of the items checked, to get a detailed report.

Fig. 3: Verisign DNSSEC Debugger

DNSViz

URL: https://dnsviz.net/

DNSViz provides a visual analysis of the DNSSEC authentication chain for a domain name and its resolution path in the DNS namespace.

9.5.4 Signing Easy Start Explained

Enable Automatic DNSSEC Maintenance Explained

Signing a zone requires a number of separate steps:

- Generation of the keys to sign the zone.
- Inclusion of the keys into the zone.
- Signing of the records in the file (including the generation of the NSEC or NSEC3 records).

Maintaining a signed zone comprises a set of ongoing tasks:

- Re-signing the zone as signatures approach expiration.
- Generation of new keys as the time approaches for a key roll.
- Inclusion of new keys into the zone when the rollover starts.
- Transition from signing the zone with the old set of keys to signing the zone with the new set of keys.
- Waiting the appropriate interval before removing the old keys from the zone.
- Deleting the old keys.

That is quite complex, and it is all handled in BIND 9 with the single `dnssec-policy default` statement. We will see later on (in the *Creating a Custom DNSSEC Policy* section) how these actions can be tuned, by setting up our own DNSSEC policy with customized parameters. However, in many cases the defaults are adequate.

At the time of this writing (mid-2020), `dnssec-policy` is still a relatively new feature in BIND. Although it is the preferred way to run DNSSEC in a zone, it is not yet able to automatically implement all the features that are available with a more "hands-on" approach to signing and key maintenance. For this reason, we cover alternative signing techniques in *Alternate Ways of Signing a Zone*.

9.5.5 Working With the Parent Zone

As mentioned in *Uploading Information to the Parent Zone*, the format of the information uploaded to your parent zone is dictated by your parent zone administrator. The two main formats are:

1. DS record format
2. DNSKEY format

Check with your parent zone to see which format they require.

But how can you get each of the formats from your existing data?

When `named` turned on automatic DNSSEC maintenance, essentially the first thing it did was to create the DNSSEC keys and put them in the directory you specified in the configuration file. If you look in that directory, you will see three files with names like `Kexample.com.+013+10376.key`, `Kexample.com.+013+10376.private`, and `Kexample.com.+013+10376.state`. The one we are interested in is the one with the `.key` suffix, which contains the zone's public key. (The other files contain the zone's private key and the DNSSEC state associated with the key.) This public key is used to generate the information we need to pass to the parent.
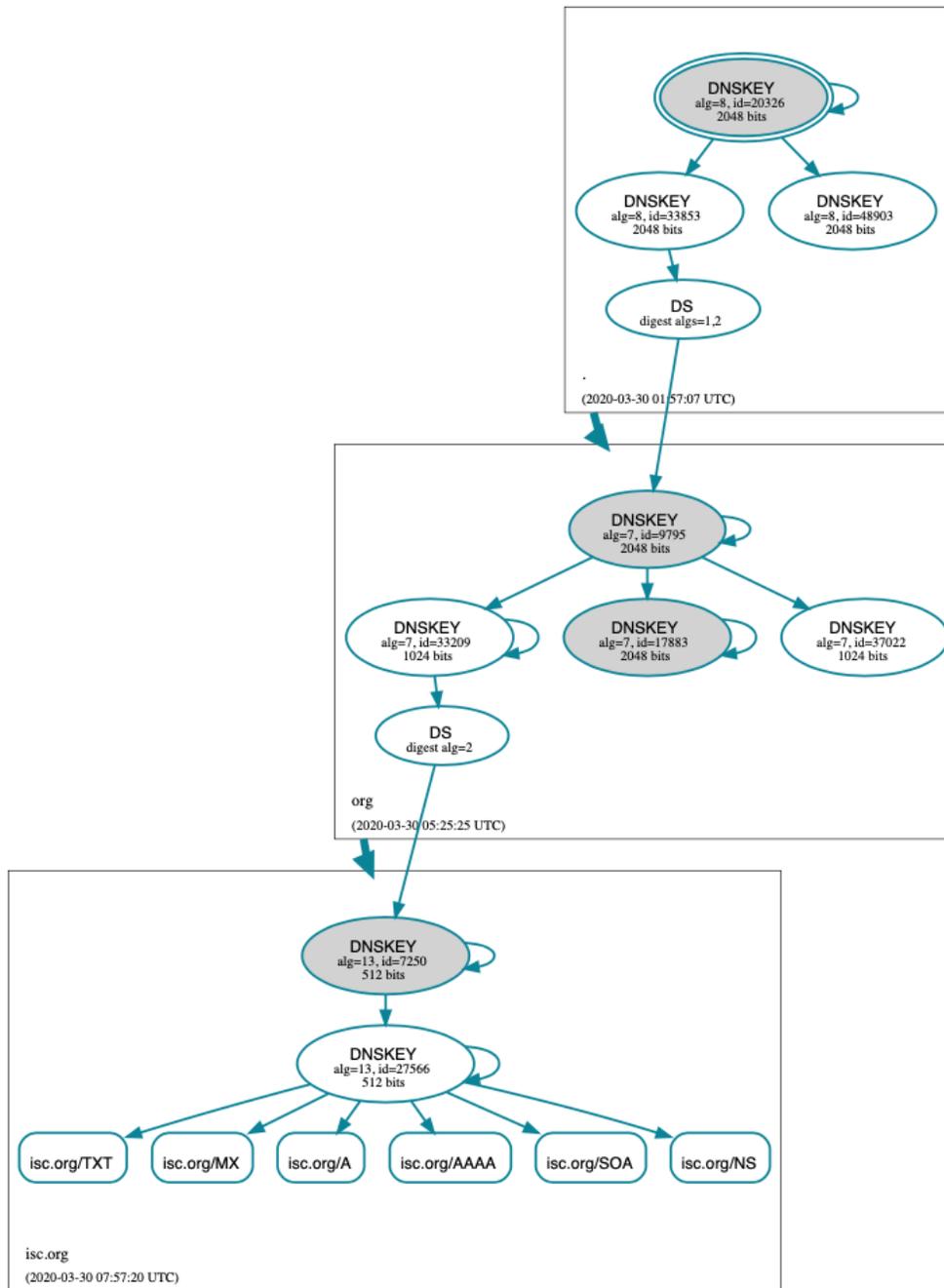
Fig. 4: DNSViz

DS Record Format

Below is an example of a DS record format generated from the KSK we created earlier (`Kexample.com.+013+10376.key`):

```
# cd /etc/bind
 dnssec-dsfromkey Kexample.com.+013+10376.key
example.com. IN DS 10376 13 2␣
↪B92E22CAE0B41430EC38D3F7EDF1183C3A94F4D4748569250C15EE33B8312EF0
```

Some registrars ask their customers to manually specify the types of algorithm and digest used. In this example, 13 represents the algorithm used, and 2 represents the digest type (SHA-256). The key tag or key ID is 10376.

DNSKEY Format

Below is an example of the same key ID (10376) using DNSKEY format (with the actual key shortened for ease of display):

```
example.com. 3600 IN DNSKEY 257 3 13 (6saiq99qDB...dqp+o0dw==) ; key id = 10376
```

The key itself is easy to find (it's difficult to miss that long base64 string) in the file.

```
# cd /etc/bind
# cat Kexample.com.+013+10376.key
; This is a key-signing key, keyid 10376, for example.com.
; Created: 20200407150255 (Tue Apr  7 16:02:55 2020)
; Publish: 20200407150255 (Tue Apr  7 16:02:55 2020)
; Activate: 20200407150255 (Tue Apr  7 16:02:55 2020)
example.com. 3600 IN DNSKEY 257 3 13 6saiq99qDB...dqp+o0dw==
```

9.5.6 Creating a Custom DNSSEC Policy

The remainder of this section describes the contents of a custom DNSSEC policy. *Advanced Discussions* describes the concepts involved here and the pros and cons of choosing particular values. If you are not already familiar with DNSSEC, it may be worth reading that chapter first.

Setting up your own DNSSEC policy means that you must include a `dnssec-policy` clause in the zone file. This sets values for the various parameters that affect the signing of zones and the rolling of keys. The following is an example of such a clause:

```
dnssec-policy standard {
    dnskey-ttl 600;
    keys {
        ksk lifetime 365d algorithm ecdsap256sha256;
        zsk lifetime 60d algorithm ecdsap256sha256;
    };
    max-zone-ttl 600;
    parent-ds-ttl 600;
    parent-propagation-delay 2h;
    publish-safety 7d;
    retire-safety 7d;
    signatures-refresh 5d;
    signatures-validity 15d;
    signatures-validity-dnskey 15d;
```

(continues on next page)

```
    zone-propagation-delay 2h;
};
```

The policy has multiple parts:

- The name must be specified. As each zone can use a different policy, `named` needs to be able to distinguish between policies. This is done by giving each policy a name, such as `standard` in the above example.

- The `keys` clause lists all keys that should be in the zone, along with their associated parameters. In this example, we are using the conventional KSK/ZSK split, with the KSK changed every year and the ZSK changed every two months (the `default` DNSSEC policy sets a CSK that is never changed). Keys are created using the ECDSAPS256SHA256 algorithm; each KSK/ZSK pair must have the same algorithm. A CSK combines the functionality of a ZSK and a KSK.

- The parameters ending in `-ttl` are, as expected, the TTLs of the associated records. Remember that during a key rollover, we have to wait for records to expire from caches? The values here tell BIND 9 the maximum amount of time it has to wait for this to happen. Values can be set for the DNSKEY records in your zone, the non-DNSKEY records in your zone, and the DS records in the parent zone.

- Another set of time-related parameters are those ending in `-propagation-delay`. These tell BIND how long it takes for a change in zone contents to become available on all secondary servers. (This may be non-negligible: for example, if a large zone is transferred over a slow link.)

- The policy also sets values for the various signature parameters: how long the signatures on the DNSKEY and non-DNSKEY records are valid, and how often BIND should re-sign the zone.

- The parameters ending in `-safety` are there to give you a bit of leeway in case a key roll doesn't go to plan. When introduced into the zone, the `publish-safety` time is the amount of additional time, over and above that calculated from the other parameters, during which the new key is in the zone but before BIND starts to sign records with it. Similarly, the `retire-safety` is the amount of additional time, over and above that calculated from the other parameters, during which the old key is retained in the zone before being removed.

- Finally, the `purge-keys` option allows you to clean up key files automatically after a period of time. If a key has been removed from the zone, this option will determine how long its key files will be retained on disk.

(You do not have to specify all the items listed above in your policy definition. Any that are not set simply take the default value.)

Usually, the exact timing of a key roll, or how long a signature remains valid, is not critical. For this reason, err on the side of caution when setting values for the parameters. It is better to have an operation like a key roll take a few days longer than absolutely required, than it is to have a quick key roll but have users get validation failures during the process.

Having defined a new policy called "standard", we now need to tell `named` to use it. We do this by adding a `dnssec-policy standard;` statement to the configuration file. Like many other configuration statements, it can be placed in the `options` statement (thus applying to all zones on the server), a `view` statement (applying to all zones in the view), or a `zone` statement (applying only to that zone). In this example, we'll add it to the `zone` statement:

```
zone "example.net" in {
    ...
    dnssec-policy standard;
    ...
};
```

Finally, tell `named` to use the new policy:

```
# rndc reconfig
```

… and that's it. `named` now applies the "standard" policy to your zone.

9.5.7 Maintenance Tasks

Zone data is signed and the parent zone has published your DS records: at this point your zone is officially secure. When other validating resolvers look up information in your zone, they are able to follow the 12-step process as described in *How Does DNSSEC Change DNS Lookup (Revisited)?* and verify the authenticity and integrity of the answers.

There is not that much left for you, as the DNS administrator, to do on an ongoing basis. Whenever you update your zone, BIND automatically re-signs your zone with new RRSIG and NSEC/NSEC3 records, and even increments the serial number for you. If you choose to split your keys into a KSK and ZSK, the rolling of the ZSK is completely automatic. Rolling of a KSK or CSK may require some manual intervention, though, so let's examine two more DNSSEC-related resource records, CDS and CDNSKEY.

The CDS and CDNSKEY Resource Records

Passing the DS record to the organization running the parent zone has always been recognized as a bottleneck in the key rollover process. To automate the process, the CDS and CDNSKEY resource records were introduced.

The CDS and CDNSKEY records are identical to the DS and DNSKEY records, except in the type code and the name. When such a record appears in the child zone, it is a signal to the parent that it should update the DS it has for that zone. In essence, when the parent notices the presence of the CDS and/or CDNSKEY record(s) in the child zone, it checks these records to verify that they are signed by a valid key for the zone. If the record(s) successfully validate, the parent zone's DS RRset for the child zone is changed to correspond to the CDS (or CDNSKEY) records. (For more information on how the signaling works and the issues surrounding it, please refer to **RFC 7344** and **RFC 8078**.)

Working with the Parent Zone (2)

Once the zone is signed, the only required manual tasks are to monitor KSK or CSK key rolls and pass the new DS record to the parent zone. However, if the parent can process CDS or CDNSKEY records, you may not even have to do that[7].

When the time approaches for the roll of a KSK or CSK, BIND adds a CDS and a CDNSKEY record for the key in question to the apex of the zone. If your parent zone supports polling for CDS/CDNSKEY records, they are uploaded and the DS record published in the parent - at least ideally.

If BIND is configured with `parental-agents`, it will check for the DS presence. Let's look at the following configuration excerpt:

```
parental-agents "net" {
    10.53.0.11; 10.53.0.12;
};

zone "example.net" in {
    ...
    dnssec-policy standard;
    parental-agents { "net"; };
    ...
};
```

BIND will check for the presence of the DS record in the parent zone by querying its parental agents (defined in **RFC 7344** to be the entities that the child zone has a relationship with to change its delegation information). In the example above, The zone *example.net* is configured with two parental agents, at the addresses 10.53.0.11 and 10.53.0.12. These addresses are used as an example only. Both addresses will have to respond with a DS RRset that includes the DS record

---

[7] For security reasons, a parent zone that supports CDS/CDNSKEY may require the DS record to be manually uploaded when we first sign the zone. Until our zone is signed, the parent cannot be sure that a CDS or CDNSKEY record it finds by querying our zone really comes from our zone; thus, it needs to use some other form of secure transfer to obtain the information.

identifying the key that is being rolled. If one or both don't have the DS included yet the rollover is paused, and the check for DS presence is retried after an hour. The same applies for DS withdrawal.

Alternatively, you can use the `rndc` tool to tell `named` that the DS record has been published or withdrawn. For example:

```
# rndc dnssec -checkds published example.net
```

If your parent zone doesn't support CDS/CDNSKEY, you will have to supply the DNSKEY or DS record to the parent zone manually when a new KSK appears in your zone, presumably using the same mechanism you used to upload the records for the first time. Again, you need to use the `rndc` tool to tell `named` that the DS record has been published.

9.5.8 Alternate Ways of Signing a Zone

Although use of the automatic `dnssec-policy` is the preferred way to sign zones in BIND, there are occasions where a more manual approach may be needed, such as when external hardware is used to generate and sign the zone. `dnssec-policy` does not currently support the use of external hardware, so if your security policy requires it, you need to use one of the methods described here.

The idea of DNSSEC was first discussed in the 1990s and has been extensively developed over the intervening years. BIND has tracked the development of this technology, often being the first name server implementation to introduce new features. However, for compatibility reasons, BIND retained older ways of doing things even when new ways were added. This particularly applies to signing and maintaining zones, where different levels of automation are available.

The following is a list of the available methods of signing in BIND, in the order that they were introduced - and in order of decreasing complexity.

Manual "Manual" signing was the first method to be introduced into BIND and its name describes it perfectly: the user needs to do everything. In the more-automated methods, you load an unsigned zone file into `named`, which takes care of signing it. With manual signing, you have to provide a signed zone for `named` to serve.

In practice, this means creating an unsigned zone file as usual, then using the BIND-provided tools `dnssec-keygen` to create the keys and `dnssec-signzone` to sign the zone. The signed zone is stored in another file and is the one you tell BIND to load. To update the zone (for example, to add a resource record), you update the unsigned zone, re-sign it, and tell `named` to load the updated signed copy. The same goes for refreshing signatures or rolling keys; the user is responsible for providing the signed zone served by `named`. (In the case of rolling keys, you are also responsible for ensuring that the keys are added and removed at the correct times.)

Why would you want to sign your zone this way? You probably wouldn't in the normal course of events, but as there may be circumstances in which it is required, the scripts have been left in the BIND distribution.

Semi-Automatic The first step in DNSSEC automation came with BIND 9.7, when the `auto-dnssec` option was added. This causes `named` to periodically search the directory holding the key files (see *Generate Keys* for a description) and to use the information in them to both add and remove keys and sign the zone.

Use of `auto-dnssec` alone requires that the zone be dynamic, something not suitable for a number of situations, so BIND 9.9 added the `inline-signing` option. With this, `named` essentially keeps the signed and unsigned copies of the zone separate. The signed zone is created from the unsigned one using the key information; when the unsigned zone is updated and the zone reloaded, `named` detects the changes and updates the signed copy of the zone.

This mode of signing has been termed "semi-automatic" in this document because keys still have to be manually created (and deleted when appropriate). Although not an onerous task, it is still additional work.

Why would anyone want to use this method when fully automated ones are available? At the time of this writing (mid-2020), the fully automatic methods cannot handle all scenarios, particularly that of having a single key shared among multiple zones. They also do not handle keys stored in Hardware Security Modules (HSMs), which are briefly covered in *Hardware Security Modules (HSMs)*.

Fully Automatic with `dnssec-keymgr` The next step in the automation of DNSSEC operations came with BIND 9.11, which introduced the `dnssec-keymgr` utility. This is a separate program and is expected to be run on a regular basis (probably via `cron`). It reads a DNSSEC policy from its configuration file and reads timing information from the DNSSEC key files. With this information it creates new key files with timing information in them consistent with the policy. `named` is run as usual, picking up the timing information in the key files to determine when to add and remove keys, and when to sign with them.

In BIND 9.17.0 and later, this method of handling DNSSEC policies has been replaced by the `dnssec-policy` statement in the configuration file.

Fully Automatic with `dnssec-policy` Introduced a BIND 9.16, `dnssec-policy` replaces `dnssec-keymgr` from BIND 9.17 onwards and avoids the need to run a separate program. It also handles the creation of keys if a zone is added (`dnssec-keymgr` requires an initial key) and deletes old key files as they are removed from the zone. This is the method described in *Easy-Start Guide for Signing Authoritative Zones*.

We now look at some of these methods in more detail. We cover semi-automatic signing first, as that contains a lot of useful information about keys and key timings. We then describe what `dnssec-keymgr` adds to semi-automatic signing. After that, we touch on fully automatic signing with `dnssec-policy`. Since this has already been described in *Easy-Start Guide for Signing Authoritative Zones*, we will just mention a few additional points. Finally, we briefly describe manual signing.

Semi-Automatic Signing

As noted above, the term semi-automatic signing has been used in this document to indicate the mode of signing enabled by the `auto-dnssec` and `inline-signing` keywords. `named` signs the zone without any manual intervention, based purely on the timing information in the DNSSEC key files. The files, however, must be created manually.

By appropriately setting the key parameters and the timing information in the key files, you can implement any DNSSEC policy you want for your zones. But why manipulate the key information yourself rather than rely on `dnssec-keymgr` or `dnssec-policy` to do it for you? The answer is that semi-automatic signing allows you to do things that, at the time of this writing (mid-2020), are currently not possible with one of the key managers: for example, the ability to use an HSM to store keys, or the ability to use the same key for multiple zones.

To convert a traditional (insecure) DNS zone to a secure one, we need to create various additional records (DNSKEY, RRSIG, NSEC/NSEC3) and, as with fully automatic signing, to upload verifiable information (such as a DS record) to the parent zone to complete the chain of trust.

Note: Again, we assume all configuration files, key files, and zone files are stored in `/etc/bind`, and most examples show commands run as the root user. This may not be ideal, but the point is not to distract from what is important here: learning how to sign a zone. There are many best practices for deploying a more secure BIND installation, with techniques such as jailed process and restricted user privileges, but those are not covered in this document. We trust you, a responsible DNS administrator, to take the necessary precautions to secure your system.

For our examples below, we work with the assumption that there is an existing insecure zone `example.com` that we are converting to a secure version. The secure version uses both a KSK and a ZSK.

Generate Keys

Everything in DNSSEC centers around keys, so we begin by generating our own keys.

```
# cd /etc/bind
# dnssec-keygen -a RSASHA256 -b 1024 example.com
Generating key pair............................+++++ ......................+++++
Kexample.com.+008+34371
# dnssec-keygen -a RSASHA256 -b 2048 -f KSK example.com
Generating key pair........................+++ ..................................+++
Kexample.com.+008+00472
```

This command generates four key files in `/etc/bind/keys`:

- Kexample.com.+008+34371.key

- Kexample.com.+008+34371.private

- Kexample.com.+008+00472.key

- Kexample.com.+008+00472.private

The two files ending in `.key` are the public keys. These contain the DNSKEY resource records that appear in the zone. The two files ending in `.private` are the private keys, and contain the information that `named` actually uses to sign the zone.

Of the two pairs, one is the zone-signing key (ZSK), and one is the key-signing key (KSK). We can tell which is which by looking at the file contents (the actual keys are shortened here for ease of display):

```
# cat Kexample.com.+008+34371.key
; This is a zone-signing key, keyid 34371, for example.com.
; Created: 20200616104249 (Tue Jun 16 11:42:49 2020)
; Publish: 20200616104249 (Tue Jun 16 11:42:49 2020)
; Activate: 20200616104249 (Tue Jun 16 11:42:49 2020)
example.com. IN DNSKEY 256 3 8 AwEAAfel66...LqkA7cvn8=
# cat Kexample.com.+008+00472.key
; This is a key-signing key, keyid 472, for example.com.
; Created: 20200616104254 (Tue Jun 16 11:42:54 2020)
; Publish: 20200616104254 (Tue Jun 16 11:42:54 2020)
; Activate: 20200616104254 (Tue Jun 16 11:42:54 2020)
example.com. IN DNSKEY 257 3 8 AwEAAbCR6U...l8xPjokVU=
```

The first line of each file tells us what type of key it is. Also, by looking at the actual DNSKEY record, we can tell them apart: 256 is ZSK, and 257 is KSK.

The name of the file also tells us something about the contents. The file names are of the form:

```
K<zone-name>+<algorithm-id>+<keyid>
```

The "zone name" is self-explanatory. The "algorithm ID" is a number assigned to the algorithm used to construct the key: the number appears in the DNSKEY resource record. In our example, 8 means the algorithm RSASHA256. Finally, the "keyid" is essentially a hash of the key itself.

Make sure these files are readable by `named` and make sure that the `.private` files are not readable by anyone else.

Refer to *System Entropy* for information on how to speed up the key generation process if your random number generator has insufficient entropy.

Setting Key Timing Information

You may remember that in the above description of this method, we said that time information related to rolling keys is stored in the key files. This is placed there by `dnssec-keygen` when the file is created, and it can be modified using `dnssec-settime`. By default, only a limited amount of timing information is included in the file, as illustrated in the examples in the previous section.

All the dates are the same, and are the date and time that `dnssec-keygen` created the key. We can use `dnssec-settime` to modify the dates[8]. For example, to publish this key in the zone on 1 July 2020, use it to sign records for a year starting on 15 July 2020, and remove it from the zone at the end of July 2021, we can use the following command:

```
# dnssec-settime -P 20200701 -A 20200715 -I 20210715 -D 20210731 Kexample.com.
↪+008+34371.key
./Kexample.com.+008+34371.key
./Kexample.com.+008+34371.private
```

which would set the contents of the key file to:

```
; This is a zone-signing key, keyid 34371, for example.com.
; Created: 20200616104249 (Tue Jun 16 11:42:49 2020)
; Publish: 20200701000000 (Wed Jul  1 01:00:00 2020)
; Activate: 20200715000000 (Wed Jul 15 01:00:00 2020)
; Inactive: 20210715000000 (Thu Jul 15 01:00:00 2021)
; Delete: 20210731000000 (Sat Jul 31 01:00:00 2021)
example.com. IN DNSKEY 256 3 8 AwEAAfel66...LqkA7cvn8=
```

(The actual key is truncated here to improve readability.)

Below is a complete list of each of the metadata fields, and how each one affects the signing of your zone:

1. *Created*: This records the date on which the key was created. It is not used in calculations; it is useful simply for documentation purposes.

2. *Publish*: This sets the date on which a key is to be published to the zone. After that date, the key is included in the zone but is not used to sign it. This allows validating resolvers to get a copy of the new key in their cache before there are any resource records signed with it. By default, if not specified at creation time, this is set to the current time, meaning the key is published as soon as `named` picks it up.

3. *Activate*: This sets the date on which the key is to be activated. After that date, resource records are signed with the key. By default, if not specified during creation time, this is set to the current time, meaning the key is used to sign data as soon as `named` picks it up.

4. *Revoke:* This sets the date on which the key is to be revoked. After that date, the key is flagged as revoked, although it is still included in the zone and used to sign it. This is used to notify validating resolvers that this key is about to be removed or retired from the zone. (This state is not used in normal day-to-day operations. See **RFC 5011** to understand the circumstances where it may be used.)

5. *Inactive*: This sets the date on which the key is to become inactive. After that date, the key is still included in the zone, but it is no longer used to sign it. This sets the "expiration" or "retire" date for a key.

6. *Delete*: This sets the date on which the key is to be deleted. After that date, the key is no longer included in the zone, but it continues to exist on the file system or key repository.

This can be summarized as follows:

---

[8] The dates can also be modified using an editor, but that is likely to be more error-prone than using `dnssec-settime`.

Table 1: Key Metadata Comparison

Metadata	Included in Zone File?	Used to Sign Data?	Purpose
Created	No	No	Recording of key creation
Publish	Yes	No	Introduction of a key soon to be active
Activate	Yes	Yes	Activation date for new key
Revoke	Yes	Yes	Notification of a key soon to be retired
Inactive	Yes	No	Inactivation or retirement of a key
Delete	No	No	Deletion or removal of a key from a zone

The publication date is the date the key is introduced into the zone. Sometime later it is activated and is used to sign resource records. After a specified period, BIND stops using it to sign records, and at some other specified later time it is removed from the zone.

Finally, we should note that the `dnssec-keygen` command supports the same set of switches so we could have set the dates when we created the key.

Reconfiguring BIND

Having created the keys with the appropriate timing information, the next step is to turn on DNSSEC signing. Below is a very simple `named.conf`; in our example environment, this file is `/etc/bind/named.conf`.

```
options {
    directory "/etc/bind";
    recursion no;
    minimal-responses yes;
};

zone "example.com" IN {
    type primary;
    file "example.com.db";
    auto-dnssec maintain;
    inline-signing yes;
};
```

Once the configuration file is updated, tell `named` to reload:

```
# rndc reload
server reload successful
```

Verifying That the Zone Is Signed Correctly

You should now check that the zone is signed. Follow the steps in *Verification*.

Uploading the DS Record to the Parent

As described in *Uploading Information to the Parent Zone*, we must now upload the new information to the parent zone. The format of the information and how to generate it is described in *Working With the Parent Zone*, although it is important to remember that you must use the contents of the KSK file that you generated above as part of the process.

When the DS record is published in the parent zone, your zone is fully signed.

Checking That Your Zone Can Be Validated

Finally, follow the steps in *How To Test Authoritative Zones* to confirm that a query recognizes the zone as properly signed and vouched for by the parent zone.

So... What Now?

Once the zone is signed, it must be monitored as described in *Maintenance Tasks*. However, as the time approaches for a key roll, you must create the new key. Of course, it is possible to create keys for the next fifty years all at once and set the key times appropriately. Whether the increased risk in having the private key files for future keys available on disk offsets the overhead of having to remember to create a new key before a rollover depends on your organization's security policy.

Fully Automatic Signing With `dnssec-keymgr`

`dnssec-keymgr` is a program supplied with BIND (versions 9.11 to 9.16) to help with key rollovers. When run, it compares the timing information for existing keys with the defined policy, and adjusts it if necessary. It also creates additional keys as required.

`dnssec-keymgr` is completely separate from `named`. As we will see, the policy states a coverage period; `dnssec-keymgr` generates enough key files to handle all rollovers in that period. However, it is a good idea to schedule it to run on a regular basis; that way there is no chance of forgetting to run it when the coverage period ends.

BIND should be set up exactly the same way as described in *Semi-Automatic Signing*, i.e., with `auto-dnssec` set to `maintain` and `inline-signing` set to `true`. Then a policy file must be created. The following is an example of such a file:

```
# cat policy.conf
policy standard {
    coverage 1y;
    algorithm RSASHA256;
    directory "/etc/bind";
    keyttl 2h;

    key-size ksk 4096;
    roll-period ksk 1y;
    pre-publish ksk 30d;
    post-publish ksk 30d;

    key-size zsk 2048;
    roll-period zsk 90d;
```

(continues on next page)

```
    pre-publish zsk 30d;
    post-publish zsk 30d;
};

zone example.com {
    policy standard;
};

zone example.net {
    policy standard;
    keyttl 300;
};
```

As can be seen, the syntax is similar to that of the `named` configuration file.

In the example above, we define a DNSSEC policy called "standard". Keys are created using the RSASHA256 algorithm, assigned a TTL of two hours, and placed in the directory `/etc/bind`. KSKs have a key size of 4096 bits and are expected to roll once a year; the new key is added to the zone 30 days before it becomes active, and is retained in the zone for 30 days after it is rolled. ZSKs have a key size of 2048 bits and roll every 90 days; like the KSKs, the are added to the zone 30 days before they are used for signing, and retained for 30 days after `named` ceases signing with them.

The policy is applied to two zones, `example.com` and `example.net`. The policy is applied unaltered to the former, but for the latter the setting for the DNSKEY TTL has been overridden and set to 300 seconds.

To apply the policy, we need to run `dnssec-keymgr`. Since this does not read the `named` configuration file, it relies on the presence of at least one key file for a zone to tell it that the zone is DNSSEC-enabled. If a key file does not already exist, we first need to create one for each zone. We can do that either by running `dnssec-keygen` to create a key file for each zone[9], or by specifying the zones in question on the command line. Here, we do the latter:

```
# dnssec-keymgr -c policy.conf example.com example.net
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -L 7200 -a RSASHA256 -b 2048 example.
↪net
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -L 7200 -fk -a RSASHA256 -b 4096␣
↪example.net
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20200915110318 -D 20201005110318␣
↪Kexample.net.+008+31339
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.net.+008+31339 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20201214110318 -D 20210113110318␣
↪Kexample.net.+008+14526
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.net.+008+14526 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20210314110318 -D 20210413110318␣
↪Kexample.net.+008+46069
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.net.+008+46069 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20210612110318 -D 20210712110318␣
↪Kexample.net.+008+13018
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.net.+008+13018 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20210617110318 -D 20210717110318␣
↪Kexample.net.+008+55237
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.net.+008+55237 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -L 7200 -a RSASHA256 -b 2048 example.
↪com
```

---

[9] Only one key file - for either a KSK or ZSK - is needed to signal the presence of the zone. `dnssec-keygen` creates files of both types as needed.

```
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -L 7200 -fk -a RSASHA256 -b 4096␣
↪example.com
# /usr/local/sbin/dnssec-settime -K /etc/bind -P 20200617110318 -A 20200617110318 -I␣
↪20200915110318 -D 20201015110318 Kexample.com.+008+31168
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.com.+008+31168 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20201214110318 -D 20210113110318␣
↪Kexample.com.+008+24199
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.com.+008+24199 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20210314110318 -D 20210413110318␣
↪Kexample.com.+008+08728
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.com.+008+08728 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -I 20210612110318 -D 20210712110318␣
↪Kexample.com.+008+12874
# /usr/local/sbin/dnssec-keygen -q -K /etc/bind -S Kexample.com.+008+12874 -L 7200 -i␣
↪2592000
# /usr/local/sbin/dnssec-settime -K /etc/bind -P 20200617110318 -A 20200617110318␣
↪Kexample.com.+008+26186
```

This creates enough key files to last for the coverage period, set in the policy file to be one year. The script should be run on a regular basis (probably via `cron`) to keep the reserve of key files topped up. With the shortest roll period set to 90 days, every 30 days is more than adequate.

At any time, you can check what key changes are coming up and whether the keys and timings are correct by using `dnssec-coverage`. For example, to check coverage for the next 60 days:

```
# dnssec-coverage -d 2h -m 1d -l 60d -K /etc/bind/keys
PHASE 1--Loading keys to check for internal timing problems
PHASE 2--Scanning future key events for coverage failures
Checking scheduled KSK events for zone example.net, algorithm RSASHA256...
  Wed Jun 17 11:03:18 UTC 2020:
    Publish: example.net/RSASHA256/55237 (KSK)
    Activate: example.net/RSASHA256/55237 (KSK)

Ignoring events after Sun Aug 16 11:47:24 UTC 2020

No errors found

Checking scheduled ZSK events for zone example.net, algorithm RSASHA256...
  Wed Jun 17 11:03:18 UTC 2020:
    Publish: example.net/RSASHA256/31339 (ZSK)
    Activate: example.net/RSASHA256/31339 (ZSK)
  Sun Aug 16 11:03:18 UTC 2020:
    Publish: example.net/RSASHA256/14526 (ZSK)

Ignoring events after Sun Aug 16 11:47:24 UTC 2020

No errors found

Checking scheduled KSK events for zone example.com, algorithm RSASHA256...
  Wed Jun 17 11:03:18 UTC 2020:
    Publish: example.com/RSASHA256/26186 (KSK)
    Activate: example.com/RSASHA256/26186 (KSK)

No errors found
```

```
Checking scheduled ZSK events for zone example.com, algorithm RSASHA256...
  Wed Jun 17 11:03:18 UTC 2020:
    Publish: example.com/RSASHA256/31168 (ZSK)
    Activate: example.com/RSASHA256/31168 (ZSK)
  Sun Aug 16 11:03:18 UTC 2020:
    Publish: example.com/RSASHA256/24199 (ZSK)

Ignoring events after Sun Aug 16 11:47:24 UTC 2020

No errors found
```

The `-d 2h` and `-m 1d` on the command line specify the maximum TTL for the DNSKEYs and other resource records in the zone: in this example two hours and one day, respectively. `dnssec-coverage` needs this information when it checks that the zones will remain secure through key rolls.

Fully Automatic Signing With `dnssec-policy`

The latest development in DNSSEC key management appeared with BIND 9.16, and is the full integration of key management into `named`. Managing the signing process and rolling of these keys has been described in *Easy-Start Guide for Signing Authoritative Zones* and is not repeated here. A few points are worth noting, though:

- The `dnssec-policy` statement in the `named` configuration file describes all aspects of the DNSSEC policy, including the signing. With `dnssec-keymgr`, this is split between two configuration files and two programs.

- When using `dnssec-policy`, there is no need to set the `auto-dnssec` and `inline-signing` options for a zone. The zone's `policy` statement implicitly does this.

- It is possible to manage some zones served by an instance of BIND through `dnssec-policy` and others through `dnssec-keymgr`, but this is not recommended. Although it should work, if you modify the configuration files and inadvertently specify a zone to be managed by both systems, BIND will not operate properly.

Manual Signing

Manual signing of a zone was the first method of signing introduced into BIND and offers, as the name suggests, no automation. The user must handle everything: create the keys, sign the zone file with them, load the signed zone, periodically re-sign the zone, and manage key rolls, including interaction with the parent. A user certainly can do all this, but why not use one of the automated methods? Nevertheless, it may be useful for test purposes, so we cover it briefly here.

The first step is to create the keys as described in *Generate Keys*. Then, edit the zone file to make sure the proper DNSKEY entries are included in your zone file. Finally, use the command `dnssec-signzone`:

```
# cd /etc/bind/keys/example.com/
# dnssec-signzone -A -t -N INCREMENT -o example.com -f /etc/bind/db/example.com.
↪signed.db \
> /etc/bind/db/example.com.db Kexample.com.+008+17694.key Kexample.com.+008+06817.key
Verifying the zone using the following algorithms: RSASHA256.
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                      ZSKs: 1 active, 0 stand-by, 0 revoked
/etc/bind/db/example.com.signed.db
Signatures generated:                      17
Signatures retained:                        0
Signatures dropped:                         0
```

```
Signatures successfully verified:          0
Signatures unsuccessfully verified:        0
Signing time in seconds:               0.046
Signatures per second:               364.634
Runtime in seconds:                    0.055
```

The -o switch explicitly defines the domain name (`example.com` in this case), while the -f switch specifies the output file name. The second line has three parameters: the unsigned zone name (`/etc/bind/db/example.com.db`), the ZSK file name, and the KSK file name. This also generates a plain text file `/etc/bind/db/example.com.signed.db`, which you can verify for correctness.

Finally, you'll need to update `named.conf` to load the signed version of the zone, which looks something like this:

```
zone "example.com" IN {
    type primary;
    file "db/example.com.signed.db";
};
```

Once the `rndc reconfig` command is issued, BIND serves a signed zone. The file `dsset-example.com` (created by `dnssec-signzone` when it signed the `example.com` zone) contains the DS record for the zone's KSK. You will need to pass that to the administrator of the parent zone, to be placed in the zone.

Since this is a manual process, you will need to re-sign periodically, as well as every time the zone data changes. You will also need to manually roll the keys by adding and removing DNSKEY records (and interacting with the parent) at the appropriate times.

9.6 Basic DNSSEC Troubleshooting

In this chapter, we cover some basic troubleshooting techniques, some common DNSSEC symptoms, and their causes and solutions. This is not a comprehensive "how to troubleshoot any DNS or DNSSEC problem" guide, because that could easily be an entire book by itself.

9.6.1 Query Path

The first step in troubleshooting DNS or DNSSEC should be to determine the query path. Whenever you are working with a DNS-related issue, it is always a good idea to determine the exact query path to identify the origin of the problem.

End clients, such as laptop computers or mobile phones, are configured to talk to a recursive name server, and the recursive name server may in turn forward requests on to other recursive name servers before arriving at the authoritative name server. The giveaway is the presence of the Authoritative Answer (`aa`) flag in a query response: when present, we know we are talking to the authoritative server; when missing, we are talking to a recursive server. The example below shows an answer to a query for `www.example.com` without the Authoritative Answer flag:

```
$ dig @10.53.0.3 www.example.com A

; <<>> DiG 9.16.0 <<>> @10.53.0.3 www.example.com a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62714
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
```

```
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c823fe302625db5b010000005e722b504d81bb01c2227259 (good)
;; QUESTION SECTION:
;www.example.com.          IN  A

;; ANSWER SECTION:
www.example.com.     60  IN  A   10.1.0.1

;; Query time: 3 msec
;; SERVER: 10.53.0.3#53(10.53.0.3)
;; WHEN: Wed Mar 18 14:08:16 GMT 2020
;; MSG SIZE  rcvd: 88
```

Not only do we not see the `aa` flag, we see an `ra` flag, which indicates Recursion Available. This indicates that the server we are talking to (10.53.0.3 in this example) is a recursive name server: although we were able to get an answer for `www.example.com`, we know that the answer came from somewhere else.

If we query the authoritative server directly, we get:

```
$ dig @10.53.0.2 www.example.com A

; <<>> DiG 9.16.0 <<>> @10.53.0.2 www.example.com a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39542
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available
...
```

The `aa` flag tells us that we are now talking to the authoritative name server for `www.example.com`, and that this is not a cached answer it obtained from some other name server; it served this answer to us right from its own database. In fact, the Recursion Available (`ra`) flag is not present, which means this name server is not configured to perform recursion (at least not for this client), so it could not have queried another name server to get cached results.

9.6.2 Visible DNSSEC Validation Symptoms

After determining the query path, it is necessary to determine whether the problem is actually related to DNSSEC validation. You can use the `+cd` flag in `dig` to disable validation, as described in *How Do I Know I Have a Validation Problem?*.

When there is indeed a DNSSEC validation problem, the visible symptoms, unfortunately, are very limited. With DNSSEC validation enabled, if a DNS response is not fully validated, it results in a generic SERVFAIL message, as shown below when querying against a recursive name server at 192.168.1.7:

```
$ dig @10.53.0.3 www.example.org. A

; <<>> DiG 9.16.0 <<>> @10.53.0.3 www.example.org A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 28947
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
```

```
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d1301968aca086ad010000005e723a7113603c01916d136b (good)
;; QUESTION SECTION:
;www.example.org.         IN  A

;; Query time: 3 msec
;; SERVER: 10.53.0.3#53(10.53.0.3)
;; WHEN: Wed Mar 18 15:12:49 GMT 2020
;; MSG SIZE  rcvd: 72
```

With `delv`, a "resolution failed" message is output instead:

```
$ delv @10.53.0.3 www.example.org. A +rtrace
;; fetch: www.example.org/A
;; resolution failed: SERVFAIL
```

BIND 9 logging features may be useful when trying to identify DNSSEC errors.

9.6.3 Basic Logging

DNSSEC validation error messages show up in `syslog` as a query error by default. Here is an example of what it may look like:

```
validating www.example.org/A: no valid signature found
RRSIG failed to verify resolving 'www.example.org/A/IN': 10.53.0.2#53
```

Usually, this level of error logging is sufficient. Debug logging, described in *BIND DNSSEC Debug Logging*, gives information on how to get more details about why DNSSEC validation may have failed.

9.6.4 BIND DNSSEC Debug Logging

A word of caution: before you enable debug logging, be aware that this may dramatically increase the load on your name servers. Enabling debug logging is thus not recommended for production servers.

With that said, sometimes it may become necessary to temporarily enable BIND debug logging to see more details of how and whether DNSSEC is validating. DNSSEC-related messages are not recorded in `syslog` by default, even if query log is enabled; only DNSSEC errors show up in `syslog`.

The example below shows how to enable debug level 3 (to see full DNSSEC validation messages) in BIND 9 and have it sent to `syslog`:

```
logging {
   channel dnssec_log {
        syslog daemon;
        severity debug 3;
        print-category yes;
    };
    category dnssec { dnssec_log; };
};
```

The example below shows how to log DNSSEC messages to their own file (here, `/var/log/dnssec.log`):

```
logging {
    channel dnssec_log {
        file "/var/log/dnssec.log";
        severity debug 3;
    };
    category dnssec { dnssec_log; };
};
```

After turning on debug logging and restarting BIND, a large number of log messages appear in `syslog`. The example below shows the log messages as a result of successfully looking up and validating the domain name `ftp.isc.org`.

```
validating ./NS: starting
validating ./NS: attempting positive response validation
  validating ./DNSKEY: starting
  validating ./DNSKEY: attempting positive response validation
  validating ./DNSKEY: verify rdataset (keyid=20326): success
  validating ./DNSKEY: marking as secure (DS)
validating ./NS: in validator_callback_dnskey
validating ./NS: keyset with trust secure
validating ./NS: resuming validate
validating ./NS: verify rdataset (keyid=33853): success
validating ./NS: marking as secure, noqname proof not needed
validating ftp.isc.org/A: starting
validating ftp.isc.org/A: attempting positive response validation
validating isc.org/DNSKEY: starting
validating isc.org/DNSKEY: attempting positive response validation
  validating isc.org/DS: starting
  validating isc.org/DS: attempting positive response validation
validating org/DNSKEY: starting
validating org/DNSKEY: attempting positive response validation
  validating org/DS: starting
  validating org/DS: attempting positive response validation
  validating org/DS: keyset with trust secure
  validating org/DS: verify rdataset (keyid=33853): success
  validating org/DS: marking as secure, noqname proof not needed
validating org/DNSKEY: in validator_callback_ds
validating org/DNSKEY: dsset with trust secure
validating org/DNSKEY: verify rdataset (keyid=9795): success
validating org/DNSKEY: marking as secure (DS)
  validating isc.org/DS: in fetch_callback_dnskey
  validating isc.org/DS: keyset with trust secure
  validating isc.org/DS: resuming validate
  validating isc.org/DS: verify rdataset (keyid=33209): success
  validating isc.org/DS: marking as secure, noqname proof not needed
validating isc.org/DNSKEY: in validator_callback_ds
validating isc.org/DNSKEY: dsset with trust secure
validating isc.org/DNSKEY: verify rdataset (keyid=7250): success
validating isc.org/DNSKEY: marking as secure (DS)
validating ftp.isc.org/A: in fetch_callback_dnskey
validating ftp.isc.org/A: keyset with trust secure
validating ftp.isc.org/A: resuming validate
validating ftp.isc.org/A: verify rdataset (keyid=27566): success
validating ftp.isc.org/A: marking as secure, noqname proof not needed
```

Note that these log messages indicate that the chain of trust has been established and `ftp.isc.org` has been successfully validated.

If validation had failed, you would see log messages indicating errors. We cover some of the most validation problems in

the next section.

9.6.5 Common Problems

Security Lameness

Similar to lame delegation in traditional DNS, security lameness refers to the condition when the parent zone holds a set of DS records that point to something that does not exist in the child zone. As a result, the entire child zone may "disappear," having been marked as bogus by validating resolvers.

Below is an example attempting to resolve the A record for a test domain name `www.example.net`. From the user's perspective, as described in *How Do I Know I Have a Validation Problem?*, only a SERVFAIL message is returned. On the validating resolver, we see the following messages in `syslog`:

```
named[126063]: validating example.net/DNSKEY: no valid signature found (DS)
named[126063]: no valid RRSIG resolving 'example.net/DNSKEY/IN': 10.53.0.2#53
named[126063]: broken trust chain resolving 'www.example.net/A/IN': 10.53.0.2#53
```

This gives us a hint that it is a broken trust chain issue. Let's take a look at the DS records that are published for the zone (with the keys shortened for ease of display):

```
$ dig @10.53.0.3 example.net. DS

; <<>> DiG 9.16.0 <<>> @10.53.0.3 example.net DS
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59602
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7026d8f7c6e77e2a010000005e735d7c9d038d061b2d24da (good)
;; QUESTION SECTION:
;example.net.            IN  DS

;; ANSWER SECTION:
example.net.        256 IN  DS  14956 8 2 9F3CACD...D3E3A396

;; Query time: 0 msec
;; SERVER: 10.53.0.3#53(10.53.0.3)
;; WHEN: Thu Mar 19 11:54:36 GMT 2020
;; MSG SIZE  rcvd: 116
```

Next, we query for the DNSKEY and RRSIG of `example.net` to see if there's anything wrong. Since we are having trouble validating, we can use the `+cd` option to temporarily disable checking and return results, even though they do not pass the validation tests. The `+multiline` option tells `dig` to print the type, algorithm type, and key id for DNSKEY records. Again, some long strings are shortened for ease of display:

```
$ dig @10.53.0.3 example.net. DNSKEY +dnssec +cd +multiline

; <<>> DiG 9.16.0 <<>> @10.53.0.3 example.net DNSKEY +cd +multiline +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42980
```

<div align="right">(continues on next page)</div>

```
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 4b5e7c88b3680c35010000005e73722057551f9f8be1990e (good)
;; QUESTION SECTION:
;example.net.          IN DNSKEY

;; ANSWER SECTION:
example.net.          287 IN DNSKEY 256 3 8 (
              AwEAAbu3NX...ADU/D7xjFFDu+8WRIn
              ) ; ZSK; alg = RSASHA256 ; key id = 35328
example.net.          287 IN DNSKEY 257 3 8 (
              AwEAAbKtU1...PPP4aQZTybk75ZW+uL
              6OJMAF63NO0s1nAZM2EWAVasbnn/X+J4N2rLuhk=
              ) ; KSK; alg = RSASHA256 ; key id = 27247
example.net.          287 IN RRSIG DNSKEY 8 2 300 (
              20811123173143 20180101000000 27247 example.net.
              Fz1sjClIoF...YEjzpAWuAj9peQ== )
example.net.          287 IN RRSIG DNSKEY 8 2 300 (
              20811123173143 20180101000000 35328 example.net.
              seKtUeJ4/l...YtDc1rcXTVlWIOw= )

;; Query time: 0 msec
;; SERVER: 10.53.0.3#53(10.53.0.3)
;; WHEN: Thu Mar 19 13:22:40 GMT 2020
;; MSG SIZE  rcvd: 962
```

Here is the problem: the parent zone is telling the world that `example.net` is using the key 14956, but the authoritative server indicates that it is using keys 27247 and 35328. There are several potential causes for this mismatch: one possibility is that a malicious attacker has compromised one side and changed the data. A more likely scenario is that the DNS administrator for the child zone did not upload the correct key information to the parent zone.

Incorrect Time

In DNSSEC, every record comes with at least one RRSIG, and each RRSIG contains two timestamps: one indicating when it becomes valid, and one when it expires. If the validating resolver's current system time does not fall within the two RRSIG timestamps, error messages appear in the BIND debug log.

The example below shows a log message when the RRSIG appears to have expired. This could mean the validating resolver system time is incorrectly set too far in the future, or the zone administrator has not kept up with RRSIG maintenance.

```
validating example.com/DNSKEY: verify failed due to bad signature (keyid=19036):␣
→RRSIG has expired
```

The log below shows that the RRSIG validity period has not yet begun. This could mean the validation resolver's system time is incorrectly set too far in the past, or the zone administrator has incorrectly generated signatures for this domain name.

```
validating example.com/DNSKEY: verify failed due to bad signature (keyid=4521): RRSIG␣
→validity period has not begun
```

Unable to Load Keys

This is a simple yet common issue. If the key files are present but unreadable by `named` for some reason, the `syslog` returns clear error messages, as shown below:

```
named[32447]: zone example.com/IN (signed): reconfiguring zone keys
named[32447]: dns_dnssec_findmatchingkeys: error reading key file Kexample.com.
→+008+06817.private: permission denied
named[32447]: dns_dnssec_findmatchingkeys: error reading key file Kexample.com.
→+008+17694.private: permission denied
named[32447]: zone example.com/IN (signed): next key event: 27-Nov-2014 20:04:36.521
```

However, if no keys are found, the error is not as obvious. Below shows the `syslog` messages after executing `rndc reload` with the key files missing from the key directory:

```
named[32516]: received control channel command 'reload'
named[32516]: loading configuration from '/etc/bind/named.conf'
named[32516]: reading built-in trusted keys from file '/etc/bind/bind.keys'
named[32516]: using default UDP/IPv4 port range: [1024, 65535]
named[32516]: using default UDP/IPv6 port range: [1024, 65535]
named[32516]: sizing zone task pool based on 6 zones
named[32516]: the working directory is not writable
named[32516]: reloading configuration succeeded
named[32516]: reloading zones succeeded
named[32516]: all zones loaded
named[32516]: running
named[32516]: zone example.com/IN (signed): reconfiguring zone keys
named[32516]: zone example.com/IN (signed): next key event: 27-Nov-2014 20:07:09.292
```

This happens to look exactly the same as if the keys were present and readable, and appears to indicate that `named` loaded the keys and signed the zone. It even generates the internal (raw) files:

```
# cd /etc/bind/db
# ls
example.com.db  example.com.db.jbk  example.com.db.signed
```

If `named` really loaded the keys and signed the zone, you should see the following files:

```
# cd /etc/bind/db
# ls
example.com.db  example.com.db.jbk  example.com.db.signed  example.com.db.signed.jnl
```

So, unless you see the `*.signed.jnl` file, your zone has not been signed.

Invalid Trust Anchors

In most cases, you never need to explicitly configure trust anchors. `named` supplies the current root trust anchor and, with the default setting of `dnssec-validation`, updates it on the infrequent occasions when it is changed.

However, in some circumstances you may need to explicitly configure your own trust anchor. As we saw in the *Trust Anchors* section, whenever a DNSKEY is received by the validating resolver, it is compared to the list of keys the resolver explicitly trusts to see if further action is needed. If the two keys match, the validating resolver stops performing further verification and returns the answer(s) as validated.

But what if the key file on the validating resolver is misconfigured or missing? Below we show some examples of log messages when things are not working properly.

First of all, if the key you copied is malformed, BIND does not even start and you will likely find this error message in syslog:

```
named[18235]: /etc/bind/named.conf.options:29: bad base64 encoding
named[18235]: loading configuration: failure
```

If the key is a valid base64 string but the key algorithm is incorrect, or if the wrong key is installed, the first thing you will notice is that virtually all of your DNS lookups result in SERVFAIL, even when you are looking up domain names that have not been DNSSEC-enabled. Below shows an example of querying a recursive server 10.53.0.3:

```
$ dig @10.53.0.3 www.example.com. A

; <<>> DiG 9.16.0 <<>> @10.53.0.3 www.example.org A +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 29586
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: ee078fc321fa1367010000005e73a58bf5f205ca47e04bed (good)
;; QUESTION SECTION:
;www.example.org.        IN  A
```

`delv` shows a similar result:

```
$ delv @192.168.1.7 www.example.com. +rtrace
;; fetch: www.example.com/A
;; resolution failed: SERVFAIL
```

The next symptom you see is in the DNSSEC log messages:

```
managed-keys-zone: DNSKEY set for zone '.' could not be verified with current keys
validating ./DNSKEY: starting
validating ./DNSKEY: attempting positive response validation
validating ./DNSKEY: no DNSKEY matching DS
validating ./DNSKEY: no DNSKEY matching DS
validating ./DNSKEY: no valid signature found (DS)
```

These errors are indications that there are problems with the trust anchor.

9.6.6 Negative Trust Anchors

BIND 9.11 introduced Negative Trust Anchors (NTAs) as a means to *temporarily* disable DNSSEC validation for a zone when you know that the zone's DNSSEC is misconfigured.

NTAs are added using the `rndc` command, e.g.:

```
$ rndc nta example.com
 Negative trust anchor added: example.com/_default, expires 19-Mar-2020 19:57:42.000
```

The list of currently configured NTAs can also be examined using `rndc`, e.g.:

```
$ rndc nta -dump
 example.com/_default: expiry 19-Mar-2020 19:57:42.000
```

The default lifetime of an NTA is one hour, although by default, BIND polls the zone every five minutes to see if the zone correctly validates, at which point the NTA automatically expires. Both the default lifetime and the polling interval may be configured via `named.conf`, and the lifetime can be overridden on a per-zone basis using the `-lifetime duration` parameter to `rndc nta`. Both timer values have a permitted maximum value of one week.

9.6.7 NSEC3 Troubleshooting

BIND includes a tool called `nsec3hash` that runs through the same steps as a validating resolver, to generate the correct hashed name based on NSEC3PARAM parameters. The command takes the following parameters in order: salt, algorithm, iterations, and domain. For example, if the salt is 1234567890ABCDEF, hash algorithm is 1, and iteration is 10, to get the NSEC3-hashed name for `www.example.com` we would execute a command like this:

```
$ nsec3hash 1234567890ABCEDF 1 10 www.example.com
RN7I9ME6E1I6BDKIP91B9TCE4FHJ7LKF (salt=1234567890ABCEDF, hash=1, iterations=10)
```

While it is unlikely you would construct a rainbow table of your own zone data, this tool may be useful when troubleshooting NSEC3 problems.

9.7 Advanced Discussions

9.7.1 Signature Validity Periods and Zone Re-Signing Intervals

In *How Are Answers Verified?*, we saw that record signatures have a validity period outside of which they are not valid. This means that at some point, a signature will no longer be valid and a query for the associated record will fail DNSSEC validation. But how long should a signature be valid for?

The maximum value for the validity period should be determined by the impact of a replay attack: if this is low, the period can be long; if high, the period should be shorter. There is no "right" value, but periods of between a few days to a month are common.

Deciding a minimum value is probably an easier task. Should something fail (e.g., a hidden primary distributing to secondary servers that actually answer queries), how long will it take before the failure is noticed, and how long before it is fixed? If you are a large 24x7 operation with operators always on-site, the answer might be less than an hour. In smaller companies, if the failure occurs just after everyone has gone home for a long weekend, the answer might be several days.

Again, there are no "right" values - they depend on your circumstances. The signature validity period you decide to use should be a value between the two bounds. At the time of this writing (mid-2020), the default policy used by BIND sets a value of 14 days.

To keep the zone valid, the signatures must be periodically refreshed since they expire - i.e., the zone must be periodically re-signed. The frequency of the re-signing depends on your network's individual needs. For example, signing puts a load on your server, so if the server is very highly loaded, a lower re-signing frequency is better. Another consideration is the signature lifetime: obviously the intervals between signings must not be longer than the signature validity period. But if you have set a signature lifetime close to the minimum (see above), the signing interval must be much shorter. What would happen if the system failed just before the zone was re-signed?

Again, there is no single "right" answer; it depends on your circumstances. The BIND 9 default policy sets the signature refresh interval to 5 days.

9.7.2 Proof of Non-Existence (NSEC and NSEC3)

How do you prove that something does not exist? This zen-like question is an interesting one, and in this section we provide an overview of how DNSSEC solves the problem.

Why is it even important to have authenticated denial of existence in DNS? Couldn't we just send back "hey, what you asked for does not exist," and somehow generate a digital signature to go with it, proving it really is from the correct authoritative source? Aside from the technical challenge of signing something that doesn't exist, this solution has flaws, one of which is it gives an attacker a way to create the appearance of denial of service by replaying this message on the network.

Let's use a little story, told three different ways, to illustrate how proof of nonexistence works. In our story, we run a small company with three employees: Alice, Edward, and Susan. For reasons that are far too complicated to go into, they don't have email accounts; instead, email for them is sent to a single account and a nameless intern passes the message to them. The intern has access to our private DNSSEC key to create signatures for their responses.

If we followed the approach of giving back the same answer no matter what was asked, when people emailed and asked for the message to be passed to "Bob," our intern would simply answer "Sorry, that person doesn't work here" and sign this message. This answer could be validated because our intern signed the response with our private DNSSEC key. However, since the signature doesn't change, an attacker could record this message. If the attacker were able to intercept our email, when the next person emailed asking for the message to be passed to Susan, the attacker could return the exact same message: "Sorry, that person doesn't work here," with the same signature. Now the attacker has successfully fooled the sender into thinking that Susan doesn't work at our company, and might even be able to convince all senders that no one works at this company.

To solve this problem, two different solutions were created. We will look at the first one, NSEC, next.

NSEC

The NSEC record is used to prove that something does not exist, by providing the name before it and the name after it. Using our tiny company example, this would be analogous to someone sending an email for Bob and our nameless intern responding with with: "I'm sorry, that person doesn't work here. The name before the location where 'Bob' would be is Alice, and the name after that is Edward." Let's say another email was received for a non-existent person, this time Oliver; our intern would respond "I'm sorry, that person doesn't work here. The name before the location where 'Oliver' would be is Edward, and the name after that is Susan." If another sender asked for Todd, the answer would be: "I'm sorry, that person doesn't work here. The name before the location where 'Todd' would be is Susan, and there are no other names after that."

So we end up with four NSEC records:

```
example.com.        300   IN   NSEC    alice.example.com.  A RRSIG NSEC
alice.example.com.  300 IN   NSEC    edward.example.com. A RRSIG NSEC
edward.example.com. 300 IN   NSEC    susan.example.com.  A RRSIG NSEC
susan.example.com.  300 IN   NSEC    example.com.        A RRSIG NSEC
```

What if the attacker tried to use the same replay method described earlier? If someone sent an email for Edward, none of the four answers would fit. If attacker replied with message #2, "I'm sorry, that person doesn't work here. The name before it is Alice, and the name after it is Edward," it is obviously false, since "Edward" is in the response; and the same goes for #3, Edward and Susan. As for #1 and #4, Edward does not fall in the alphabetical range before Alice or after Susan, so the sender can logically deduce that it was an incorrect answer.

When BIND signs your zone, the zone data is automatically sorted on the fly before generating NSEC records, much like how a phone directory is sorted.

The NSEC record allows for a proof of non-existence for record types. If you ask a signed zone for a name that exists but for a record type that doesn't (for that name), the signed NSEC record returned lists all of the record types that *do* exist for the requested domain name.

NSEC records can also be used to show whether a record was generated as the result of a wildcard expansion. The details of this are not within the scope of this document, but are described well in **RFC 7129**.

Unfortunately, the NSEC solution has a few drawbacks, one of which is trivial "zone walking." In our story, a curious person can keep sending emails, and our nameless, gullible intern keeps divulging information about our employees. Imagine if the sender first asked: "Is Bob there?" and received back the names Alice and Edward. Our sender could then email again: "Is Edwarda there?", and will get back Edward and Susan. (No, "Edwarda" is not a real name. However, it is the first name alphabetically after "Edward" and that is enough to get the intern to reply with a message telling us the next valid name after Edward.) Repeat the process enough times and the person sending the emails eventually learns every name in our company phone directory. For many of you, this may not be a problem, since the very idea of DNS is similar to a public phone book: if you don't want a name to be known publicly, don't put it in DNS! Consider using DNS views (split DNS) and only display your sensitive names to a select audience.

The second drawback of NSEC is actually increased operational overhead: there is no opt-out mechanism for insecure child zones. This generally is a problem for parent-zone operators dealing with a lot of insecure child zones, such as `.com`. To learn more about opt-out, please see *NSEC3 Opt-Out*.

NSEC3

NSEC3 adds two additional features that NSEC does not have:

1. It offers no easy zone enumeration.

2. It provides a mechanism for the parent zone to exclude insecure delegations (i.e., delegations to zones that are not signed) from the proof of non-existence.

Recall that in *NSEC* we provided a range of names to prove that something does not exist. But as it turns out, even disclosing these ranges of names becomes a problem: this made it very easy for the curious-minded to look at our entire zone. Not only that, unlike a zone transfer, this "zone walking" is more resource-intensive. So how do we disclose something without actually disclosing it?

The answer is actually quite simple: hashing functions, or one-way hashes. Without going into many details, think of it like a magical meat grinder. A juicy piece of ribeye steak goes in one end, and out comes a predictable shape and size of ground meat (hash) with a somewhat unique pattern. No matter how hard you try, you cannot turn the ground meat back into the ribeye steak: that's what we call a one-way hash.

NSEC3 basically runs the names through a one-way hash before giving them out, so the recipients can verify the non-existence without any knowledge of the actual names.

So let's tell our little story for the third time, this time with NSEC3. In this version, our intern is not given a list of actual names; he is given a list of "hashed" names. So instead of Alice, Edward, and Susan, the list he is given reads like this (hashes shortened for easier reading):

```
FSK5.... (produced from Edward)
JKMA.... (produced from Susan)
NTQ0.... (produced from Alice)
```

Then, an email is received for Bob again. Our intern takes the name Bob through a hash function, and the result is L8J2…, so he replies: "I'm sorry, that person doesn't work here. The name before that is JKMA…, and the name after that is NTQ0…". There, we proved Bob doesn't exist, without giving away any names! To put that into proper NSEC3 resource records, they would look like this (again, hashes shortened for ease of display):

```
FSK5....example.com. 300 IN NSEC3 1 0 10 1234567890ABCDEF  JKMA... A RRSIG
JKMA....example.com. 300 IN NSEC3 1 0 10 1234567890ABCDEF  NTQ0... A RRSIG
NTQ0....example.com. 300 IN NSEC3 1 0 10 1234567890ABCDEF  FSK5... A RRSIG
```

Note: Just because we employed one-way hash functions does not mean there is no way for a determined individual to figure out our zone data. Someone could still gather all of our NSEC3 records and hashed names and perform an offline brute-force attack by trying all possible combinations to figure out what the original name is. In our meat-grinder analogy, this would be like someone buying all available cuts of meat and grinding them up at home using the same model of meat grinder, and comparing the output with the meat you gave him. It is expensive and time-consuming (especially with real meat), but like everything else in cryptography, if someone has enough resources and time, nothing is truly private forever. If you are concerned about someone performing this type of attack on your zone data, read more about adding salt as described in *NSEC3 Salt*.

NSEC3PARAM

The above NSEC3 examples used four parameters: 1, 0, 10, and 1234567890ABCDEF. 1 represents the algorithm, 0 represents the opt-out flag, 10 represents the number of iterations, and 1234567890ABCDEF is the salt. Let's look at how each one can be configured:

- *Algorithm*: The only currently defined value is 1 for SHA-1, so there is no configuration field for it.

- *Opt-out*: Set this to 1 for NSEC3 opt-out, which we discuss in *NSEC3 Opt-Out*.

- *Iterations*: Iterations defines the number of additional times to apply the algorithm when generating an NSEC3 hash. More iterations yield more secure results, but consume more resources for both authoritative servers and validating resolvers. The considerations here are similar to those seen in *Key Sizes*, of security versus resources.

- *Salt*: The salt cannot be configured explicitly, but you can provide a salt length and `named` generates a random salt of the given length. We learn more about salt in *NSEC3 Salt*.

If you want to use these NSEC3 parameters for a zone, you can add the following configuration to your `dnssec-policy`. For example, to create an NSEC3 chain using the SHA-1 hash algorithm, with no opt-out flag, 5 iterations, and a salt that is 8 characters long, use:

```
dnssec-policy "nsec3" {
    ...
    nsec3param iterations 5 optout no salt-length 8;
};
```

To set the opt-out flag, 15 iterations, and no salt, use:

```
dnssec-policy "nsec3" {
    ...
    nsec3param iterations 15 optout yes salt-length 0;
};
```

NSEC3 Opt-Out

One of the advantages of NSEC3 over NSEC is the ability for a parent zone to publish less information about its child or delegated zones. Why would you ever want to do that? If a significant number of your delegations are not yet DNSSEC-aware, meaning they are still insecure or unsigned, generating DNSSEC-records for their NS and glue records is not a good use of your precious name server resources.

The resources may not seem like a lot, but imagine that you are the operator of busy top-level domains such as `.com` or `.net`, with millions of insecure delegated domain names: it quickly adds up. As of mid-2020, less than 1.5% of all `.com` zones are signed. Basically, without opt-out, with 1,000,000 delegations, only 5 of which are secure, you still have to generate NSEC RRsets for the other 999,995 delegations; with NSEC3 opt-out, you will have saved yourself 999,995 sets of records.

For most DNS administrators who do not manage a large number of delegations, the decision whether to use NSEC3 opt-out is probably not relevant.

To learn more about how to configure NSEC3 opt-out, please see *NSEC3 Opt-Out*.

NSEC3 Salt

As described in *NSEC3*, while NSEC3 does not put your zone data in plain public display, it is still not difficult for an attacker to collect all the hashed names and perform an offline attack. All that is required is running through all the combinations to construct a database of plaintext names to hashed names, also known as a "rainbow table."

There is one more feature NSEC3 gives us to provide additional protection: salt. Basically, salt gives us the ability to introduce further randomness into the hashed results. Whenever the salt is changed, any pre-computed rainbow table is rendered useless, and a new rainbow table must be re-computed. If the salt is changed periodically, it becomes difficult to construct a useful rainbow table, and thus difficult to walk the DNS zone data programmatically. How often you want to change your NSEC3 salt is up to you.

To learn more about the steps to take to change NSEC3, please see *Changing the NSEC3 Salt*.

NSEC or NSEC3?

So which one should you choose: NSEC or NSEC3? There is not a single right answer here that fits everyone; it comes down to your network's needs or requirements.

If you prefer not to make your zone easily enumerable, implementing NSEC3 paired with a periodically changed salt provides a certain level of privacy protection. However, someone could still randomly guess the names in your zone (such as "ftp" or "www"), as in the traditional insecure DNS.

If you have many delegations and need to be able to opt-out to save resources, NSEC3 is for you.

In other situations, NSEC is typically a good choice for most zone administrators, as it relieves the authoritative servers of the additional cryptographic operations that NSEC3 requires, and NSEC is comparatively easier to troubleshoot than NSEC3.

NSEC3 in conjunction with `dnssec-policy` is supported in BIND as of version 9.16.9.

9.7.3 DNSSEC Keys

Types of Keys

Although DNSSEC documentation talks about three types of keys, they are all the same thing - but they have different roles. The roles are:

Zone-Signing Key (ZSK) This is the key used to sign the zone. It signs all records in the zone apart from the DNSSEC key-related RRsets: DNSKEY, CDS, and CDNSKEY.

Key-Signing Key (KSK) This is the key used to sign the DNSSEC key-related RRsets and is the key used to link the parent and child zones. The parent zone stores a digest of the KSK. When a resolver verifies the chain of trust it checks to see that the DS record in the parent (which holds the digest of a key) matches a key in the DNSKEY RRset, and that it is able to use that key to verify the DNSKEY RRset. If it can do that, the resolver knows that it can trust the DNSKEY resource records, and so can use one of them to validate the other records in the zone.

Combined Signing Key (CSK) A CSK combines the functionality of a ZSK and a KSK. Instead of having one key for signing the zone and one for linking the parent and child zones, a CSK is a single key that serves both roles.

It is important to realize the terms ZSK, KSK, and CSK describe how the keys are used - all these keys are represented by DNSKEY records. The following examples are the DNSKEY records from a zone signed with a KSK and ZSK:

```
$ dig @192.168.1.12 example.com DNSKEY

; <<>> DiG 9.16.0 <<>> @192.168.1.12 example.com dnskey +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54989
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5258d7ed09db0d76010000005ea1cc8c672d8db27a464e37 (good)
;; QUESTION SECTION:
;example.com.        IN DNSKEY

;; ANSWER SECTION:
example.com.         60 IN DNSKEY 256 3 13 (
                tAeXLtIQ3aVDqqS/1UVRt9AE6/nzfoAuaT1Vy4dYl2CK
                pLNcUJxME1Z//pnGXY+HqDU7Gr5HkJY8V0W3r5fzlw==
                ) ; ZSK; alg = ECDSAP256SHA256 ; key id = 63722
example.com.         60 IN DNSKEY 257 3 13 (
                cxkNegsgubBPXSra5ug2P8rWy63B8jTnS4n0IYSsD9eW
                VhiyQDmdgevKUhfG3SE1wbLChjJc2FAbvSZ1qk03Nw==
                ) ; KSK; alg = ECDSAP256SHA256 ; key id = 42933
```

… and a zone signed with just a CSK:

```
$ dig @192.168.1.13 example.com DNSKEY

; <<>> DiG 9.16.0 <<>> @192.168.1.13 example.com dnskey +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22628
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: bf19ee914b5df46e010000005ea1cd02b66c06885d274647 (good)
;; QUESTION SECTION:
;example.com.        IN DNSKEY

;; ANSWER SECTION:
example.com.         60 IN DNSKEY 257 3 13 (
                p0XM6AJ68qid2vtOdyGaeH1jnrdk2GhZeVvGzXfP/PNa
                71wGtzR6jdUrTbXo5Z1W5QeeJF4dls4lh4z7DByF5Q==
                ) ; KSK; alg = ECDSAP256SHA256 ; key id = 1231
```

The only visible difference between the records (apart from the key data itself) is the value of the flags fields; this is 256 for a ZSK and 257 for a KSK or CSK. Even then, the flags field is only a hint to the software using it as to the role of the key: zones can be signed by any key. The fact that a CSK and KSK both have the same flags emphasizes this. A KSK usually only signs the DNSSEC key-related RRsets in a zone, whereas a CSK is used to sign all records in the zone.

The original idea of separating the function of the key into a KSK and ZSK was operational. With a single key, changing it for any reason is "expensive," as it requires interaction with the parent zone (e.g., uploading the key to the parent may require manual interaction with the organization running that zone). By splitting it, interaction with the parent is required

only if the KSK is changed; the ZSK can be changed as often as required without involving the parent.

The split also allows the keys to be of different lengths. So the ZSK, which is used to sign the record in the zone, can be of a (relatively) short length, lowering the load on the server. The KSK, which is used only infrequently, can be of a much longer length. The relatively infrequent use also allows the private part of the key to be stored in a way that is more secure but that may require more overhead to access, e.g., on an HSM (see *Hardware Security Modules (HSMs)*).

In the early days of DNSSEC, the idea of splitting the key went more or less unchallenged. However, with the advent of more powerful computers and the introduction of signaling methods between the parent and child zones (see *The CDS and CDNSKEY Resource Records*), the advantages of a ZSK/KSK split are less clear and, for many zones, a single key is all that is required.

As with many questions related to the choice of DNSSEC policy, the decision on which is "best" is not clear and depends on your circumstances.

Which Algorithm?

There are three algorithm choices for DNSSEC as of this writing (mid-2020):

- RSA

- Elliptic Curve DSA (ECDSA)

- Edwards Curve Digital Security Algorithm (EdDSA)

All are supported in BIND 9, but only RSA and ECDSA (specifically RSASHA256 and ECDSAP256SHA256) are mandatory to implement in DNSSEC. However, RSA is a little long in the tooth, and ECDSA/EdDSA are emerging as the next new cryptographic standards. In fact, the US federal government recommended discontinuing RSA use altogether by September 2015 and migrating to using ECDSA or similar algorithms.

For now, use ECDSAP256SHA256 but keep abreast of developments in this area. For details about rolling over DNSKEYs to a new algorithm, see *Algorithm Rollovers*.

Key Sizes

If using RSA keys, the choice of key sizes is a classic issue of finding the balance between performance and security. The larger the key size, the longer it takes for an attacker to crack the key; but larger keys also mean more resources are needed both when generating signatures (authoritative servers) and verifying signatures (recursive servers).

Of the two sets of keys, ZSK is used much more frequently. ZSK is used whenever zone data changes or when signatures expire, so performance certainly is of a bigger concern. As for KSK, it is used less frequently, so performance is less of a factor, but its impact is bigger because of its role in signing other keys.

In earlier versions of this guide, the following key lengths were chosen for each set, with the recommendation that they be rotated more frequently for better security:

- *ZSK*: RSA 1024 bits, rollover every year

- *KSK*: RSA 2048 bits, rollover every five years

These should be considered minimum RSA key sizes. At the time of this writing (mid-2020), the root zone and many TLDs are already using 2048 bit ZSKs. If you choose to implement larger key sizes, keep in mind that larger key sizes result in larger DNS responses, which this may mean more load on network resources. Depending on your network configuration, end users may even experience resolution failures due to the increased response sizes, as discussed in *What's EDNS All About (And Why Should I Care)?*.

ECDSA key sizes can be much smaller for the same level of security, e.g., an ECDSA key length of 224 bits provides the same level of security as a 2048-bit RSA key. Currently BIND 9 sets a key size of 256 for all ECDSA keys.

Key Storage

Public Key Storage

The beauty of a public key cryptography system is that the public key portion can and should be distributed to as many people as possible. As the administrator, you may want to keep the public keys on an easily accessible file system for operational ease, but there is no need to securely store them, since both ZSK and KSK public keys are published in the zone data as DNSKEY resource records.

Additionally, a hash of the KSK public key is also uploaded to the parent zone (see *Working With the Parent Zone* for more details), and is published by the parent zone as DS records.

Private Key Storage

Ideally, private keys should be stored offline, in secure devices such as a smart card. Operationally, however, this creates certain challenges, since the private key is needed to create RRSIG resource records, and it is a hassle to bring the private key out of storage every time the zone file changes or signatures expire.

A common approach to strike the balance between security and practicality is to have two sets of keys: a ZSK set and a KSK set. A ZSK private key is used to sign zone data, and can be kept online for ease of use, while a KSK private key is used to sign just the DNSKEY (the ZSK); it is used less frequently, and can be stored in a much more secure and restricted fashion.

For example, a KSK private key stored on a USB flash drive that is kept in a fireproof safe, only brought online once a year to sign a new pair of ZSKs, combined with a ZSK private key stored on the network file system and available for routine use, may be a good balance between operational flexibility and security.

For more information on changing keys, please see *Key Rollovers*.

Hardware Security Modules (HSMs)

A Hardware Security Module (HSM) may come in different shapes and sizes, but as the name indicates, it is a physical device or devices, usually with some or all of the following features:

- Tamper-resistant key storage
- Strong random-number generation
- Hardware for faster cryptographic operations

Most organizations do not incorporate HSMs into their security practices due to cost and the added operational complexity.

BIND supports Public Key Cryptography Standard #11 (PKCS #11) for communication with HSMs and other cryptographic support devices. For more information on how to configure BIND to work with an HSM, please refer to the BIND 9 Administrator Reference Manual.

9.7.4 Rollovers

Key Rollovers

A key rollover is where one key in a zone is replaced by a new one. There are arguments for and against regularly rolling keys. In essence these are:

Pros:

1. Regularly changing the key hinders attempts at determination of the private part of the key by cryptanalysis of signatures.

2. It gives administrators practice at changing a key; should a key ever need to be changed in an emergency, they would not be doing it for the first time.

Cons:

1. A lot of effort is required to hack a key, and there are probably easier ways of obtaining it, e.g., by breaking into the systems on which it is stored.

2. Rolling the key adds complexity to the system and introduces the possibility of error. We are more likely to have an interruption to our service than if we had not rolled it.

Whether and when to roll the key is up to you. How serious would the damage be if a key were compromised without you knowing about it? How serious would a key roll failure be?

Before going any further, it is worth noting that if you sign your zone with either of the fully automatic methods (described in ref:*signing_alternative_ways*), you don't really need to concern yourself with the details of a key rollover: BIND 9 takes care of it all for you. If you are doing a manual key roll or are setting up the keys for a semi-automatic key rollover, you do need to familiarize yourself with the various steps involved and the timing details.

Rolling a key is not as simple as replacing the DNSKEY statement in the zone. That is an essential part of it, but timing is everything. For example, suppose that we run the `example.com` zone and that a friend queries for the AAAA record of `www.example.com`. As part of the resolution process (described in *How Does DNSSEC Change DNS Lookup?*), their recursive server looks up the keys for the `example.com` zone and uses them to verify the signature associated with the AAAA record. We'll assume that the records validated successfully, so they can use the address to visit `example.com`'s website.

Let's also assume that immediately after the lookup, we want to roll the ZSK for `example.com`. Our first attempt at this is to remove the old DNSKEY record and signatures, add a new DNSKEY record, and re-sign the zone with it. So one minute our server is serving the old DNSKEY and records signed with the old key, and the next minute it is serving the new key and records signed with it. We've achieved our goal - we are serving a zone signed with the new keys; to check this is really the case, we booted up our laptop and looked up the AAAA record `ftp.example.com`. The lookup succeeded so all must be well. Or is it? Just to be sure, we called our friend and asked them to check. They tried to lookup `ftp.example.com` but got a SERVFAIL response from their recursive server. What's going on?

The answer, in a word, is "caching." When our friend looked up `www.example.com`, their recursive server retrieved and cached not only the AAAA record, but also a lot of other records. It cached the NS records for `com` and `example.com`, as well as the AAAA (and A) records for those name servers (and this action may, in turn, have caused the lookup and caching of other NS and AAAA/A records). Most importantly for this example, it also looked up and cached the DNSKEY records for the root, `com`, and `example.com` zones. When a query was made for `ftp.example.com`, the recursive server believed it already had most of the information we needed. It knew what nameservers served `example.com` and their addresses, so it went directly to one of those to get the AAAA record for `ftp.example.com` and its associated signature. But when it tried to validate the signature, it used the cached copy of the DNSKEY, and that is when our friend had the problem. Their recursive server had a copy of the old DNSKEY in its cache, but the AAAA record for `ftp.example.com` was signed with the new key. So, not surprisingly, the signature could not validate.

How should we roll the keys for `example.com`? A clue to the answer is to note that the problem came about because the DNSKEY records were cached by the recursive server. What would have happened had our friend flushed the DNSKEY records from the recursive server's cache before making the query? That would have worked; those records would have

been retrieved from `example.com`'s nameservers at the same time that we retrieved the AAAA record for `ftp.` `example.com`. Our friend's server would have obtained the new key along with the AAAA record and associated signature created with the new key, and all would have been well.

As it is obviously impossible for us to notify all recursive server operators to flush our DNSKEY records every time we roll a key, we must use another solution. That solution is to wait for the recursive servers to remove old records from caches when they reach their TTL. How exactly we do this depends on whether we are trying to roll a ZSK, a KSK, or a CSK.

ZSK Rollover Methods

The ZSK can be rolled in one of the following two ways:

1. *Pre-Publication*: Publish the new ZSK into zone data before it is actually used. Wait at least one TTL interval, so the world's recursive servers know about both keys, then stop using the old key and generate a new RRSIG using the new key. Wait at least another TTL, so the cached old key data is expunged from the world's recursive servers, and then remove the old key.

   The benefit of the pre-publication approach is it does not dramatically increase the zone size; however, the duration of the rollover is longer. If insufficient time has passed after the new ZSK is published, some resolvers may only have the old ZSK cached when the new RRSIG records are published, and validation may fail. This is the method described in *ZSK Rollover*.

2. *Double-Signature*: Publish the new ZSK and new RRSIG, essentially doubling the size of the zone. Wait at least one TTL interval, and then remove the old ZSK and old RRSIG.

   The benefit of the double-signature approach is that it is easier to understand and execute, but it causes a significantly increased zone size during a rollover event.

KSK Rollover Methods

Rolling the KSK requires interaction with the parent zone, so operationally this may be more complex than rolling ZSKs. There are three methods of rolling the KSK:

1. *Double-KSK*: Add the new KSK to the DNSKEY RRset, which is then signed with both the old and new keys. After waiting for the old RRset to expire from caches, change the DS record in the parent zone. After waiting a further TTL interval for this change to be reflected in caches, remove the old key from the RRset.

   Basically, the new KSK is added first at the child zone and used to sign the DNSKEY; then the DS record is changed, followed by the removal of the old KSK. Double-KSK keeps the interaction with the parent zone to a minimum, but for the duration of the rollover, the size of the DNSKEY RRset is increased.

2. *Double-DS*: Publish the new DS record. After waiting for this change to propagate into caches, change the KSK. After a further TTL interval during which the old DNSKEY RRset expires from caches, remove the old DS record.

   Double-DS is the reverse of Double-KSK: the new DS is published at the parent first, then the KSK at the child is updated, then the old DS at the parent is removed. The benefit is that the size of the DNSKEY RRset is kept to a minimum, but interactions with the parent zone are increased to two events. This is the method described in *KSK Rollover*.

3. *Double-RRset*: Add the new KSK to the DNSKEY RRset, which is then signed with both the old and new key, and add the new DS record to the parent zone. After waiting a suitable interval for the old DS and DNSKEY RRsets to expire from caches, remove the old DNSKEY and old DS record.

   Double-RRset is the fastest way to roll the KSK (i.e., it has the shortest rollover time), but has the drawbacks of both of the other methods: a larger DNSKEY RRset and two interactions with the parent.

CSK Rollover Methods

Rolling the CSK is more complex than rolling either the ZSK or KSK, as the timing constraints relating to both the parent zone and the caching of records by downstream recursive servers must be taken into account. There are numerous possible methods that are a combination of ZSK rollover and KSK rollover methods. BIND 9 automatic signing uses a combination of ZSK Pre-Publication and Double-KSK rollover.

Emergency Key Rollovers

Keys are generally rolled on a regular schedule - if you choose to roll them at all. But sometimes, you may have to rollover keys out-of-schedule due to a security incident. The aim of an emergency rollover is to re-sign the zone with a new key as soon as possible, because when a key is suspected of being compromised, a malicious attacker (or anyone who has access to the key) could impersonate your server and trick other validating resolvers into believing that they are receiving authentic, validated answers.

During an emergency rollover, follow the same operational procedures described in *Rollovers*, with the added task of reducing the TTL of the current active (potentially compromised) DNSKEY RRset, in an attempt to phase out the compromised key faster before the new key takes effect. The time frame should be significantly reduced from the 30-days-apart example, since you probably do not want to wait up to 60 days for the compromised key to be removed from your zone.

Another method is to carry a spare key with you at all times. If you have a second key pre-published and that one is not compromised at the same time as the first key, you could save yourself some time by immediately activating the spare key if the active key is compromised. With pre-publication, all validating resolvers should already have this spare key cached, thus saving you some time.

With a KSK emergency rollover, you also need to consider factors related to your parent zone, such as how quickly they can remove the old DS records and publish the new ones.

As with many other facets of DNSSEC, there are multiple aspects to take into account when it comes to emergency key rollovers. For more in-depth considerations, please check out **RFC 7583**.

Algorithm Rollovers

From time to time, new digital signature algorithms with improved security are introduced, and it may be desirable for administrators to roll over DNSKEYs to a new algorithm, e.g., from RSASHA1 (algorithm 5 or 7) to RSASHA256 (algorithm 8). The algorithm rollover steps must be followed with care to avoid breaking DNSSEC validation.

If you are managing DNSSEC by using the `dnssec-policy` configuration, `named` handles the rollover for you. Simply change the algorithm for the relevant keys, and `named` uses the new algorithm when the key is next rolled. It performs a smooth transition to the new algorithm, ensuring that the zone remains valid throughout rollover.

If you are using other methods to sign the zone, the administrator needs to do more work. As with other key rollovers, when the zone is a primary zone, an algorithm rollover can be accomplished using dynamic updates or automatic key rollovers. For secondary zones, only automatic key rollovers are possible, but the `dnssec-settime` utility can be used to control the timing.

In any case, the first step is to put DNSKEYs in place using the new algorithm. You must generate the `K*` files for the new algorithm and put them in the zone's key directory, where `named` can access them. Take care to set appropriate ownership and permissions on the keys. If the `auto-dnssec` zone option is set to `maintain`, `named` automatically signs the zone with the new keys, based on their timing metadata when the `dnssec-loadkeys-interval` elapses or when you issue the `rndc loadkeys` command. Otherwise, for primary zones, you can use `nsupdate` to add the new DNSKEYs to the zone; this causes `named` to use them to sign the zone. For secondary zones, e.g., on a "bump in the wire" signing server, `nsupdate` cannot be used.

Once the zone has been signed by the new DNSKEYs (and you have waited for at least one TTL period), you must inform the parent zone and any trust anchor repositories of the new KSKs, e.g., you might place DS records in the parent zone through your DNS registrar's website.

Before starting to remove the old algorithm from a zone, you must allow the maximum TTL on its DS records in the parent zone to expire. This assures that any subsequent queries retrieve the new DS records for the new algorithm. After the TTL has expired, you can remove the DS records for the old algorithm from the parent zone and any trust anchor repositories. You must then allow another maximum TTL interval to elapse so that the old DS records disappear from all resolver caches.

The next step is to remove the DNSKEYs using the old algorithm from your zone. Again this can be accomplished using `nsupdate` to delete the old DNSKEYs (for primary zones only) or by automatic key rollover when `auto-dnssec` is set to `maintain`. You can cause the automatic key rollover to take place immediately by using the `dnssec-settime` utility to set the *Delete* date on all keys to any time in the past. (See the `dnssec-settime -D <date/offset>` option.)

After adjusting the timing metadata, the `rndc loadkeys` command causes `named` to remove the DNSKEYs and RRSIGs for the old algorithm from the zone. Note also that with the `nsupdate` method, removing the DNSKEYs also causes `named` to remove the associated RRSIGs automatically.

Once you have verified that the old DNSKEYs and RRSIGs have been removed from the zone, the final (optional) step is to remove the key files for the old algorithm from the key directory.

9.7.5 Other Topics

DNSSEC and Dynamic Updates

Dynamic DNS (DDNS) is actually independent of DNSSEC. DDNS provides a mechanism, separate from editing the zone file or zone database, to edit DNS data. Most DNS clients and servers are able to handle dynamic updates, and DDNS can also be integrated as part of your DHCP environment.

When you have both DNSSEC and dynamic updates in your environment, updating zone data works the same way as with traditional (insecure) DNS: you can use `rndc freeze` before editing the zone file, and `rndc thaw` when you have finished editing, or you can use the command `nsupdate` to add, edit, or remove records like this:

```
$ nsupdate
> server 192.168.1.13
> update add xyz.example.com. 300 IN A 1.1.1.1
> send
> quit
```

The examples provided in this guide make `named` automatically re-sign the zone whenever its content has changed. If you decide to sign your own zone file manually, you need to remember to execute the `dnssec-signzone` command whenever your zone file has been updated.

As far as system resources and performance are concerned, be mindful that with a DNSSEC zone that changes frequently, every time the zone changes your system is executing a series of cryptographic operations to (re)generate signatures and NSEC or NSEC3 records.

DNSSEC on Private Networks

Let's clarify what we mean: in this section, "private networks" really refers to a private or internal DNS view. Most DNS products offer the ability to have different versions of DNS answers, depending on the origin of the query. This feature is often called "DNS views" or "split DNS," and is most commonly implemented as an "internal" versus an "external" setup.

For instance, your organization may have a version of `example.com` that is offered to the world, and its names most likely resolve to publicly reachable IP addresses. You may also have an internal version of `example.com` that is only accessible when you are on the company's private networks or via a VPN connection. These private networks typically fall under 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16 for IPv4.

So what if you want to offer DNSSEC for your internal version of `example.com`? This can be done: the golden rule is to use the same key for both the internal and external versions of the zones. This avoids problems that can occur when machines (e.g., laptops) move between accessing the internal and external zones, when it is possible that they may have cached records from the wrong zone.

Introduction to DANE

With your DNS infrastructure secured with DNSSEC, information can now be stored in DNS and its integrity and authenticity can be proved. One of the new features that takes advantage of this is the DNS-Based Authentication of Named Entities, or DANE. This improves security in a number of ways, including:

- The ability to store self-signed X.509 certificates and bypass having to pay a third party (such as a Certificate Authority) to sign the certificates (**RFC 6698**).

- Improved security for clients connecting to mail servers (**RFC 7672**).

- A secure way of getting public PGP keys (**RFC 7929**).

9.7.6 Disadvantages of DNSSEC

DNSSEC, like many things in this world, is not without its problems. Below are a few challenges and disadvantages that DNSSEC faces.

1. *Increased, well, everything*: With DNSSEC, signed zones are larger, thus taking up more disk space; for DNSSEC-aware servers, the additional cryptographic computation usually results in increased system load; and the network packets are bigger, possibly putting more strains on the network infrastructure.

2. *Different security considerations*: DNSSEC addresses many security concerns, most notably cache poisoning. But at the same time, it may introduce a set of different security considerations, such as amplification attack and zone enumeration through NSEC. These concerns are still being identified and addressed by the Internet community.

3. *More complexity*: If you have read this far, you have probably already concluded this yourself. With additional resource records, keys, signatures, and rotations, DNSSEC adds many more moving pieces on top of the existing DNS machine. The job of the DNS administrator changes, as DNS becomes the new secure repository of everything from spam avoidance to encryption keys, and the amount of work involved to troubleshoot a DNS-related issue becomes more challenging.

4. *Increased fragility*: The increased complexity means more opportunities for things to go wrong. Before DNSSEC, DNS was essentially "add something to the zone and forget it." With DNSSEC, each new component - re-signing, key rollover, interaction with parent zone, key management - adds more opportunity for error. It is entirely possible that a failure to validate a name may come down to errors on the part of one or more zone operators rather than the result of a deliberate attack on the DNS.

5. *New maintenance tasks*: Even if your new secure DNS infrastructure runs without any hiccups or security breaches, it still requires regular attention, from re-signing to key rollovers. While most of these can be automated, some of the tasks, such as KSK rollover, remain manual for the time being.

6. *Not enough people are using it today*: While it's estimated (as of mid-2020) that roughly 30% of the global Internet DNS traffic is validating[10] , that doesn't mean that many of the DNS zones are actually signed. What this means is, even if your company's zone is signed today, fewer than 30% of the Internet's servers are taking advantage of this extra security. It gets worse: with less than 1.5% of the `.com` domains signed, even if your DNSSEC validation is enabled today, it's not likely to buy you or your users a whole lot more protection until these popular domain names decide to sign their zones.

The last point may have more impact than you realize. Consider this: HTTP and HTTPS make up the majority of traffic on the Internet. While you may have secured your DNS infrastructure through DNSSEC, if your web hosting is outsourced to a third party that does not yet support DNSSEC in its own domain, or if your web page loads contents and components from insecure domains, end users may experience validation problems when trying to access your web page. For example, although you may have signed the zone `company.com`, the web address `www.company.com` may actually be a CNAME to `foo.random-cloud-provider.com`. As long as `random-cloud-provider.com` remains an insecure DNS zone, users cannot fully validate everything when they visit your web page and could be redirected elsewhere by a cache poisoning attack.

9.8 Recipes

This chapter provides step-by-step "recipes" for some common DNSSEC configurations.

9.8.1 DNSSEC Signing

There are two recipes here: the first shows an example using DNSSEC signing on the primary server, which has been covered in this guide; the second shows how to setup a "bump in the wire" between a hidden primary and the secondary servers to seamlessly sign the zone "on the fly."

Primary Server DNSSEC Signing

In this recipe, our servers are illustrated as shown in *DNSSEC Signing Recipe #1*: we have a primary server (192.168.1.1) and three secondary servers (192.168.1.2, 192.168.1.3, and 192.168.1.4) that receive zone transfers. To get the zone signed, we need to reconfigure the primary server. Once reconfigured, a signed version of the zone is generated on the fly; zone transfers take care of synchronizing the signed zone data to all secondary name servers, without configuration or software changes on them.
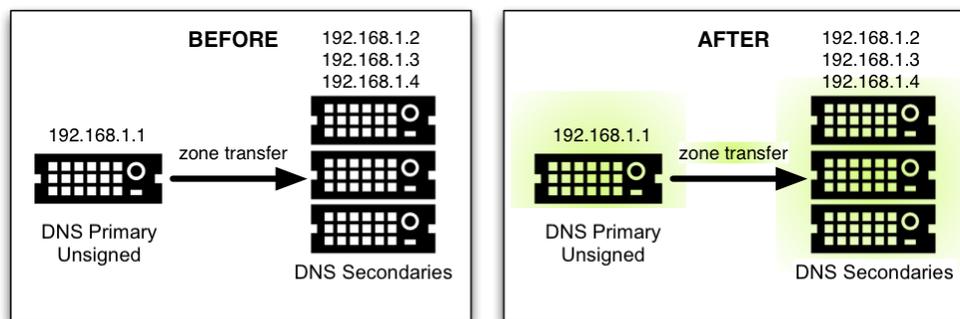

Fig. 5: DNSSEC Signing Recipe #1

---

[10] Based on APNIC statistics at https://stats.labs.apnic.net/dnssec/XA

Using the method described in *Easy-Start Guide for Signing Authoritative Zones*, we just need to add a `dnssec-policy` statement to the relevant zone clause. This is what the `named.conf` zone statement looks like on the primary server, 192.168.1.1:

```
zone "example.com" IN {
    type primary;
    file "db/example.com.db";
    key-directory "keys/example.com";
    dnssec-policy default;
    allow-transfer { 192.168.1.2; 192.168.1.3; 192.168.1.4; };
};
```

We have chosen to use the default policy, storing the keys generated for the zone in the directory `keys/example.com`. To use a custom policy, define the policy in the configuration file and select it in the zone statement (as described in *Creating a Custom DNSSEC Policy*).

On the secondary servers, `named.conf` does not need to be updated, and it looks like this:

```
zone "example.com" IN {
    type secondary;
    file "db/example.com.db";
    primaries { 192.168.1.1; };
};
```

In fact, the secondary servers do not even need to be running BIND; they can run any DNS product that supports DNSSEC.

"Bump in the Wire" Signing

In this recipe, we take advantage of the power of automated signing by placing an additional name server (192.168.1.5) between the hidden primary (192.168.1.1) and the DNS secondaries (192.168.1.2, 192.168.1.3, and 192.168.1.4). The additional name server, 192.168.1.5, acts as a "bump in the wire," taking an unsigned zone from the hidden primary, and sending out signed data on the other end to the secondary name servers. The steps described in this recipe may be used as part of a DNSSEC deployment strategy, since it requires only minimal changes made to the existing hidden DNS primary and DNS secondaries.
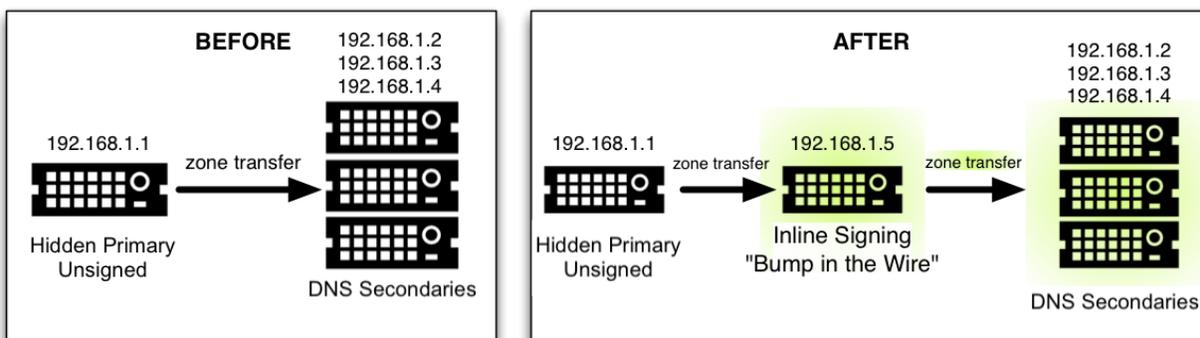

Fig. 6: DNSSEC Signing Recipe #2

It is important to remember that 192.168.1.1 in this case is a hidden primary not exposed to the world, and it must not be listed in the NS RRset. Otherwise the world will get conflicting answers: unsigned answers from the hidden primary and signed answers from the other name servers.

The only configuration change needed on the hidden primary, 192.168.1.1, is to make sure it allows our middle box to perform a zone transfer:

```
zone "example.com" IN {
    ...
    allow-transfer { 192.168.1.5; };
    ...
};
```

On the middle box, 192.168.1.5, all the tasks described in *Easy-Start Guide for Signing Authoritative Zones* still need to be performed, such as generating key pairs and uploading information to the parent zone. This server is configured as secondary to the hidden primary 192.168.1.1 to receive the unsigned data; then, using keys accessible to this middle box, to sign data on the fly; and finally, to send out the signed data via zone transfer to the other three DNS secondaries. Its `named.conf` zone statement looks like this:

```
zone example.com {
    type secondary;
    primaries { 192.168.1.1; };
    file "db/example.com.db";
    key-directory "keys/example.com";
    dnssec-policy default;
    allow-transfer { 192.168.1.2; 192.168.1.3; 192.168.1.4; };
};
```

(As before, the default policy has been selected here. See *Creating a Custom DNSSEC Policy* for instructions on how to define and use a custom policy.)

Finally, on the three secondary servers, the configuration should be updated to receive a zone transfer from 192.168.1.5 (the middle box) instead of from 192.168.1.1 (the hidden primary). If using BIND, the `named.conf` file looks like this:

```
zone "example.com" IN {
    type secondary;
    file "db/example.com.db";
    primaries { 192.168.1.5; };    # this was 192.168.1.1 before!
};
```

9.8.2 Rollovers

If you are signing your zone using a `dnssec-policy` statement, this section is not really relevant to you. In the policy statement, you set how long you want your keys to be valid for, the time taken for information to propagate through your zone, the time it takes for your parent zone to register a new DS record, etc., and that's more or less it. `named` implements everything for you automatically, apart from uploading the new DS records to your parent zone - which is covered in *Uploading Information to the Parent Zone*. (Some screenshots from a session where a KSK is uploaded to the parent zone are presented here for convenience.) However, these recipes may be useful in describing what happens through the rollover process and what you should be monitoring.

ZSK Rollover

This recipe covers how to perform a ZSK rollover using what is known as the Pre-Publication method. For other ZSK rolling methods, please see *ZSK Rollover Methods* in *Advanced Discussions*.

Below is a sample timeline for a ZSK rollover to occur on January 1, 2021:

1. December 1, 2020 (one month before rollover)

   - Generate new ZSK

   - Add DNSKEY for new ZSK to zone

2. January 1, 2021 (day of rollover)

   - New ZSK used to replace RRSIGs for the bulk of the zone

3. February 1, 2021 (one month after rollover)

   - Remove old ZSK DNSKEY RRset from zone

   - DNSKEY signatures made with KSK are changed

The current active ZSK has the ID 17694 in the example below. For more information on key management and rollovers, please see *Rollovers*.

One Month Before ZSK Rollover

On December 1, 2020, a month before the example rollover, you (as administrator) should change the parameters on the current key (17694). Set it to become inactive on January 1, 2021 and be deleted from the zone on February 1, 2021; also, generate a successor key (51623):

```
# cd /etc/bind/keys/example.com/
# dnssec-settime -I 20210101 -D 20210201 Kexample.com.+008+17694
./Kexample.com.+008+17694.key/GoDaddy

./Kexample.com.+008+17694.private
# dnssec-keygen -S Kexample.com.+008+17694
Generating key pair..++++++ ...........++++++
Kexample.com.+008+51623
```

The first command gets us into the key directory `/etc/bind/keys/example.com/`, where keys for `example.com` are stored.

The second, `dnssec-settime`, sets an inactive (`-I`) date of January 1, 2021, and a deletion (`-D`) date of February 1, 2021, for the current ZSK (`Kexample.com.+008+17694`).

The third command, `dnssec-keygen`, creates a successor key, using the exact same parameters (algorithms, key sizes, etc.) as the current ZSK. The new ZSK created in our example is `Kexample.com.+008+51623`.

Make sure the successor keys are readable by `named`.

`named`'s logging messages indicate when the next key checking event is scheduled to occur, the frequency of which can be controlled by `dnssec-loadkeys-interval`. The log message looks like this:

```
zone example.com/IN (signed): next key event: 01-Dec-2020 00:13:05.385
```

And you can check the publish date of the key by looking at the key file:

```
# cd /etc/bind/keys/example.com
# cat Kexample.com.+008+51623.key
; This is a zone-signing key, keyid 11623, for example.com.
; Created: 20201130160024 (Mon Dec  1 00:00:24 2020)
; Publish: 20201202000000 (Fri Dec  2 08:00:00 2020)
; Activate: 20210101000000 (Sun Jan  1 08:00:00 2021)
...
```

Since the publish date is set to the morning of December 2, and our example scenario takes place on December 1, the next morning you will notice that your zone has gained a new DNSKEY record, but the new ZSK is not yet being used to generate signatures. Below is the abbreviated output - with shortened DNSKEY and RRSIG - when querying the authoritative name server, 192.168.1.13:

```
$ dig @192.168.1.13 example.com. DNSKEY +dnssec +multiline

...
;; ANSWER SECTION:
example.com.        600 IN DNSKEY 257 3 8 (
                AwEAAcWDps...lM3NRn/G/R
                ) ; KSK; alg = RSASHA256; key id = 6817
example.com.        600 IN DNSKEY 256 3 8 (
                AwEAAbi6Vo...qBW5+iAqNz
                ) ; ZSK; alg = RSASHA256; key id = 51623
example.com.        600 IN DNSKEY 256 3 8 (
                AwEAAcjGaU...0rzuu55If5
                ) ; ZSK; alg = RSASHA256; key id = 17694
example.com.        600 IN RRSIG DNSKEY 8 2 600 (
                20210101000000 20201201230000 6817 example.com.
                LAiaJM26T7...FU9syh/TQ= )
example.com.        600 IN RRSIG DNSKEY 8 2 600 (
                20210101000000 20201201230000 17694 example.com.
                HK4EBbbOpj...n5V6nvAkI= )
...
```

For good measure, let's take a look at the SOA record and its signature for this zone. Notice the RRSIG is signed by the current ZSK, 17694. This will come in handy later when you want to verify whether the new ZSK is in effect:

```
$ dig @192.168.1.13 example.com. SOA +dnssec +multiline

...
;; ANSWER SECTION:
example.com.        600 IN SOA ns1.example.com. admin.example.com. (
                2020120102 ; serial
                1800       ; refresh (30 minutes)
                900        ; retry (15 minutes)
                2419200    ; expire (4 weeks)
                300        ; minimum (5 minutes)
                )
example.com.        600 IN RRSIG SOA 8 2 600 (
                20201230160109 20201130150109 17694 example.com.
                YUTC8rFULaWbW+nAHzbfGwNqzARHevpryzRIJMvZBYPo
                NAeejNk9saNAoCYKWxGJ0YBc2k+r5fYq1Mg4ll2JkBF5
                buAsAYLw8vEOIxVpXwlArY+oSp9T1w2wfTZ0vhVIxaYX
                6dkcz4I3wbDx2xmG0yngtA6A8lAchERx2EGy0RM= )
```

These are all the manual tasks you need to perform for a ZSK rollover. If you have followed the configuration examples in this guide of using `inline-signing` and `auto-dnssec`, everything else is automated for you by BIND.

Day of ZSK Rollover

On the actual day of the rollover, although there is technically nothing for you to do, you should still keep an eye on the zone to make sure new signatures are being generated by the new ZSK (51623 in this example). The easiest way is to query the authoritative name server 192.168.1.13 for the SOA record as you did a month ago:

```
$ dig @192.168.1.13 example.com. SOA +dnssec +multiline

...
;; ANSWER SECTION:
example.com.         600 IN SOA ns1.example.com. admin.example.com. (
                2020112011 ; serial
                1800       ; refresh (30 minutes)
                900        ; retry (15 minutes)
                2419200    ; expire (4 weeks)
                300        ; minimum (5 minutes)
                )
example.com.         600 IN RRSIG SOA 8 2 600 (
                20210131000000 20201231230000 51623 example.com.
                J4RMNpJPOmMidElyBugJp0RLqXoNqfvo/2AT6yAAvx9X
                zZRL1cuhkRcyCSLZ9Z+zZ2y4u2lvQGrNiondaKdQCor7
                uTqH5WCPoqalOCBjqU7c7vlAM27O9RD11nzPNpVQ7xPs
                y5nkGqf83OXTK26IfnjU1jqiUKSzg6QR7+XpLk0= )
...
```

As you can see, the signature generated by the old ZSK (17694) has disappeared, replaced by a new signature generated from the new ZSK (51623).

Note: Not all signatures will disappear magically on the same day; it depends on when each one was generated. In the worst-case scenario, a new signature could have been signed by the old ZSK (17694) moments before it was deactivated, meaning that the signature could live for almost 30 more days, until just before February 1.

This is why it is important to keep the old ZSK in the zone and not delete it right away.

One Month After ZSK Rollover

Again, technically there is nothing you need to do on this day, but it doesn't hurt to verify that the old ZSK (17694) is now completely gone from your zone. `named` will not touch `Kexample.com.+008+17694.private` and `Kexample.com.+008+17694.key` on your file system. Running the same `dig` command for DNSKEY should suffice:

```
$ dig @192.168.1.13 example.com. DNSKEY +multiline +dnssec

...
;; ANSWER SECTION:
example.com.         600 IN DNSKEY 257 3 8 (
                AwEAAcWDps...lM3NRn/G/R
                ) ; KSK; alg = RSASHA256; key id = 6817
example.com.         600 IN DNSKEY 256 3 8 (
                AwEAAdeCGr...1DnEfX+Xzn
                ) ; ZSK; alg = RSASHA256; key id = 51623
example.com.         600 IN RRSIG DNSKEY 8 2 600 (
                20170203000000 20170102230000 6817 example.com.
                KHY8P0zE21...Y3szrmjAM= )
```

```
example.com.        600 IN RRSIG DNSKEY 8 2 600 (
                20170203000000 20170102230000 51623 example.com.
                G2g3crN17h...Oe4gw6gH8= )
...
```

Congratulations, the ZSK rollover is complete! As for the actual key files (the files ending in `.key` and `.private`), they may be deleted at this point, but they do not have to be.

KSK Rollover

This recipe describes how to perform KSK rollover using the Double-DS method. For other KSK rolling methods, please see *KSK Rollover Methods* in *Advanced Discussions*. The registrar used in this recipe is GoDaddy. Also for this recipe, we are keeping the number of DS records down to just one per active set using just SHA-1, for the sake of better clarity, although in practice most zone operators choose to upload two DS records as shown in *Working With the Parent Zone*. For more information on key management and rollovers, please see *Rollovers*.

Below is a sample timeline for a KSK rollover to occur on January 1, 2021:

1. December 1, 2020 (one month before rollover)

   • Change timer on the current KSK

   • Generate new KSK and DS records

   • Add DNSKEY for the new KSK to zone

   • Upload new DS records to parent zone

2. January 1, 2021 (day of rollover)

   • Use the new KSK to sign all DNSKEY RRsets, which generates new RRSIGs

   • Add new RRSIGs to the zone

   • Remove RRSIG for the old ZSK from zone

   • Start using the new KSK to sign DNSKEY

3. February 1, 2021 (one month after rollover)

   • Remove the old KSK DNSKEY from zone

   • Remove old DS records from parent zone

The current active KSK has the ID 24828, and this is the DS record that has already been published by the parent zone:

```
# dnssec-dsfromkey -a SHA-1 Kexample.com.+007+24828.key
example.com. IN DS 24828 7 1 D4A33E8DD550A9567B4C4971A34AD6C4B80A6AD3
```

One Month Before KSK Rollover

On December 1, 2020, a month before the planned rollover, you (as administrator) should change the parameters on the current key. Set it to become inactive on January 1, 2021, and be deleted from the zone on February 1st, 2021; also generate a successor key (23550). Finally, generate a new DS record based on the new key, 23550:

```
# cd /etc/bind/keys/example.com/
# dnssec-settime -I 20210101 -D 20210201 Kexample.com.+007+24828
./Kexample.com.+007+24848.key
```

```
./Kexample.com.+007+24848.private
# dnssec-keygen -S Kexample.com.+007+24848
Generating key pair.................................................................
↪...................++ ..................................++
Kexample.com.+007+23550
# dnssec-dsfromkey -a SHA-1 Kexample.com.+007+23550.key
example.com. IN DS 23550 7 1 54FCF030AA1C79C0088FDEC1BD1C37DAA2E70DFB
```

The first command gets us into the key directory `/etc/bind/keys/example.com/`, where keys for `example.com` are stored.

The second, `dnssec-settime`, sets an inactive (`-I`) date of January 1, 2021, and a deletion (`-D`) date of February 1, 2021 for the current KSK (`Kexample.com.+007+24848`).

The third command, `dnssec-keygen`, creates a successor key, using the exact same parameters (algorithms, key sizes, etc.) as the current KSK. The new key pair created in our example is `Kexample.com.+007+23550`.

The fourth and final command, `dnssec-dsfromkey`, creates a DS record from the new KSK (23550), using SHA-1 as the digest type. Again, in practice most people generate two DS records for both supported digest types (SHA-1 and SHA-256), but for our example here we are only using one to keep the output small and hopefully clearer.

Make sure the successor keys are readable by `named`.

The `syslog` message indicates when the next key checking event is. The log message looks like this:

```
zone example.com/IN (signed): next key event: 01-Dec-2020 00:13:05.385
```

You can check the publish date of the key by looking at the key file:

```
# cd /etc/bind/keys/example.com
# cat Kexample.com.+007+23550.key
; This is a key-signing key, keyid 23550, for example.com.
; Created: 20201130160024 (Thu Dec  1 00:00:24 2020)
; Publish: 20201202000000 (Fri Dec  2 08:00:00 2020)
; Activate: 20210101000000 (Sun Jan  1 08:00:00 2021)
...
```

Since the publish date is set to the morning of December 2, and our example scenario takes place on December 1, the next morning you will notice that your zone has gained a new DNSKEY record based on your new KSK, but with no corresponding RRSIG yet. Below is the abbreviated output - with shortened DNSKEY and RRSIG - when querying the authoritative name server, 192.168.1.13:

```
$ dig @192.168.1.13 example.com. DNSKEY +dnssec +multiline

...
;; ANSWER SECTION:
example.com.    300 IN DNSKEY 256 3 7 (
                AwEAAdYqAc...TiSlrma6Ef
                ) ; ZSK; alg = NSEC3RSASHA1; key id = 29747
example.com.    300 IN DNSKEY 257 3 7 (
                AwEAAeTJ+w...O+Zy9j0m63
                ) ; KSK; alg = NSEC3RSASHA1; key id = 24828
example.com.    300 IN DNSKEY 257 3 7 (
                AwEAAc1BQN...Wdc0qoH21H
                ) ; KSK; alg = NSEC3RSASHA1; key id = 23550
example.com.    300 IN RRSIG DNSKEY 7 2 300 (
                20201206125617 20201107115617 24828 example.com.
                4y1iPVJOrK...aC3iF9vgc= )
```

```
example.com.   300 IN RRSIG DNSKEY 7 2 300 (
                20201206125617 20201107115617 29747 example.com.
                g/gfmPjr+y...rt/S/xjPo= )

...
```

Anytime after generating the DS record, you can upload it; it is not necessary to wait for the DNSKEY to be published in your zone, since this new KSK is not active yet. You can do it immediately after the new DS record has been generated on December 1, or you can wait until the next day after you have verified that the new DNSKEY record is added to the zone. Below are some screenshots from GoDaddy's web-based interface, used to add a new DS record[11].

1. After logging in, click the green "Launch" button next to the domain name you want to manage.
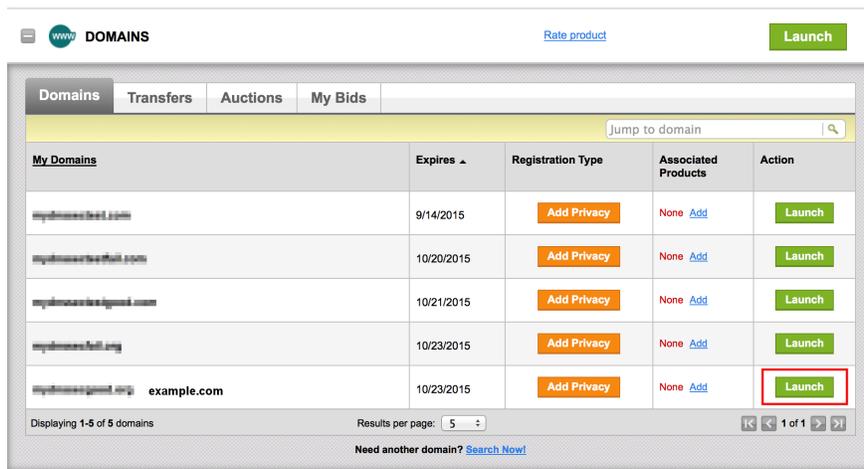
Fig. 7: Upload DS Record Step #1

2. Scroll down to the "DS Records" section and click "Manage."

3. A dialog appears, displaying the current key (24828). Click "Add DS Record."

4. Enter the Key ID, algorithm, digest type, and the digest, then click "Next."

5. Address any errors and click "Finish."

6. Both DS records are shown. Click "Save."

Finally, let's verify that the registrar has published the new DS record. This may take anywhere from a few minutes to a few days, depending on your parent zone. You can verify whether your parent zone has published the new DS record by querying for the DS record of your zone. In the example below, the Google public DNS server 8.8.8.8 is used:

```
$ dig @8.8.8.8 example.com. DS

...
;; ANSWER SECTION:
example.com.    21552   IN  DS  24828 7 1 D4A33E8DD550A9567B4C4971A34AD6C4B80A6AD3
example.com.    21552   IN  DS  23550 7 1 54FCF030AA1C79C0088FDEC1BD1C37DAA2E70DFB
```

You can also query your parent zone's authoritative name servers directly to see if these records have been published. DS records will not show up on your own authoritative zone, so you cannot query your own name servers for them. In this recipe, the parent zone is `.com`, so querying a few of the `.com` name servers is another appropriate verification.

---

[11] The screenshots were taken from GoDaddy's interface at the time the original version of this guide was published (2015). It may have changed since then.

Fig. 8: Upload DS Record Step #2

Fig. 9: Upload DS Record Step #3

Fig. 10: Upload DS Record Step #4
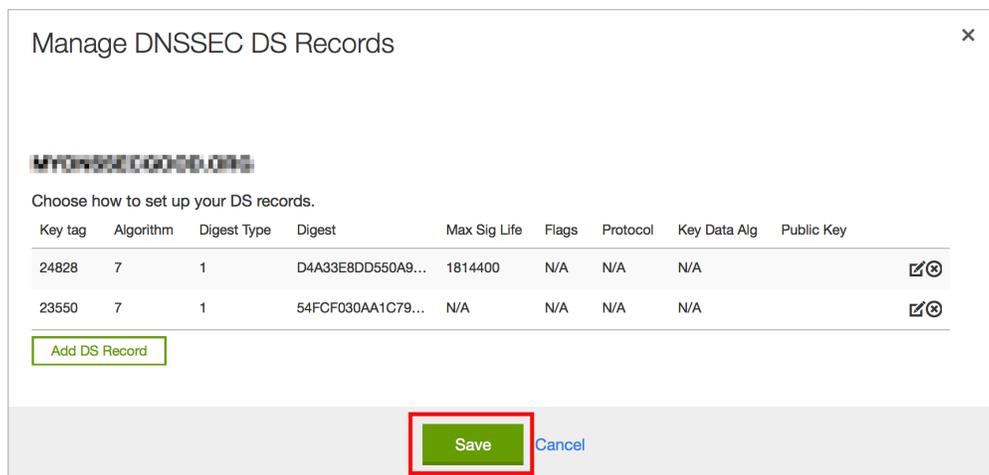

Fig. 11: Upload DS Record Step #5

Fig. 12: Upload DS Record Step #6

Day of KSK Rollover

If you have followed the examples in this document, as described in *Easy-Start Guide for Signing Authoritative Zones*, there is technically nothing you need to do manually on the actual day of the rollover. However, you should still keep an eye on the zone to make sure new signature(s) are being generated by the new KSK (23550 in this example). The easiest way is to query the authoritative name server 192.168.1.13 for the same DNSKEY and signatures, as you did a month ago:

```
$ dig @192.168.1.13 example.com. DNSKEY +dnssec +multiline

...
;; ANSWER SECTION:
example.com.     300 IN DNSKEY 256 3 7 (
                 AwEAAdYqAc...TiSlrma6Ef
                 ) ; ZSK; alg = NSEC3RSASHA1; key id = 29747
example.com.     300 IN DNSKEY 257 3 7 (
                 AwEAAeTJ+w...O+Zy9j0m63
                 ) ; KSK; alg = NSEC3RSASHA1; key id = 24828
example.com.     300 IN DNSKEY 257 3 7 (
                 AwEAAc1BQN...Wdc0qoH21H
                 ) ; KSK; alg = NSEC3RSASHA1; key id = 23550
example.com.     300 IN RRSIG DNSKEY 7 2 300 (
                 20210201074900 20210101064900 23550 mydnssecgood.org.
                 S6zTbBTfvU...Ib5eXkbtE= )
example.com.     300 IN RRSIG DNSKEY 7 2 300 (
                 20210105074900 20201206064900 29747 mydnssecgood.org.
                 VY5URQA2/d...OVKr1+KX8= )
...
```

As you can see, the signature generated by the old KSK (24828) has disappeared, replaced by a new signature generated from the new KSK (23550).

One Month After KSK Rollover

While the removal of the old DNSKEY from the zone should be automated by `named`, the removal of the DS record is manual. You should make sure the old DNSKEY record is gone from your zone first, by querying for the DNSKEY records of the zone; this time we expect not to see the key with an ID of 24828:

```
$ dig @192.168.1.13 example.com. DNSKEY +dnssec +multiline

...
;; ANSWER SECTION:
example.com.    300 IN DNSKEY 256 3 7 (
                AwEAAdYqAc...TiSlrma6Ef
                ) ; ZSK; alg = NSEC3RSASHA1; key id = 29747
example.com.    300 IN DNSKEY 257 3 7 (
                AwEAAc1BQN...Wdc0qoH21H
                ) ; KSK; alg = NSEC3RSASHA1; key id = 23550
example.com.    300 IN RRSIG DNSKEY 7 2 300 (
                20210208000000 20210105230000 23550 mydnssecgood.org.
                Qw9Em3dDok...bNCS7KISw= )
example.com.    300 IN RRSIG DNSKEY 7 2 300 (
                20210208000000 20210105230000 29747 mydnssecgood.org.
                OuelpIlpY9...XfsKupQgc= )
...
```

Since the key with the ID 24828 is gone, you can now remove the old DS record for that key from our parent zone. Be careful to remove the correct DS record. If you accidentally remove the new DS record(s) with key ID 23550, it could lead to a problem called "security lameness," as discussed in *Security Lameness*, and may cause users to be unable to resolve any names in the zone.

1. After logging in (again, GoDaddy.com in our example) and launching the domain, scroll down to the "DS Records" section and click Manage.

2. A dialog appears, displaying both keys (24828 and 23550). Use the far right-hand X button to remove key 24828.

3. Key 24828 now appears crossed out; click "Save" to complete the removal.

Congratulations, the KSK rollover is complete! As for the actual key files (ending in `.key` and `.private`), they may be deleted at this point, but they do not have to be.

9.8.3 NSEC and NSEC3

Migrating from NSEC to NSEC3

This recipe describes how to transition from using NSEC to NSEC3, as described in *Proof of Non-Existence (NSEC and NSEC3)*. This recipe assumes that the zones are already signed, and that `named` is configured according to the steps described in *Easy-Start Guide for Signing Authoritative Zones*.

> **Warning:** If your zone is signed with RSASHA1 (algorithm 5), you cannot migrate to NSEC3 without also performing an algorithm rollover to RSASHA1-NSEC3-SHA1 (algorithm 7), as described in *Algorithm Rollovers*. This ensures that older validating resolvers that do not understand NSEC3 will fall back to treating the zone as unsecured (rather than "bogus"), as described in Section 2 of **RFC 5155**.

To enable NSEC3, update your `dnssec-policy` and add the desired NSEC3 parameters. The example below enables NSEC3 for zones with the `standard` DNSSEC policy, using 10 iterations, no opt-out, and a random string that is 16 characters long:

Fig. 13: Remove DS Record Step #1
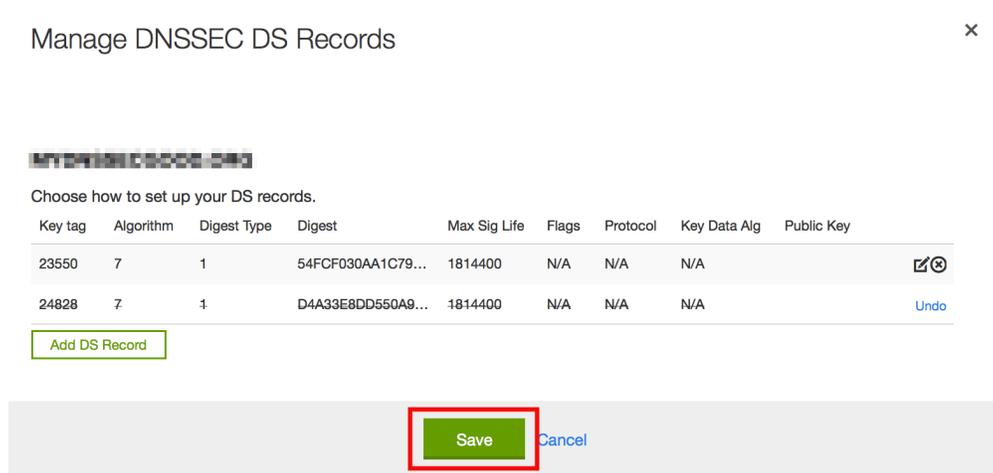

Fig. 14: Remove DS Record Step #2

Fig. 15: Remove DS Record Step #3

```
dnssec-policy "standard" {
    nsec3param iterations 10 optout no salt-length 16;
};
```

Then reconfigure the server with `rndc`. You can tell that it worked if you see the following debug log messages:

```
Oct 21 13:47:21 received control channel command 'reconfig'
Oct 21 13:47:21 zone example.com/IN (signed): zone_addnsec3chain(1,CREATE,10,
→1234567890ABCDEF)
```

You can also verify that it worked by querying for a name that you know does not exist, and checking for the presence of the NSEC3 record. For example:

```
$ dig @192.168.1.13 thereisnowaythisexists.example.com. A +dnssec +multiline

...
TOM10UQBL336NFAQB3P6MOO53LSVG8UI.example.com. 300 IN NSEC3 1 0 10 1234567890ABCDEF (
              TQ9QBEGA6CROHEOC8KIH1A2C06IVQ5ER
              NS SOA RRSIG DNSKEY NSEC3PARAM )
...
```

Our example used four parameters: 1, 0, 10, and 1234567890ABCDEF, in order. 1 represents the algorithm, 0 represents the opt-out flag, 10 represents the number of iterations, and 1234567890ABCDEF is the salt. To learn more about each of these parameters, please see *NSEC3PARAM*.

Migrating from NSEC3 to NSEC

Migrating from NSEC3 back to NSEC is easy; just remove the `nsec3param` configuration option from your `dnssec-policy` and reconfigure the name server. You can tell that it worked if you see these messages in the log:

```
named[14093]: received control channel command 'reconfig'
named[14093]: zone example.com/IN: zone_addnsec3chain(1,REMOVE,10,1234567890ABCDEF)
```

You can also query for a name that you know does not exist, and you should no longer see any traces of NSEC3 records.

```
$ dig @192.168.1.13 reieiergiuhewhiouwe.example.com. A +dnssec +multiline

...
example.com.        300 IN NSEC aaa.example.com. NS SOA RRSIG NSEC DNSKEY
...
ns1.example.com.    300 IN NSEC web.example.com. A RRSIG NSEC
...
```

Changing the NSEC3 Salt

In *NSEC3 Salt*, we discuss the reasons why you may want to change your salt periodically for better privacy. In this recipe, we look at what command to execute to actually change the salt, and how to verify that it has been changed.

The `dnssec-policy` currently has no easy way to re-salt using the same salt length, so to change your NSEC3 salt you need to change the `salt-length` value and reconfigure your server. You should see the following messages in the log, assuming your old salt was "1234567890ABCDEF" and `named` created "FEDCBA09" (salt length 8) as the new salt:

```
named[15848]: zone example.com/IN: zone_addnsec3chain(1,REMOVE,10,1234567890ABCDEF)
named[15848]: zone example.com/IN: zone_addnsec3chain(1,CREATE|OPTOUT,10,
→FEDCBA0987654321)
```

To verify that it worked, you can query the name server (192.168.1.13 in our example) for a name that you know does not exist, and check the NSEC3 record returned:

```
$ dig @192.168.1.13 thereisnowaythisexists.example.com. A +dnssec +multiline

...
TOM10UQBL336NFAQB3P6MOO53LSVG8UI.example.com. 300 IN NSEC3 1 0 10 FEDCBA09 (
                TQ9QBEGA6CROHEOC8KIH1A2C06IVQ5ER
                NS SOA RRSIG DNSKEY NSEC3PARAM )
...
```

If you want to use the same salt length, you can repeat the above steps and go back to your original length value.

NSEC3 Opt-Out

This recipe discusses how to enable and disable NSEC3 opt-out, and how to show the results of each action. As discussed in *NSEC3 Opt-Out*, NSEC3 opt-out is a feature that can help conserve resources on parent zones with many delegations that have not yet been signed.

Because the NSEC3PARAM record does not keep track of whether opt-out is used, it is hard to check whether changes need to be made to the NSEC3 chain if the flag is changed. Similar to changing the NSEC3 salt, your best option is to change the value of `optout` together with another NSEC3 parameter, like `iterations`, and in a following step restore the `iterations` value.

For this recipe we assume the zone `example.com` has the following four entries (for this example, it is not relevant what record types these entries are):

- `ns1.example.com`
- `ftp.example.com`
- `www.example.com`
- `web.example.com`

And the zone `example.com` has five delegations to five subdomains, only one of which is signed and has a valid DS RRset:

- `aaa.example.com`, not signed

- `bbb.example.com`, signed

- `ccc.example.com`, not signed

- `ddd.example.com`, not signed

- `eee.example.com`, not signed

Before enabling NSEC3 opt-out, the zone `example.com` contains ten NSEC3 records; below is the list with the plain text name before the actual NSEC3 record:

- *aaa.example.com*: 9NE0VJGTRTMJOS171EC3EDL6I6GT4P1Q.example.com.

- *bbb.example.com*: AESO0NT3N44OOSDQS3PSL0HACHUE1O0U.example.com.

- *ccc.example.com*: SF3J3VR29LDDO3ONT1PM6HAPHV372F37.example.com.

- *ddd.example.com*: TQ9QBEGA6CROHEOC8KIH1A2C06IVQ5ER.example.com.

- *eee.example.com*: L16L08NEH48IFQIEIPS1HNRMQ523MJ8G.example.com.

- *ftp.example.com*: JKMAVHL8V7EMCL8JHIEN8KBOAB0MGUK2.example.com.

- *ns1.example.com*: FSK5TK9964BNE7BPHN0QMMD68IUDKT8I.example.com.

- *web.example.com*: D65CIIG0GTRKQ26Q774DVMRCNHQO6F81.example.com.

- *www.example.com*: NTQ0CQEJHM0S17POMCUSLG5IOQQEDTBJ.example.com.

- *example.com*: TOM10UQBL336NFAQB3P6MOO53LSVG8UI.example.com.

We can enable NSEC3 opt-out with the following configuration, changing the `optout` configuration value from `no` to `yes`:

```
dnssec-policy "standard" {
    nsec3param iterations 10 optout yes salt-length 16;
};
```

After NSEC3 opt-out is enabled, the number of NSEC3 records is reduced. Notice that the unsigned delegations `aaa`, `ccc`, `ddd`, and `eee` no longer have corresponding NSEC3 records.

- *bbb.example.com*: AESO0NT3N44OOSDQS3PSL0HACHUE1O0U.example.com.

- *ftp.example.com*: JKMAVHL8V7EMCL8JHIEN8KBOAB0MGUK2.example.com.

- *ns1.example.com*: FSK5TK9964BNE7BPHN0QMMD68IUDKT8I.example.com.

- *web.example.com*: D65CIIG0GTRKQ26Q774DVMRCNHQO6F81.example.com.

- *www.example.com*: NTQ0CQEJHM0S17POMCUSLG5IOQQEDTBJ.example.com.

- *example.com*: TOM10UQBL336NFAQB3P6MOO53LSVG8UI.example.com.

To undo NSEC3 opt-out, change the configuration again:

```
dnssec-policy "standard" {
    nsec3param iterations 10 optout no salt-length 16;
};
```

Note: NSEC3 hashes the plain text domain name, and we can compute our own hashes using the tool `nsec3hash`. For example, to compute the hashed name for `www.example.com` using the parameters we listed above, we can execute this command:

```
# nsec3hash 1234567890ABCDEF 1 10 www.example.com.
NTQ0CQEJHM0S17POMCUSLG5IOQQEDTBJ (salt=1234567890ABCDEF, hash=1, iterations=10)
```

9.8.4 Reverting to Unsigned

This recipe describes how to revert from a signed zone (DNSSEC) back to an unsigned (DNS) zone.

Whether the world thinks your zone is signed is determined by the presence of DS records hosted by your parent zone; if there are no DS records, the world sees your zone as unsigned. So reverting to unsigned is as easy as removing all DS records from the parent zone.

Below is an example showing how to remove DS records using the GoDaddy web-based interface:

1. After logging in, click the green "Launch" button next to the domain name you want to manage.
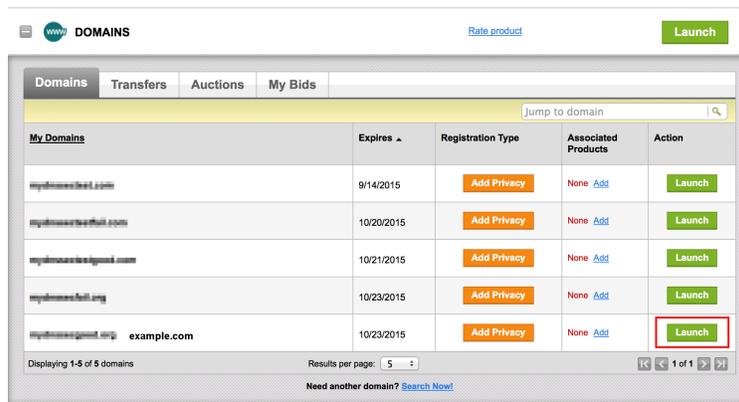
Fig. 16: Revert to Unsigned Step #1

2. Scroll down to the "DS Records" section and click Manage.

3. A dialog appears, displaying all current keys. Use the far right-hand X button to remove each key.

4. Click Save.

To be on the safe side, wait a while before actually deleting all signed data from your zone, just in case some validating resolvers have cached information. After you are certain that all cached information has expired (usually this means one TTL interval has passed), you may reconfigure your zone.

Here is what `named.conf` looks like when it is signed:

Fig. 17: Revert to Unsigned Step #2

Fig. 18: Revert to Unsigned Step #3
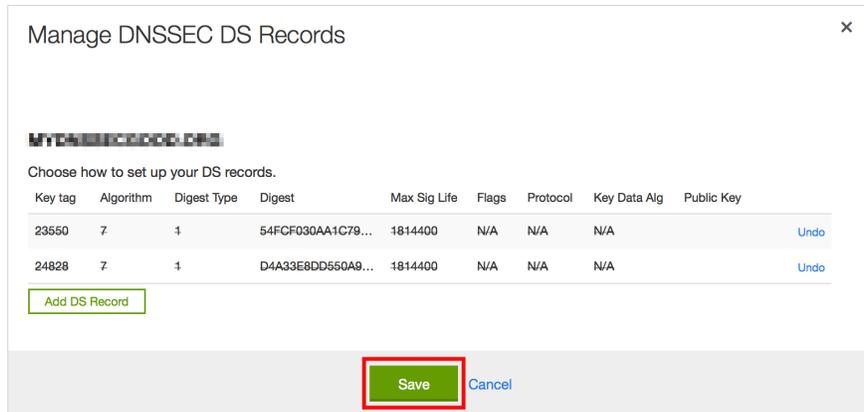
Fig. 19: Revert to Unsigned Step #4

```
zone "example.com" IN {
    type primary;
    file "db/example.com.db";
    allow-transfer { any; };
    dnssec-policy "default";
};
```

Change your `dnssec-policy` line to indicate you want to revert to unsigned:

```
zone "example.com" IN {
    type primary;
    file "db/example.com.db";
    allow-transfer { any; };
    dnssec-policy "insecure";
};
```

Then use `rndc reload` to reload the zone.

The "insecure" policy is a built-in policy (like "default"). It will make sure the zone is still DNSSEC maintained, to allow for a graceful transition to unsigned.

When the DS records have been removed from the parent zone, use `rndc dnssec -checkds -key <id> withdrawn example.com` to tell `named` that the DS is removed, and the remaining DNSSEC records will be removed in a timely manner. Or if you have parental agents configured, the DNSSEC records will be automatically removed after BIND has seen that the parental agents no longer serves the DS RRset for this zone.

After a while, your zone is reverted back to the traditional, insecure DNS format. You can verify by checking that all DNSKEY and RRSIG records have been removed from the zone.

You can then remove the `dnssec-policy` line from your `named.conf` and reload the zone. The zone will now no longer be subject to any DNSSEC maintenance.

9.9 Commonly Asked Questions

No questions are too stupid to ask. Below are some common questions you may have and (hopefully) some answers that help.

Do I need IPv6 to have DNSSEC? No. DNSSEC can be deployed without IPv6.

Does DNSSEC encrypt my DNS traffic, so others cannot eavesdrop on my DNS queries? No. Although cryptographic keys and digital signatures are used in DNSSEC, they only provide authenticity and integrity, not privacy. Someone who sniffs network traffic can still see all the DNS queries and answers in plain text; DNSSEC just makes it very difficult for the eavesdropper to alter or spoof the DNS responses.

Does DNSSEC protect the communication between my laptop and my name server? Unfortunately, not at the moment. DNSSEC is designed to protect the communication between end clients (laptop) and name servers; however, there are few applications or stub resolver libraries as of mid-2020 that take advantage of this capability. While enabling DNSSEC today does little to enhance the security of communications between a recursive server and its clients (commonly called the "last mile"), we hope that will change in the near future as more applications become DNSSEC-aware.

Does DNSSEC secure zone transfers? No. You should consider using TSIG to secure zone transfers among your name servers.

Does DNSSEC protect my network from malicious websites? The answer in the early stages of DNSSEC deployment is, unfortunately, no. DNSSEC is designed to provide confidence that when you receive a DNS response for www.company.com over port 53, it really came from Company's name servers and the answers are authentic. But that does not mean the web server a user visits over port 80 or port 443 is necessarily safe. Furthermore, 98.5% of domain name operators (as of this writing in mid-2020) have not yet signed their zones, so DNSSEC cannot even validate their answers.

The answer for sometime in the future is that, as more zones are signed and more recursive servers validate, DNSSEC will make it much more difficult for attackers to spoof DNS responses or perform cache poisoning. It will still not protect against users who visit a malicious website that an attacker owns and operates, or prevent users from mistyping a domain name; it will just become less likely that an attacker can hijack other domain names.

If I enable DNSSEC validation, will it break DNS lookup, since most domain names do not yet use DNSSEC?
No, DNSSEC is backwards-compatible to "standard" DNS. As of this writing (in mid-2020), although 98.5% of the .com domains have yet to be signed, a DNSSEC-enabled validating resolver can still look up all of these domain names as it always has under standard DNS.

There are four (4) categories of responses (see **RFC 4035**):

*Secure***:** Domains that have DNSSEC deployed correctly.

*Insecure***:** Domains that have yet to deploy DNSSEC.

*Bogus***:** Domains that have deployed DNSSEC but have done it incorrectly.

*Indeterminate***:** Domains for which it is not possible to determine whether these domains use DNSSEC.

A DNSSEC-enabled validating resolver still resolves #1 and #2; only #3 and #4 result in a SERVFAIL. You may already be using DNSSEC validation without realizing it, since some ISPs have begun enabling DNSSEC validation on their recursive name servers. Google public DNS (8.8.8.8) also has enabled DNSSEC validation.

Do I need to have special client software to use DNSSEC? No. DNSSEC only changes the communication behavior among DNS servers, not between a DNS server (validating resolver) and a client (stub resolver). With DNSSEC validation enabled on your recursive server, if a domain name does not pass the checks, an error message (typically SERVFAIL) is returned to clients; to most client software today, it appears that the DNS query has failed or that the domain name does not exist.

Since DNSSEC uses public key cryptography, do I need Public Key Infrastructure (PKI) in order to use DNSSEC?
No, DNSSEC does not depend on an existing PKI. Public keys are stored within the DNS hierarchy; the trustworthiness of each zone is guaranteed by its parent zone, all the way back to the root zone. A copy of the trust anchor for the root zone is distributed with BIND 9.

Do I need to purchase SSL certificates from a Certificate Authority (CA) to use DNSSEC? No. With DNSSEC, you generate and publish your own keys, and sign your own data as well. There is no need to pay someone else to do it for you.

My parent zone does not support DNSSEC; can I still sign my zone? Technically, yes, but you will not get the full benefit of DNSSEC, as other validating resolvers are not able to validate your zone data. Without the DS record(s) in your parent zone, other validating resolvers treat your zone as an insecure (traditional) zone, and no actual verification is carried out. To the rest of the world, your zone still appears to be insecure, and it will continue to be insecure until your parent zone can host the DS record(s) for you and tell the rest of the world that your zone is signed.

Is DNSSEC the same thing as TSIG? No. TSIG is typically used between primary and secondary name servers to secure zone transfers, while DNSSEC secures DNS lookup by validating answers. Even if you enable DNSSEC, zone transfers are still not validated; to secure the communication between your primary and secondary name servers, consider setting up TSIG or similar secure channels.

How are keys copied from primary to secondary server(s)? DNSSEC uses public cryptography, which results in two types of keys: public and private. The public keys are part of the zone data, stored as DNSKEY record types. Thus the public keys are synchronized from primary to secondary server(s) as part of the zone transfer. The private keys are not, and should not be, stored anywhere other than secured on the primary server. See *Key Storage* for more information on key storage options and considerations.

Can I use the same key for multiple zones? Yes and no. Good security practice suggests that you should use unique key pairs for each zone, just as you should have different passwords for your email account, social media login, and online banking credentials. On a technical level, it is completely feasible to reuse a key, but multiple zones are at risk if one key pair is compromised. However, if you have hundreds or thousands of zones to administer, a single key pair for all might be less error-prone to manage. You may choose to use the same approach as with password management: use unique passwords for your bank accounts and shopping sites, but use a standard password for your not-very-important logins. First, categorize your zones: high-value zones (or zones that have specific key rollover requirements) get their own key pairs, while other, more "generic" zones can use a single key pair for easier management. Note that at present (mid-2020), fully automatic signing (using the `dnssec-policy` clause in your `named` configuration file) does not support reuse of keys except when the same zone appears in multiple views (see next question). To use the same key for multiple zones, sign your zones using semi-automatic signing. Each zone wishing to use the key should point to the same key directory.

How do I sign the different instances of a zone that appears in multiple views? Add a `dnssec-policy` statement to each `zone` definition in the configuration file. To avoid problems when a single computer accesses different instances of the zone while information is still in its cache (e.g., a laptop moving from your office to a customer site), you should sign all instances with the same key. This means setting the same DNSSEC policy for all instances of the zone, and making sure that the key directory is the same for all instances of the zone.

Will there be any problems if I change the DNSSEC policy for a zone? If you are using fully automatic signing, no. Just change the parameters in the `dnssec-policy` statement and reload the configuration file. `named` makes a smooth transition to the new policy, ensuring that your zone remains valid at all times.

A BRIEF HISTORY OF THE DNS AND BIND

Although the Domain Name System "officially" began in 1984 with the publication of **RFC 920**, the core of the new system was described in 1983 in **RFC 882** and **RFC 883**. From 1984 to 1987, the ARPAnet (the precursor to today's Internet) became a testbed of experimentation for developing the new naming/addressing scheme in a rapidly expanding, operational network environment. New RFCs were written and published in 1987 that modified the original documents to incorporate improvements based on the working model. **RFC 1034**, "Domain Names-Concepts and Facilities," and **RFC 1035**, "Domain Names-Implementation and Specification," were published and became the standards upon which all DNS implementations are built.

The first working domain name server, called "Jeeves," was written in 1983-84 by Paul Mockapetris for operation on DEC Tops-20 machines located at the University of Southern California's Information Sciences Institute (USC-ISI) and SRI International's Network Information Center (SRI-NIC). A DNS server for Unix machines, the Berkeley Internet Name Domain (BIND) package, was written soon after by a group of graduate students at the University of California at Berkeley under a grant from the US Defense Advanced Research Projects Administration (DARPA).

Versions of BIND through 4.8.3 were maintained by the Computer Systems Research Group (CSRG) at UC Berkeley. Douglas Terry, Mark Painter, David Riggle, and Songnian Zhou made up the initial BIND project team. After that, additional work on the software package was done by Ralph Campbell. Kevin Dunlap, a Digital Equipment Corporation employee on loan to the CSRG, worked on BIND for 2 years, from 1985 to 1987. Many other people also contributed to BIND development during that time: Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss, Jim Bloom, and Mike Schwartz. BIND maintenance was subsequently handled by Mike Karels and Øivind Kure.

BIND versions 4.9 and 4.9.1 were released by Digital Equipment Corporation (which became Compaq Computer Corporation and eventually merged with Hewlett-Packard). Paul Vixie, then a DEC employee, became BIND's primary caretaker. He was assisted by Phil Almquist, Robert Elz, Alan Barrett, Paul Albitz, Bryan Beecher, Andrew Partan, Andy Cherenson, Tom Limoncelli, Berthold Paffrath, Fuat Baran, Anant Kumar, Art Harkin, Win Treese, Don Lewis, Christophe Wolfhugel, and others.

In 1994, BIND version 4.9.2 was sponsored by Vixie Enterprises. Paul Vixie became BIND's principal architect/programmer.

BIND versions from 4.9.3 onward have been developed and maintained by Internet Systems Consortium and its predecessor, the Internet Software Consortium, with support provided by ISC's sponsors.

As co-architects/programmers, Bob Halley and Paul Vixie released the first production-ready version of BIND version 8 in May 1997.

BIND version 9 was released in September 2000 and is a major rewrite of nearly all aspects of the underlying BIND architecture.

BIND versions 4 and 8 are officially deprecated. No additional development is done on BIND version 4 or BIND version 8.

BIND development work is made possible today by the sponsorship of corporations who purchase professional support services from ISC (https://www.isc.org/contact/) and/or donate to our mission, and by the tireless efforts of numerous individuals.

ELEVEN

GENERAL DNS REFERENCE INFORMATION

11.1 Requests for Comment (RFCs)

Specification documents for the Internet protocol suite, including the DNS, are published as part of the Request for Comments (RFCs) series of technical notes. The standards themselves are defined by the Internet Engineering Task Force (IETF) and the Internet Engineering Steering Group (IESG). RFCs can be viewed online at: https://www.rfc-editor.org/.

While reading RFCs, please keep in mind that **not all RFCs are standards**, and also that the validity of documents does change over time. Every RFC needs to be interpreted in the context of other documents.

BIND 9 strives for strict compliance with IETF standards. To the best of our knowledge, BIND 9 complies with the following RFCs, with the caveats and exceptions listed in the numbered notes below. Many of these RFCs were written by current or former ISC staff members. The list is non-exhaustive.

Some of these RFCs, though DNS-related, are not concerned with implementing software.

11.1.1 Protocol Specifications

RFC 1034 - P. Mockapetris. *Domain Names — Concepts and Facilities.* November 1987.

RFC 1035 - P. Mockapetris. *Domain Names — Implementation and Specification.* November 1987.[1][2]

RFC 1183 - C. F. Everhart, L. A. Mamakos, R. Ullmann, P. Mockapetris. *New DNS RR Definitions.* October 1990.

RFC 1706 - B. Manning and R. Colella. *DNS NSAP Resource Records.* October 1994.

RFC 1712 - C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni. *DNS Encoding of Geographical Location.* November 1994.

RFC 1876 - C. Davis, P. Vixie, T. Goodwin, and I. Dickinson. *A Means for Expressing Location Information in the Domain Name System.* January 1996.

RFC 1982 - R. Elz and R. Bush. *Serial Number Arithmetic.* August 1996.

RFC 1995 - M. Ohta. *Incremental Zone Transfer in DNS.* August 1996.

RFC 1996 - P. Vixie. *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY).* August 1996.

RFC 2136 - P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE).* April 1997.

RFC 2163 - A. Allocchio. *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM).* January 1998.

RFC 2181 - R. Elz and R. Bush. *Clarifications to the DNS Specification.* July 1997.

---

[1] Queries to zones that have failed to load return SERVFAIL rather than a non-authoritative response. This is considered a feature.

[2] CLASS ANY queries are not supported. This is considered a feature.

RFC 2230 - R. Atkinson. *Key Exchange Delegation Record for the DNS.* November 1997.

RFC 2308 - M. Andrews. *Negative Caching of DNS Queries (DNS NCACHE).* March 1998.

RFC 2539 - D. Eastlake, 3rd. *Storage of Diffie-Hellman Keys in the Domain Name System (DNS).* March 1999.

RFC 2782 - A. Gulbrandsen, P. Vixie, and L. Esibov. *A DNS RR for Specifying the Location of Services (DNS SRV).* February 2000.

RFC 2930 - D. Eastlake, 3rd. *Secret Key Establishment for DNS (TKEY RR).* September 2000.

RFC 2931 - D. Eastlake, 3rd. *DNS Request and Transaction Signatures (SIG(0)s).* September 2000.[3]

RFC 3007 - B. Wellington. *Secure Domain Name System (DNS) Dynamic Update.* November 2000.

RFC 3110 - D. Eastlake, 3rd. *RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS).* May 2001.

RFC 3123 - P. Koch. *A DNS RR Type for Lists of Address Prefixes (APL RR).* June 2001.

RFC 3225 - D. Conrad. *Indicating Resolver Support of DNSSEC.* December 2001.

RFC 3226 - O. Gudmundsson. *DNSSEC and IPv6 A6 Aware Server/Resolver Message Size Requirements.* December 2001.

RFC 3363 - R. Bush, A. Durand, B. Fink, O. Gudmundsson, and T. Hain. *Representing Internet Protocol Version 6 (IPv6) Addresses in the Domain Name System (DNS).* August 2002.[15]

RFC 3403 - M. Mealling. *Dynamic Delegation Discovery System (DDDS). Part Three: The Domain Name System (DNS) Database.* October 2002.

RFC 3492 - A. Costello. *Punycode: A Bootstring Encoding of Unicode for Internationalized Domain Names in Applications (IDNA).* March 2003.

RFC 3493 - R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens. *Basic Socket Interface Extensions for IPv6.* March 2003.

RFC 3496 - A. G. Malis and T. Hsiao. *Protocol Extension for Support of Asynchronous Transfer Mode (ATM) Service Class-aware Multiprotocol Label Switching (MPLS) Traffic Engineering.* March 2003.

RFC 3596 - S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. *DNS Extensions to Support IP Version 6.* October 2003.

RFC 3597 - A. Gustafsson. *Handling of Unknown DNS Resource Record (RR) Types.* September 2003.

RFC 3645 - S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, and R. Hall. *Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG).* October 2003.

RFC 4025 - M. Richardson. *A Method for Storing IPsec Keying Material in DNS.* March 2005.

RFC 4033 - R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *DNS Security Introduction and Requirements.* March 2005.

RFC 4034 - R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *Resource Records for the DNS Security Extensions.* March 2005.

RFC 4035 - R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *Protocol Modifications for the DNS Security Extensions.* March 2005.

RFC 4255 - J. Schlyter and W. Griffin. *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints.* January 2006.

RFC 4343 - D. Eastlake, 3rd. *Domain Name System (DNS) Case Insensitivity Clarification.* January 2006.

RFC 4398 - S. Josefsson. *Storing Certificates in the Domain Name System (DNS).* March 2006.

---

[3] When receiving a query signed with a SIG(0), the server is only able to verify the signature if it has the key in its local authoritative data; it cannot do recursion or validation to retrieve unknown keys.

[15] Section 4 is ignored.

RFC 4470 - S. Weiler and J. Ihren. *Minimally covering NSEC Records and DNSSEC On-line Signing.* April 2006.[6]

RFC 4509 - W. Hardaker. *Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs).* May 2006.

RFC 4592 - E. Lewis. *The Role of Wildcards in the Domain Name System.* July 2006.

RFC 4635 - D. Eastlake, 3rd. *HMAC SHA (Hashed Message Authentication Code, Secure Hash Algorithm) TSIG Algorithm Identifiers.* August 2006.

RFC 4701 - M. Stapp, T. Lemon, and A. Gustafsson. *A DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR).* October 2006.

RFC 4955 - D. Blacka. *DNS Security (DNSSEC) Experiments.* July 2007.[7]

RFC 5001 - R. Austein. *DNS Name Server Identifier (NSID) Option.* August 2007.

RFC 5011 - M. StJohns. *Automated Updates of DNS Security (DNSSEC) Trust Anchors.*

RFC 5155 - B. Laurie, G. Sisson, R. Arends, and D. Blacka. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence.* March 2008.

RFC 5205 - P. Nikander and J. Laganier. *Host Identity Protocol (HIP) Domain Name System (DNS) Extension.* April 2008.

RFC 5452 - A. Hubert and R. van Mook. *Measures for Making DNS More Resilient Against Forged Answers.* January 2009.[8]

RFC 5702 - J. Jansen. *Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC.* October 2009.

RFC 5891 - J. Klensin. *Internationalized Domain Names in Applications (IDNA): Protocol.* August 2010

RFC 5936 - E. Lewis and A. Hoenes, Ed. *DNS Zone Transfer Protocol (AXFR).* June 2010.

RFC 5952 - S. Kawamura and M. Kawashima. *A Recommendation for IPv6 Address Text Representation.* August 2010.

RFC 6052 - C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li. *IPv6 Addressing of IPv4/IPv6 Translators.* October 2010.

RFC 6147 - M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum. *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers.* April 2011.[9]

RFC 6604 - D. Eastlake, 3rd. *xNAME RCODE and Status Bits Clarification.* April 2012.

RFC 6605 - P. Hoffman and W. C. A. Wijngaards. *Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC.* April 2012.[10]

RFC 6672 - S. Rose and W. Wijngaards. *DNAME Redirection in the DNS.* June 2012.

RFC 6698 - P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA.* August 2012.

RFC 6725 - S. Rose. *DNS Security (DNSSEC) DNSKEY Algorithm IANA Registry Updates.* August 2012.[11]

RFC 6742 - RJ Atkinson, SN Bhatti, U. St. Andrews, and S. Rose. *DNS Resource Records for the Identifier-Locator Network Protocol (ILNP).* November 2012.

---

[6] Minimally Covering NSEC records are accepted but not generated.

[7] BIND 9 interoperates with correctly designed experiments.

[8] `named` only uses ports to extend the ID space; addresses are not used.

[9] Section 5.5 does not match reality. `named` uses the presence of DO=1 to detect if validation may be occurring. CD has no bearing on whether validation occurs.

[10] Compliance is conditional on the OpenSSL library being linked against a supporting ECDSA.

[11] RSAMD5 support has been removed. See **RFC 8624**.

RFC 6840 - S. Weiler, Ed., and D. Blacka, Ed. *Clarifications and Implementation Notes for DNS Security (DNSSEC).* February 2013.[12]

RFC 6891 - J. Damas, M. Graff, and P. Vixie. *Extension Mechanisms for DNS (EDNS(0)).* April 2013.

RFC 7043 - J. Abley. *Resource Records for EUI-48 and EUI-64 Addresses in the DNS.* October 2013.

RFC 7208 - S. Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1.* April 2014.

RFC 7314 - M. Andrews. *Extension Mechanisms for DNS (EDNS) EXPIRE Option.* July 2014.

RFC 7344 - W. Kumari, O. Gudmundsson, and G. Barwood. *Automating DNSSEC Delegation Trust Maintenance.* September 2014.[13]

RFC 7477 - W. Hardaker. *Child-to-Parent Synchronization in DNS.* March 2015.

RFC 7553 - P. Faltstrom and O. Kolkman. *The Uniform Resource Identifier (URI) DNS Resource Record.* June 2015.

RFC 7583 - S. Morris, J. Ihren, J. Dickinson, and W. Mekking. *DNSSEC Key Rollover Timing Considerations.* October 2015.

RFC 7766 - J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. *DNS Transport over TCP - Implementation Requirements.* March 2016.

RFC 7828 - P. Wouters, J. Abley, S. Dickinson, and R. Bellis. *The edns-tcp-keepalive EDNS0 Option.* April 2016.

RFC 7830 - A. Mayrhofer. *The EDNS(0) Padding Option.* May 2016.[14]

RFC 7929 - P. Wouters. *DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP.* August 2016.

RFC 8078 - O. Gudmundsson and P. Wouters. *Managing DS Records from the Parent via CDS/CDNSKEY.* March 2017.[20]

RFC 8080 - O. Sury and R. Edmonds. *Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC.* February 2017.

RFC 8624 - P. Wouters and O. Sury. *Algorithm Implementation Requirements and Usage Guidance for DNSSEC.* June 2019.

RFC 8659 - P. Hallam-Baker, R. Stradling, and J. Hoffman-Andrews. *DNS Certification Authority Authorization (CAA) Resource Record.* November 2019.

RFC 8945 - F. Dupont, S. Morris, P. Vixie, D. Eastlake 3rd, O. Gudmundsson, and B. Wellington. *Secret Key Transaction Authentication for DNS (TSIG).* November 2020.

11.1.2 Best Current Practice RFCs

RFC 2219 - M. Hamilton and R. Wright. *Use of DNS Aliases for Network Services.* October 1997.

RFC 2317 - H. Eidnes, G. de Groot, and P. Vixie. *Classless IN-ADDR.ARPA Delegation.* March 1998.

RFC 2606 - D. Eastlake, 3rd and A. Panitz. *Reserved Top Level DNS Names.* June 1999.[16]

RFC 3901 - A. Durand and J. Ihren. *DNS IPv6 Transport Operational Guidelines.* September 2004.

RFC 5625 - R. Bellis. *DNS Proxy Implementation Guidelines.* August 2009.

RFC 6303 - M. Andrews. *Locally Served DNS Zones.* July 2011.

---

[12] Section 5.9 - Always set CD=1 on queries. This is *not* done, as it prevents DNSSEC from working correctly through another recursive server. When talking to a recursive server, the best algorithm is to send CD=0 and then send CD=1 iff SERVFAIL is returned, in case the recursive server has a bad clock and/or bad trust anchor. Alternatively, one can send CD=1 then CD=0 on validation failure, in case the recursive server is under attack or there is stale/bogus authoritative data.

[13] Updating of parent zones is not yet implemented.

[14] `named` does not currently encrypt DNS requests, so the PAD option is accepted but not returned in responses.

[20] Updating of parent zones is not yet implemented.

[16] This does not apply to DNS server implementations.

RFC 7793 - M. Andrews. *Adding 100.64.0.0/10 Prefixes to the IPv4 Locally-Served DNS Zones Registry.* May 2016.

RFC 8906 - M. Andrews and R. Bellis. *A Common Operational Problem in DNS Servers: Failure to Communicate.* September 2020.

11.1.3 For Your Information

RFC 1101 - P. Mockapetris. *DNS Encoding of Network Names and Other Types.* April 1989.

RFC 1123 - R. Braden. *Requirements for Internet Hosts - Application and Support.* October 1989.

RFC 1535 - E. Gavron. *A Security Problem and Proposed Correction With Widely Deployed DNS Software.* October 1993.

RFC 1536 - A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller. *Common DNS Implementation Errors and Suggested Fixes.* October 1993.

RFC 1912 - D. Barr. *Common DNS Operational and Configuration Errors.* February 1996.

RFC 2874 - M. Crawford and C. Huitema. *DNS Extensions to Support IPv6 Address Aggregation and Renumbering.* July 2000.[4]

RFC 3833 - D. Atkins and R. Austein. *Threat Analysis of the Domain Name System (DNS).* August 2004.

RFC 4074 - Y. Morishita and T. Jinmei. *Common Misbehavior Against DNS Queries for IPv6 Addresses.* June 2005.

RFC 4431 - M. Andrews and S. Weiler. *The DNSSEC Lookaside Validation (DLV) DNS Resource Record.* February 2006.[5]

RFC 4892 - S. Woolf and D. Conrad. *Requirements for a Mechanism Identifying a Name Server Instance.* June 2007.

RFC 6781 - O. Kolkman, W. Mekking, and R. Gieben. *DNSSEC Operational Practices, Version 2.* December 2012.

RFC 7129 - R. Gieben and W. Mekking. *Authenticated Denial of Existence in the DNS.* February 2014.

RFC 8749 - W. Mekking and D. Mahoney. *Moving DNSSEC Lookaside Validation (DLV) to Historic Status.* March 2020.

11.2 Notes

11.3 Internet Drafts

Internet Drafts (IDs) are rough-draft working documents of the Internet Engineering Task Force (IETF). They are, in essence, RFCs in the preliminary stages of development. Implementors are cautioned not to regard IDs as archival, and they should not be quoted or cited in any formal documents unless accompanied by the disclaimer that they are "works in progress." IDs have a lifespan of six months, after which they are deleted unless updated by their authors.

---

[4] Compliance is with loading and serving of A6 records only. A6 records were moved to the experimental category by **RFC 3363**.
[5] Compliance is with loading and serving of DLV records only. DLV records were moved to the historic category by **RFC 8749**.

MANUAL PAGES

12.1 arpaname - translate IP addresses to the corresponding ARPA names

12.1.1 Synopsis

arpaname {*ipaddress …*}

12.1.2 Description

arpaname translates IP addresses (IPv4 and IPv6) to the corresponding IN-ADDR.ARPA or IP6.ARPA names.

12.1.3 See Also

BIND 9 Administrator Reference Manual.

12.2 ddns-confgen - TSIG key generation tool

12.2.1 Synopsis

ddns-confgen [**-a** algorithm] [**-h**] [**-k** keyname] [**-q**] [**-s** name] [**-z** zone]

12.2.2 Description

ddns-confgen is an utility that generates keys for use in TSIG signing. The resulting keys can be used, for example, to secure dynamic DNS updates to a zone, or for the rndc command channel.

The key name can specified using -k parameter and defaults to ddns-key. The generated key is accompanied by configuration text and instructions that can be used with nsupdate and named when setting up dynamic DNS, including an example update-policy statement. (This usage is similar to the rndc-confgen command for setting up command-channel security.)

Note that named itself can configure a local DDNS key for use with nsupdate -l; it does this when a zone is configured with update-policy local;. ddns-confgen is only needed when a more elaborate configuration is required: for instance, if nsupdate is to be used from a remote system.

12.2.3 Options

-a algorithm This option specifies the algorithm to use for the TSIG key. Available choices are: hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, and hmac-sha512. The default is hmac-sha256. Options are case-insensitive, and the "hmac-" prefix may be omitted.

-h This option prints a short summary of options and arguments.

-k keyname This option specifies the key name of the DDNS authentication key. The default is `ddns-key` when neither the `-s` nor `-z` option is specified; otherwise, the default is `ddns-key` as a separate label followed by the argument of the option, e.g., `ddns-key.example.com`. The key name must have the format of a valid domain name, consisting of letters, digits, hyphens, and periods.

-q This option enables quiet mode, which prints only the key, with no explanatory text or usage examples. This is essentially identical to `tsig-keygen`.

-s name This option generates a configuration example to allow dynamic updates of a single hostname. The example `named.conf` text shows how to set an update policy for the specified name using the "name" nametype. The default key name is `ddns-key.name`. Note that the "self" nametype cannot be used, since the name to be updated may differ from the key name. This option cannot be used with the `-z` option.

-z zone This option generates a configuration example to allow dynamic updates of a zone. The example `named.conf` text shows how to set an update policy for the specified zone using the "zonesub" nametype, allowing updates to all subdomain names within that zone. This option cannot be used with the `-s` option.

12.2.4 See Also

nsupchange(1), *named.conf(5)*, *named(8)*, BIND 9 Administrator Reference Manual.

12.3 delv - DNS lookup and validation utility

12.3.1 Synopsis

delv [@server] [[**-4**] | [**-6**]] [**-a** anchor-file] [**-b** address] [**-c** class] [**-d** level] [**-i**] [**-m**] [**-p** port#] [**-q** name] [**-t** type] [**-x** addr] [name] [type] [class] [queryopt…]

delv [**-h**]

delv [**-v**]

delv [queryopt…] [query…]

12.3.2 Description

`delv` is a tool for sending DNS queries and validating the results, using the same internal resolver and validator logic as `named`.

`delv` sends to a specified name server all queries needed to fetch and validate the requested data; this includes the original requested query, subsequent queries to follow CNAME or DNAME chains, queries for DNSKEY, and DS records to establish a chain of trust for DNSSEC validation. It does not perform iterative resolution, but simulates the behavior of a name server configured for DNSSEC validating and forwarding.

By default, responses are validated using the built-in DNSSEC trust anchor for the root zone ("."). Records returned by `delv` are either fully validated or were not signed. If validation fails, an explanation of the failure is included in the output; the validation process can be traced in detail. Because `delv` does not rely on an external server to carry out

validation, it can be used to check the validity of DNS responses in environments where local name servers may not be trustworthy.

Unless it is told to query a specific name server, `delv` tries each of the servers listed in `/etc/resolv.conf`. If no usable server addresses are found, `delv` sends queries to the localhost addresses (127.0.0.1 for IPv4, ::1 for IPv6).

When no command-line arguments or options are given, `delv` performs an NS query for "." (the root zone).

12.3.3 Simple Usage

A typical invocation of `delv` looks like:

```
delv @server name type
```

where:

server is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied `server` argument is a hostname, `delv` resolves that name before querying that name server (note, however, that this initial lookup is *not* validated by DNSSEC).

If no `server` argument is provided, `delv` consults `/etc/resolv.conf`; if an address is found there, it queries the name server at that address. If either of the `-4` or `-6` options is in use, then only addresses for the corresponding transport are tried. If no usable addresses are found, `delv` sends queries to the localhost addresses (127.0.0.1 for IPv4, ::1 for IPv6).

name is the domain name to be looked up.

type indicates what type of query is required - ANY, A, MX, etc. `type` can be any valid query type. If no `type` argument is supplied, `delv` performs a lookup for an A record.

12.3.4 Options

-a anchor-file This option specifies a file from which to read DNSSEC trust anchors. The default is `/etc/bind.keys`, which is included with BIND 9 and contains one or more trust anchors for the root zone (".").

Keys that do not match the root zone name are ignored. An alternate key name can be specified using the `+root=NAME` options.

Note: When reading the trust anchor file, `delv` treats `trust-anchors`, `initial-key`, and `static-key` identically. That is, for a managed key, it is the *initial* key that is trusted; **RFC 5011** key management is not supported. `delv` does not consult the managed-keys database maintained by `named`, which means that if either of the keys in `/etc/bind.keys` is revoked and rolled over, `/etc/bind.keys` must be updated to use DNSSEC validation in `delv`.

-b address This option sets the source IP address of the query to `address`. This must be a valid address on one of the host's network interfaces, or `0.0.0.0`, or `::`. An optional source port may be specified by appending `#<port>`

-c class This option sets the query class for the requested data. Currently, only class "IN" is supported in `delv` and any other value is ignored.

-d level This option sets the systemwide debug level to `level`. The allowed range is from 0 to 99. The default is 0 (no debugging). Debugging traces from `delv` become more verbose as the debug level increases. See the `+mtrace`, `+rtrace`, and `+vtrace` options below for additional debugging details.

-h This option displays the `delv` help usage output and exits.

−i This option sets insecure mode, which disables internal DNSSEC validation. (Note, however, that this does not set the CD bit on upstream queries. If the server being queried is performing DNSSEC validation, then it does not return invalid data; this can cause `delv` to time out. When it is necessary to examine invalid data to debug a DNSSEC problem, use `dig +cd`.)

−m This option enables memory usage debugging.

−p port# This option specifies a destination port to use for queries, instead of the standard DNS port number 53. This option is used with a name server that has been configured to listen for queries on a non-standard port number.

−q name This option sets the query name to `name`. While the query name can be specified without using the `−q` option, it is sometimes necessary to disambiguate names from types or classes (for example, when looking up the name "ns", which could be misinterpreted as the type NS, or "ch", which could be misinterpreted as class CH).

−t type This option sets the query type to `type`, which can be any valid query type supported in BIND 9 except for zone transfer types AXFR and IXFR. As with `−q`, this is useful to distinguish query-name types or classes when they are ambiguous. It is sometimes necessary to disambiguate names from types.

The default query type is "A", unless the `−x` option is supplied to indicate a reverse lookup, in which case it is "PTR".

−v This option prints the `delv` version and exits.

−x addr This option performs a reverse lookup, mapping an address to a name. `addr` is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. When `−x` is used, there is no need to provide the `name` or `type` arguments; `delv` automatically performs a lookup for a name like `11.12.13.10.in-addr.arpa` and sets the query type to PTR. IPv6 addresses are looked up using nibble format under the IP6.ARPA domain.

−4 This option forces `delv` to only use IPv4.

−6 This option forces `delv` to only use IPv6.

12.3.5 Query Options

`delv` provides a number of query options which affect the way results are displayed, and in some cases the way lookups are performed.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option. These may be preceded by the string `no` to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form `+keyword=value`. The query options are:

+[no]cdflag This option controls whether to set the CD (checking disabled) bit in queries sent by `delv`. This may be useful when troubleshooting DNSSEC problems from behind a validating resolver. A validating resolver blocks invalid responses, making it difficult to retrieve them for analysis. Setting the CD flag on queries causes the resolver to return invalid responses, which `delv` can then validate internally and report the errors in detail.

+[no]class This option controls whether to display the CLASS when printing a record. The default is to display the CLASS.

+[no]ttl This option controls whether to display the TTL when printing a record. The default is to display the TTL.

+[no]rtrace This option toggles resolver fetch logging. This reports the name and type of each query sent by `delv` in the process of carrying out the resolution and validation process, including the original query and all subsequent queries to follow CNAMEs and to establish a chain of trust for DNSSEC validation.

This is equivalent to setting the debug level to 1 in the "resolver" logging category. Setting the systemwide debug level to 1 using the `−d` option produces the same output, but affects other logging categories as well.

+[no]mtrace This option toggles message logging. This produces a detailed dump of the responses received by `delv` in the process of carrying out the resolution and validation process.

This is equivalent to setting the debug level to 10 for the "packets" module of the "resolver" logging category. Setting the systemwide debug level to 10 using the `-d` option produces the same output, but affects other logging categories as well.

+[no]vtrace This option toggles validation logging. This shows the internal process of the validator as it determines whether an answer is validly signed, unsigned, or invalid.

This is equivalent to setting the debug level to 3 for the "validator" module of the "dnssec" logging category. Setting the systemwide debug level to 3 using the `-d` option produces the same output, but affects other logging categories as well.

+[no]short This option toggles between verbose and terse answers. The default is to print the answer in a verbose form.

+[no]comments This option toggles the display of comment lines in the output. The default is to print comments.

+[no]rrcomments This option toggles the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is to print per-record comments.

+[no]crypto This option toggles the display of cryptographic fields in DNSSEC records. The contents of these fields are unnecessary to debug most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted, they are replaced by the string `[omitted]` or, in the DNSKEY case, the key ID is displayed as the replacement, e.g. `[ key id = value ]`.

+[no]trust This option controls whether to display the trust level when printing a record. The default is to display the trust level.

+[no]split[=W] This option splits long hex- or base64-formatted fields in resource records into chunks of `W` characters (where `W` is rounded up to the nearest multiple of 4). `+nosplit` or `+split=0` causes fields not to be split at all. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]all This option sets or clears the display options `+[no]comments`, `+[no]rrcomments`, and `+[no]trust` as a group.

+[no]multiline This option prints long records (such as RRSIG, DNSKEY, and SOA records) in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the `delv` output.

+[no]dnssec This option indicates whether to display RRSIG records in the `delv` output. The default is to do so. Note that (unlike in `dig`) this does *not* control whether to request DNSSEC records or to validate them. DNSSEC records are always requested, and validation always occurs unless suppressed by the use of `-i` or `+noroot`.

+[no]root[=ROOT] This option indicates whether to perform conventional DNSSEC validation, and if so, specifies the name of a trust anchor. The default is to validate using a trust anchor of "." (the root zone), for which there is a built-in key. If specifying a different trust anchor, then `-a` must be used to specify a file containing the key.

+[no]tcp This option controls whether to use TCP when sending queries. The default is to use UDP unless a truncated response has been received.

+[no]unknownformat This option prints all RDATA in unknown RR-type presentation format (**RFC 3597**). The default is to print RDATA for known types in the type's presentation format.

+[no]yaml This option prints response data in YAML format.

12.3.6 Files

`/etc/bind.keys`

`/etc/resolv.conf`

12.3.7 See Also

dig(1), *named(8)*, **RFC 4034**, **RFC 4035**, **RFC 4431**, **RFC 5074**, **RFC 5155**.

12.4 dig - DNS lookup utility

12.4.1 Synopsis

dig [@server] [**-b** address] [**-c** class] [**-f** filename] [**-k** filename] [**-m**] [**-p** port#] [**-q** name] [**-t** type] [**-v**] [**-x** addr] [**-y** [hmac:]name:key] [[**-4**] | [**-6**]] [name] [type] [class] [queryopt…]

dig [**-h**]

dig [global-queryopt…] [query…]

12.4.2 Description

dig is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried. Most DNS administrators use dig to troubleshoot DNS problems because of its flexibility, ease of use, and clarity of output. Other lookup tools tend to have less functionality than dig.

Although dig is normally used with command-line arguments, it also has a batch mode of operation for reading lookup requests from a file. A brief summary of its command-line arguments and options is printed when the -h option is given. The BIND 9 implementation of dig allows multiple lookups to be issued from the command line.

Unless it is told to query a specific name server, dig tries each of the servers listed in /etc/resolv.conf. If no usable server addresses are found, dig sends the query to the local host.

When no command-line arguments or options are given, dig performs an NS query for "." (the root).

It is possible to set per-user defaults for dig via ${HOME}/.digrc. This file is read and any options in it are applied before the command-line arguments. The -r option disables this feature, for scripts that need predictable behavior.

The IN and CH class names overlap with the IN and CH top-level domain names. Either use the -t and -c options to specify the type and class, use the -q to specify the domain name, or use "IN." and "CH." when looking up these top-level domains.

12.4.3 Simple Usage

A typical invocation of dig looks like:

```
dig @server name type
```

where:

server is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied `server` argument is a hostname, `dig` resolves that name before querying that name server.

If no `server` argument is provided, `dig` consults `/etc/resolv.conf`; if an address is found there, it queries the name server at that address. If either of the `-4` or `-6` options are in use, then only addresses for the corresponding transport are tried. If no usable addresses are found, `dig` sends the query to the local host. The reply from the name server that responds is displayed.

name is the name of the resource record that is to be looked up.

type indicates what type of query is required - ANY, A, MX, SIG, etc. `type` can be any valid query type. If no `type` argument is supplied, `dig` performs a lookup for an A record.

12.4.4 Options

-4 This option indicates that only IPv4 should be used.

-6 This option indicates that only IPv6 should be used.

-b address[#port] This option sets the source IP address of the query. The `address` must be a valid address on one of the host's network interfaces, or "0.0.0.0" or "::". An optional port may be specified by appending `#port`.

-c class This option sets the query class. The default `class` is IN; other classes are HS for Hesiod records or CH for Chaosnet records.

-f file This option sets batch mode, in which `dig` reads a list of lookup requests to process from the given `file`. Each line in the file should be organized in the same way it would be presented as a query to `dig` using the command-line interface.

-k keyfile This option tells `named` to sign queries using TSIG using a key read from the given file. Key files can be generated using `tsig-keygen`. When using TSIG authentication with `dig`, the name server that is queried needs to know the key and algorithm that is being used. In BIND, this is done by providing appropriate `key` and `server` statements in `named.conf`.

-m This option enables memory usage debugging.

-p port This option sends the query to a non-standard port on the server, instead of the default port 53. This option is used to test a name server that has been configured to listen for queries on a non-standard port number.

-q name This option specifies the domain name to query. This is useful to distinguish the `name` from other arguments.

-r This option indicates that options from `${HOME}/.digrc` should not be read. This is useful for scripts that need predictable behavior.

-t type This option indicates the resource record type to query, which can be any valid query type. If it is a resource record type supported in BIND 9, it can be given by the type mnemonic (such as `NS` or `AAAA`). The default query type is `A`, unless the `-x` option is supplied to indicate a reverse lookup. A zone transfer can be requested by specifying a type of AXFR. When an incremental zone transfer (IXFR) is required, set the `type` to `ixfr=N`. The incremental zone transfer contains all changes made to the zone since the serial number in the zone's SOA record was `N`.

All resource record types can be expressed as `TYPEnn`, where `nn` is the number of the type. If the resource record type is not supported in BIND 9, the result is displayed as described in **RFC 3597**.

-u This option indicates that print query times should be provided in microseconds instead of milliseconds.

-v This option prints the version number and exits.

-x addr This option sets simplified reverse lookups, for mapping addresses to names. The `addr` is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. When the `-x` option is used, there is no need to

provide the name, class, and type arguments. dig automatically performs a lookup for a name like 94.
2.0.192.in-addr.arpa and sets the query type and class to PTR and IN respectively. IPv6 addresses are
looked up using nibble format under the IP6.ARPA domain.

-y [hmac:]keyname:secret This option signs queries using TSIG with the given authentication key. key-
name is the name of the key, and secret is the base64-encoded shared secret. hmac is the name of the key
algorithm; valid choices are hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384,
or hmac-sha512. If hmac is not specified, the default is hmac-md5; if MD5 was disabled, the default is
hmac-sha256.

Note: Only the -k option should be used, rather than the -y option, because with -y the shared secret is supplied as a
command-line argument in clear text. This may be visible in the output from ps1 or in a history file maintained by the
user's shell.

12.4.5 Query Options

dig provides a number of query options which affect the way in which lookups are made and the results displayed. Some
of these set or reset flag bits in the query header, some determine which sections of the answer get printed, and others
determine the timeout and retry strategies.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option; these
may be preceded by the string no to negate the meaning of that keyword. Other keywords assign values to options, like
the timeout interval. They have the form +keyword=value. Keywords may be abbreviated, provided the abbreviation
is unambiguous; for example, +cd is equivalent to +cdflag. The query options are:

+[no]aaflag This option is a synonym for +[no]aaonly.

+[no]aaonly This option sets the aa flag in the query.

+[no]additional This option displays [or does not display] the additional section of a reply. The default is to display
it.

+[no]adflag This option sets [or does not set] the AD (authentic data) bit in the query. This requests the server
to return whether all of the answer and authority sections have been validated as secure, according to the security
policy of the server. AD=1 indicates that all records have been validated as secure and the answer is not from a
OPT-OUT range. AD=0 indicates that some part of the answer was insecure or not validated. This bit is set by
default.

+[no]all This option sets or clears all display flags.

+[no]answer This option displays [or does not display] the answer section of a reply. The default is to display it.

+[no]authority This option displays [or does not display] the authority section of a reply. The default is to display
it.

+[no]badcookie This option retries the lookup with a new server cookie if a BADCOOKIE response is received.

+[no]besteffort This option attempts to display the contents of messages which are malformed. The default is to
not display malformed answers.

+bufsize[=B] This option sets the UDP message buffer size advertised using EDNS0 to B bytes. The maximum and
minimum sizes of this buffer are 65535 and 0, respectively. +bufsize=0 disables EDNS (use +bufsize=0
+edns to send an EDNS message with an advertised size of 0 bytes). +bufsize restores the default buffer size.

+[no]cdflag This option sets [or does not set] the CD (checking disabled) bit in the query. This requests the server
to not perform DNSSEC validation of responses.

+[no]class This option displays [or does not display] the CLASS when printing the record.

+[no]cmd This option toggles the printing of the initial comment in the output, identifying the version of `dig` and the query options that have been applied. This option always has a global effect; it cannot be set globally and then overridden on a per-lookup basis. The default is to print this comment.

+[no]comments This option toggles the display of some comment lines in the output, with information about the packet header and OPT pseudosection, and the names of the response section. The default is to print these comments.

Other types of comments in the output are not affected by this option, but can be controlled using other command-line switches. These include +[no]cmd, +[no]question, +[no]stats, and +[no]rrcomments.

+[no]cookie=#### This option sends [or does not send] a COOKIE EDNS option, with an optional value. Replaying a COOKIE from a previous response allows the server to identify a previous client. The default is +cookie.

+cookie is also set when +trace is set to better emulate the default queries from a nameserver.

+[no]crypto This option toggles the display of cryptographic fields in DNSSEC records. The contents of these fields are unnecessary for debugging most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted, they are replaced by the string [omitted] or, in the DNSKEY case, the key ID is displayed as the replacement, e.g. [key id = value].

+[no]defname This option, which is deprecated, is treated as a synonym for +[no]search.

+[no]dnssec This option requests that DNSSEC records be sent by setting the DNSSEC OK (DO) bit in the OPT record in the additional section of the query.

+domain=somename This option sets the search list to contain the single domain somename, as if specified in a domain directive in /etc/resolv.conf, and enables search list processing as if the +search option were given.

+dscp=value This option sets the DSCP code point to be used when sending the query. Valid DSCP code points are in the range [0…63]. By default no code point is explicitly set.

+[no]edns[=#] This option specifies the EDNS version to query with. Valid values are 0 to 255. Setting the EDNS version causes an EDNS query to be sent. +noedns clears the remembered EDNS version. EDNS is set to 0 by default.

+[no]ednsflags[=#] This option sets the must-be-zero EDNS flags bits (Z bits) to the specified value. Decimal, hex, and octal encodings are accepted. Setting a named flag (e.g., DO) is silently ignored. By default, no Z bits are set.

+[no]ednsnegotiation This option enables/disables EDNS version negotiation. By default, EDNS version negotiation is enabled.

+[no]ednsopt[=code[:value]] This option specifies the EDNS option with code point code and an optional payload of value as a hexadecimal string. code can be either an EDNS option name (for example, NSID or ECS) or an arbitrary numeric value. +noednsopt clears the EDNS options to be sent.

+[no]expire This option sends an EDNS Expire option.

+[no]fail This option indicates that named should try [or not try] the next server if a SERVFAIL is received. The default is to not try the next server, which is the reverse of normal stub resolver behavior.

+[no]header-only This option sends a query with a DNS header without a question section. The default is to add a question section. The query type and query name are ignored when this is set.

+[no]identify This option shows [or does not show] the IP address and port number that supplied the answer, when the +short option is enabled. If short form answers are requested, the default is not to show the source address and port number of the server that provided the answer.

+[no]idnin This option processes [or does not process] IDN domain names on input. This requires IDN SUPPORT to have been enabled at compile time.

The default is to process IDN input when standard output is a tty. The IDN processing on input is disabled when `dig` output is redirected to files, pipes, and other non-tty file descriptors.

+[no]idnout This option converts [or does not convert] puny code on output. This requires `IDN SUPPORT` to have been enabled at compile time.

The default is to process puny code on output when standard output is a tty. The puny code processing on output is disabled when `dig` output is redirected to files, pipes, and other non-tty file descriptors.

+[no]ignore This option ignores [or does not ignore] truncation in UDP responses instead of retrying with TCP. By default, TCP retries are performed.

+[no]keepalive This option sends [or does not send] an EDNS Keepalive option.

+[no]keepopen This option keeps [or does not keep] the TCP socket open between queries, and reuses it rather than creating a new TCP socket for each lookup. The default is `+nokeepopen`.

+[no]mapped This option allows [or does not allow] mapped IPv4-over-IPv6 addresses to be used. The default is `+mapped`.

+[no]multiline This option prints [or does not print] records, like the SOA records, in a verbose multi-line format with human-readable comments. The default is to print each record on a single line to facilitate machine parsing of the `dig` output.

+ndots=D This option sets the number of dots (`D`) that must appear in `name` for it to be considered absolute. The default value is that defined using the `ndots` statement in `/etc/resolv.conf`, or 1 if no `ndots` statement is present. Names with fewer dots are interpreted as relative names, and are searched for in the domains listed in the `search` or `domain` directive in `/etc/resolv.conf` if `+search` is set.

+[no]nsid When enabled, this option includes an EDNS name server ID request when sending a query.

+[no]nssearch When this option is set, `dig` attempts to find the authoritative name servers for the zone containing the name being looked up, and display the SOA record that each name server has for the zone. Addresses of servers that did not respond are also printed.

+[no]onesoa When enabled, this option prints only one (starting) SOA record when performing an AXFR. The default is to print both the starting and ending SOA records.

+[no]opcode=value When enabled, this option sets (restores) the DNS message opcode to the specified value. The default value is QUERY (0).

+padding=value This option pads the size of the query packet using the EDNS Padding option to blocks of `value` bytes. For example, `+padding=32` causes a 48-byte query to be padded to 64 bytes. The default block size is 0, which disables padding; the maximum is 512. Values are ordinarily expected to be powers of two, such as 128; however, this is not mandatory. Responses to padded queries may also be padded, but only if the query uses TCP or DNS COOKIE.

+[no]qr This option toggles the display of the query message as it is sent. By default, the query is not printed.

+[no]question This option toggles the display of the question section of a query when an answer is returned. The default is to print the question section as a comment.

+[no]raflag This option sets [or does not set] the RA (Recursion Available) bit in the query. The default is `+no-raflag`. This bit is ignored by the server for QUERY.

+[no]rdflag This option is a synonym for `+[no]recurse`.

+[no]recurse This option toggles the setting of the RD (recursion desired) bit in the query. This bit is set by default, which means `dig` normally sends recursive queries. Recursion is automatically disabled when the `+nssearch` or `+trace` query option is used.

+retry=T This option sets the number of times to retry UDP and TCP queries to server to `T` instead of the default, 2. Unlike `+tries`, this does not include the initial query.

+[no]rrcomments This option toggles the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is not to print record comments unless multiline mode is active.

+[no]search This option uses [or does not use] the search list defined by the searchlist or domain directive in resolv.conf, if any. The search list is not used by default.

ndots from resolv.conf (default 1), which may be overridden by +ndots, determines whether the name is treated as relative and hence whether a search is eventually performed.

+[no]short This option toggles whether a terse answer is provided. The default is to print the answer in a verbose form. This option always has a global effect; it cannot be set globally and then overridden on a per-lookup basis.

+[no]showsearch This option performs [or does not perform] a search showing intermediate results.

+[no]sigchase This feature is now obsolete and has been removed; use delv instead.

+split=W This option splits long hex- or base64-formatted fields in resource records into chunks of W characters (where W is rounded up to the nearest multiple of 4). +nosplit or +split=0 causes fields not to be split at all. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]stats This option toggles the printing of statistics: when the query was made, the size of the reply, etc. The default behavior is to print the query statistics as a comment after each lookup.

+[no]subnet=addr[/prefix-length] This option sends [or does not send] an EDNS CLIENT-SUBNET option with the specified IP address or network prefix.

dig +subnet=0.0.0.0/0, or simply dig +subnet=0 for short, sends an EDNS CLIENT-SUBNET option with an empty address and a source prefix-length of zero, which signals a resolver that the client's address information must *not* be used when resolving this query.

+[no]tcflag This option sets [or does not set] the TC (TrunCation) bit in the query. The default is +notcflag. This bit is ignored by the server for QUERY.

+[no]tcp This option uses [or does not use] TCP when querying name servers. The default behavior is to use UDP unless a type any or ixfr=N query is requested, in which case the default is TCP. AXFR queries always use TCP.

+timeout=T This option sets the timeout for a query to T seconds. The default timeout is 5 seconds. An attempt to set T to less than 1 is silently set to 1.

+[no]topdown This feature is related to dig +sigchase, which is obsolete and has been removed. Use delv instead.

+[no]trace This option toggles tracing of the delegation path from the root name servers for the name being looked up. Tracing is disabled by default. When tracing is enabled, dig makes iterative queries to resolve the name being looked up. It follows referrals from the root servers, showing the answer from each server that was used to resolve the lookup.

If @server is also specified, it affects only the initial query for the root zone name servers.

+dnssec is also set when +trace is set, to better emulate the default queries from a name server.

+tries=T This option sets the number of times to try UDP and TCP queries to server to T instead of the default, 3. If T is less than or equal to zero, the number of tries is silently rounded up to 1.

+trusted-key=#### This option formerly specified trusted keys for use with dig +sigchase. This feature is now obsolete and has been removed; use delv instead.

+[no]ttlid This option displays [or does not display] the TTL when printing the record.

+[no]ttlunits This option displays [or does not display] the TTL in friendly human-readable time units of s, m, h, d, and w, representing seconds, minutes, hours, days, and weeks. This implies +ttlid.

+[no]unexpected This option accepts [or does not accept] answers from unexpected sources. By default, `dig` will not accept a reply from a source other than the one to which it sent the query.

+[no]unknownformat This option prints all RDATA in unknown RR type presentation format ([RFC 3597](#)). The default is to print RDATA for known types in the type's presentation format.

+[no]vc This option uses [or does not use] TCP when querying name servers. This alternate syntax to `+[no]tcp` is provided for backwards compatibility. The `vc` stands for "virtual circuit."

+[no]yaml When enabled, this option prints the responses (and, if `+qr` is in use, also the outgoing queries) in a detailed YAML format.

+[no]zflag This option sets [or does not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

12.4.6 Multiple Queries

The BIND 9 implementation of `dig` supports specifying multiple queries on the command line (in addition to supporting the `-f` batch file option). Each of those queries can be supplied with its own set of flags, options, and query options.

In this case, each `query` argument represents an individual query in the command-line syntax described above. Each consists of any of the standard options and flags, the name to be looked up, an optional query type and class, and any query options that should be applied to that query.

A global set of query options, which should be applied to all queries, can also be supplied. These global query options must precede the first tuple of name, class, type, options, flags, and query options supplied on the command line. Any global query options (except `+[no]cmd` and `+[no]short` options) can be overridden by a query-specific set of query options. For example:

```
dig +qr www.isc.org any -x 127.0.0.1 isc.org ns +noqr
```

shows how `dig` can be used from the command line to make three lookups: an ANY query for `www.isc.org`, a reverse lookup of 127.0.0.1, and a query for the NS records of `isc.org`. A global query option of `+qr` is applied, so that `dig` shows the initial query it made for each lookup. The final query has a local query option of `+noqr` which means that `dig` does not print the initial query when it looks up the NS records for `isc.org`.

12.4.7 IDN Support

If `dig` has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. `dig` appropriately converts character encoding of a domain name before sending a request to a DNS server or displaying a reply from the server. To turn off IDN support, use the parameters `+noidnin` and `+noidnout`, or define the `IDN_DISABLE` environment variable.

12.4.8 Return Codes

`dig` return codes are:

0 DNS response received, including NXDOMAIN status

1 Usage error

8 Couldn't open batch file

9 No reply from server

10 Internal error

12.4.9 Files

```
/etc/resolv.conf
```

```
${HOME}/.digrc
```

12.4.10 See Also

delv(1), *host(1)*, *named(8)*, *dnssec-keygen(8)*, **RFC 1035**.

12.4.11 Bugs

There are probably too many query options.

12.5 dnssec-cds - change DS records for a child zone based on CDS/CDNSKEY

12.5.1 Synopsis

dnssec-cds [**-a** alg…] [**-c** class] [**-D**] {**-d** dsset-file} {**-f** child-file} [**-i****[**extension**]] [-s** start-time] [**-T** ttl] [**-u**] [**-v** level] [**-V**] {domain}

12.5.2 Description

The `dnssec-cds` command changes DS records at a delegation point based on CDS or CDNSKEY records published in the child zone. If both CDS and CDNSKEY records are present in the child zone, the CDS is preferred. This enables a child zone to inform its parent of upcoming changes to its key-signing keys (KSKs); by polling periodically with `dnssec-cds`, the parent can keep the DS records up-to-date and enable automatic rolling of KSKs.

Two input files are required. The `-f child-file` option specifies a file containing the child's CDS and/or CDNSKEY records, plus RRSIG and DNSKEY records so that they can be authenticated. The `-d path` option specifies the location of a file containing the current DS records. For example, this could be a `dsset-` file generated by `dnssec-signzone`, or the output of `dnssec-dsfromkey`, or the output of a previous run of `dnssec-cds`.

The `dnssec-cds` command uses special DNSSEC validation logic specified by **RFC 7344**. It requires that the CDS and/or CDNSKEY records be validly signed by a key represented in the existing DS records. This is typically the pre-existing KSK.

For protection against replay attacks, the signatures on the child records must not be older than they were on a previous run of `dnssec-cds`. Their age is obtained from the modification time of the `dsset-` file, or from the `-s` option.

To protect against breaking the delegation, `dnssec-cds` ensures that the DNSKEY RRset can be verified by every key algorithm in the new DS RRset, and that the same set of keys are covered by every DS digest type.

By default, replacement DS records are written to the standard output; with the `-i` option the input file is overwritten in place. The replacement DS records are the same as the existing records, when no change is required. The output can be empty if the CDS/CDNSKEY records specify that the child zone wants to be insecure.

> **Warning:** Be careful not to delete the DS records when `dnssec-cds` fails!

Alternatively, `dnssec-cds -u` writes an `nsupdate` script to the standard output. The `-u` and `-i` options can be used together to maintain a `dsset-` file as well as emit an `nsupdate` script.

12.5.3 Options

-a algorithm This option specifies a digest algorithm to use when converting CDNSKEY records to DS records. This option can be repeated, so that multiple DS records are created for each CDNSKEY record. This option has no effect when using CDS records.

> The algorithm must be one of SHA-1, SHA-256, or SHA-384. These values are case-insensitive, and the hyphen may be omitted. If no algorithm is specified, the default is SHA-256.

-c class This option specifies the DNS class of the zones.

-D This option generates DS records from CDNSKEY records if both CDS and CDNSKEY records are present in the child zone. By default CDS records are preferred.

-d path This specifies the location of the parent DS records. The path can be the name of a file containing the DS records; if it is a directory, `dnssec-cds` looks for a `dsset-` file for the domain inside the directory.

> To protect against replay attacks, child records are rejected if they were signed earlier than the modification time of the `dsset-` file. This can be adjusted with the `-s` option.

-f child-file This option specifies the file containing the child's CDS and/or CDNSKEY records, plus its DNSKEY records and the covering RRSIG records, so that they can be authenticated.

> The examples below describe how to generate this file.

-iextension This option updates the `dsset-` file in place, instead of writing DS records to the standard output.

> There must be no space between the `-i` and the extension. If no extension is provided, the old `dsset-` is discarded. If an extension is present, a backup of the old `dsset-` file is kept with the extension appended to its filename.

> To protect against replay attacks, the modification time of the `dsset-` file is set to match the signature inception time of the child records, provided that it is later than the file's current modification time.

-s start-time This option specifies the date and time after which RRSIG records become acceptable. This can be either an absolute or a relative time. An absolute start time is indicated by a number in YYYYMMDDHHMMSS notation; 20170827133700 denotes 13:37:00 UTC on August 27th, 2017. A time relative to the `dsset-` file is indicated with `-N`, which is N seconds before the file modification time. A time relative to the current time is indicated with `now+N`.

> If no start-time is specified, the modification time of the `dsset-` file is used.

-T ttl This option specifies a TTL to be used for new DS records. If not specified, the default is the TTL of the old DS records. If they had no explicit TTL, the new DS records also have no explicit TTL.

-u This option writes an `nsupdate` script to the standard output, instead of printing the new DS records. The output is empty if no change is needed.

> Note: The TTL of new records needs to be specified: it can be done in the original `dsset-` file, with the `-T` option, or using the `nsupdate ttl` command.

-V This option prints version information.

-v level This option sets the debugging level. Level 1 is intended to be usefully verbose for general users; higher levels are intended for developers.

domain This indicates the name of the delegation point/child zone apex.

12.5.4 Exit Status

The `dnssec-cds` command exits 0 on success, or non-zero if an error occurred.

If successful, the DS records may or may not need to be changed.

12.5.5 Examples

Before running `dnssec-signzone`, ensure that the delegations are up-to-date by running `dnssec-cds` on every `dsset-` file.

To fetch the child records required by `dnssec-cds`, invoke `dig` as in the script below. It is acceptable if the `dig` fails, since `dnssec-cds` performs all the necessary checking.

```
for f in dsset-*
do
    d=${f#dsset-}
    dig +dnssec +noall +answer $d DNSKEY $d CDNSKEY $d CDS |
    dnssec-cds -i -f /dev/stdin -d $f $d
done
```

When the parent zone is automatically signed by `named`, `dnssec-cds` can be used with `nsupdate` to maintain a delegation as follows. The `dsset-` file allows the script to avoid having to fetch and validate the parent DS records, and it maintains the replay attack protection time.

```
dig +dnssec +noall +answer $d DNSKEY $d CDNSKEY $d CDS |
dnssec-cds -u -i -f /dev/stdin -d $f $d |
nsupdate -l
```

12.5.6 See Also

dig(1), *dnssec-settime(8)*, *dnssec-signzone(8)*, *nsupdate(1)*, BIND 9 Administrator Reference Manual, **RFC 7344**.

12.6 dnssec-dsfromkey - DNSSEC DS RR generation tool

12.6.1 Synopsis

dnssec-dsfromkey [**-1** | **-2** | **-a** alg] [**-C**] [**-T** TTL] [**-v** level] [**-K** directory] {keyfile}

dnssec-dsfromkey [**-1** | **-2** | **-a** alg] [**-C**] [**-T** TTL] [**-v** level] [**-c** class] [**-A**] {**-f** file} [dnsname]

dnssec-dsfromkey [**-1** | **-2** | **-a** alg] [**-C**] [**-T** TTL] [**-v** level] [**-c** class] [**-K** directory] {**-s**} {dnsname}

dnssec-dsfromkey [**-h** | **-V**]

12.6.2 Description

The `dnssec-dsfromkey` command outputs DS (Delegation Signer) resource records (RRs), or CDS (Child DS) RRs with the `-C` option.

By default, only KSKs are converted (keys with flags = 257). The `-A` option includes ZSKs (flags = 256). Revoked keys are never included.

The input keys can be specified in a number of ways:

By default, `dnssec-dsfromkey` reads a key file named in the format `Knnnn.+aaa+iiiii.key`, as generated by `dnssec-keygen`.

With the `-f file` option, `dnssec-dsfromkey` reads keys from a zone file or partial zone file (which can contain just the DNSKEY records).

With the `-s` option, `dnssec-dsfromkey` reads a `keyset-` file, as generated by `dnssec-keygen -C`.

12.6.3 Options

-1 This option is an abbreviation for `-a SHA1`.

-2 This option is an abbreviation for `-a SHA-256`.

-a algorithm This option specifies a digest algorithm to use when converting DNSKEY records to DS records. This option can be repeated, so that multiple DS records are created for each DNSKEY record.

   The algorithm must be one of SHA-1, SHA-256, or SHA-384. These values are case-insensitive, and the hyphen may be omitted. If no algorithm is specified, the default is SHA-256.

-A This option indicates that ZSKs are to be included when generating DS records. Without this option, only keys which have the KSK flag set are converted to DS records and printed. This option is only useful in `-f` zone file mode.

-c class This option specifies the DNS class; the default is IN. This option is only useful in `-s` keyset or `-f` zone file mode.

-C This option generates CDS records rather than DS records.

-f file This option sets zone file mode, in which the final dnsname argument of `dnssec-dsfromkey` is the DNS domain name of a zone whose master file can be read from `file`. If the zone name is the same as `file`, then it may be omitted.

   If `file` is `-`, then the zone data is read from the standard input. This makes it possible to use the output of the `dig` command as input, as in:

   `dig dnskey example.com | dnssec-dsfromkey -f - example.com`

-h This option prints usage information.

-K directory This option tells BIND 9 to look for key files or `keyset-` files in `directory`.

-s This option enables keyset mode, in which the final dnsname argument from `dnssec-dsfromkey` is the DNS domain name used to locate a `keyset-` file.

-T TTL This option specifies the TTL of the DS records. By default the TTL is omitted.

-v level This option sets the debugging level.

-V This option prints version information.

12.6.4 Example

To build the SHA-256 DS RR from the `Kexample.com.+003+26160` keyfile, issue the following command:

`dnssec-dsfromkey -2 Kexample.com.+003+26160`

The command returns something similar to:

`example.com. IN DS 26160 5 2 3A1EADA7A74B8D0BA86726B0C227AA85AB8BBD2B2004F41A868A54F0C5EA0B9`

12.6.5 Files

The keyfile can be designated by the key identification `Knnnn.+aaa+iiiii` or the full file name `Knnnn.+aaa+iiiii.key`, as generated by `dnssec-keygen`.

The keyset file name is built from the `directory`, the string `keyset-`, and the `dnsname`.

12.6.6 Caveat

A keyfile error may return "file not found," even if the file exists.

12.6.7 See Also

dnssec-keygen(8), *dnssec-signzone(8)*, BIND 9 Administrator Reference Manual, RFC 3658 (DS RRs), RFC 4509 (SHA-256 for DS RRs), RFC 6605 (SHA-384 for DS RRs), RFC 7344 (CDS and CDNSKEY RRs).

12.7 dnssec-importkey - import DNSKEY records from external systems so they can be managed

12.7.1 Synopsis

dnssec-importkey [**-K** directory] [**-L** ttl] [**-P** date/offset] [**-P** sync date/offset] [**-D** date/offset] [**-D** sync date/offset] [**-h**] [**-v** level] [**-V**] {keyfile}

dnssec-importkey {**-f** filename} [**-K** directory] [**-L** ttl] [**-P** date/offset] [**-P** sync date/offset] [**-D** date/offset] [**-D** sync date/offset] [**-h**] [**-v** level] [**-V**] [dnsname]

12.7.2 Description

`dnssec-importkey` reads a public DNSKEY record and generates a pair of .key/.private files. The DNSKEY record may be read from an existing .key file, in which case a corresponding .private file is generated, or it may be read from any other file or from the standard input, in which case both .key and .private files are generated.

The newly created .private file does *not* contain private key data, and cannot be used for signing. However, having a .private file makes it possible to set publication (`-P`) and deletion (`-D`) times for the key, which means the public key can be added to and removed from the DNSKEY RRset on schedule even if the true private key is stored offline.

12.7.3 Options

–f filename This option indicates the zone file mode. Instead of a public keyfile name, the argument is the DNS domain name of a zone master file, which can be read from `filename`. If the domain name is the same as `filename`, then it may be omitted.

If `filename` is set to `"-"`, then the zone data is read from the standard input.

–K directory This option sets the directory in which the key files are to reside.

–L ttl This option sets the default TTL to use for this key when it is converted into a DNSKEY RR. This is the TTL used when the key is imported into a zone, unless there was already a DNSKEY RRset in place, in which case the existing TTL takes precedence. Setting the default TTL to `0` or `none` removes it from the key.

–h This option emits a usage message and exits.

–v level This option sets the debugging level.

–V This option prints version information.

12.7.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a + or −, it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use `none` or `never`.

–P date/offset This option sets the date on which a key is to be published to the zone. After that date, the key is included in the zone but is not used to sign it.

–P sync date/offset This option sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

–D date/offset This option sets the date on which the key is to be deleted. After that date, the key is no longer included in the zone. (However, it may remain in the key repository.)

–D sync date/offset This option sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

12.7.5 Files

A keyfile can be designed by the key identification `Knnnn.+aaa+iiiii` or the full file name `Knnnn.+aaa+iiiii.key`, as generated by `dnssec-keygen`.

12.7.6 See Also

dnssec-keygen(8), *dnssec-signzone(8)*, BIND 9 Administrator Reference Manual, RFC 5011.

12.8 dnssec-checkds - DNSSEC delegation consistency checking tool

12.8.1 Synopsis

`dnssec-checkds` [**-d***dig path*] [**-D***dsfromkey path*] [**-f***file*] [**-l***domain*] [**-s***file*] {zone}

12.8.2 Description

`dnssec-checkds` verifies the correctness of Delegation Signer (DS) resource records for keys in a specified zone.

12.8.3 Options

-a *algorithm*

> Specify a digest algorithm to use when converting the zones DNSKEY records to expected DS records. This option can be repeated, so that multiple records are checked for each DNSKEY record.

> The *algorithm* must be one of SHA-1, SHA-256, or SHA-384. These values are case insensitive, and the hyphen may be omitted. If no algorithm is specified, the default is SHA-256.

-f *file*

> If a `file` is specified, then the zone is read from that file to find the DNSKEY records. If not, then the DNSKEY records for the zone are looked up in the DNS.

-s *file*

> Specifies a prepared dsset file, such as would be generated by `dnssec-signzone`, to use as a source for the DS RRset instead of querying the parent.

-d *dig path*

> Specifies a path to a `dig` binary. Used for testing.

-D *dsfromkey path*

> Specifies a path to a `dnssec-dsfromkey` binary. Used for testing.

12.8.4 See Also

`dnssec-dsfromkey`(8), `dnssec-keygen`(8), `dnssec-signzone`(8),

12.9 dnssec-coverage - checks future DNSKEY coverage for a zone

12.9.1 Synopsis

`dnssec-coverage` [**-K***directory*] [**-l***length*] [**-f***file*] [**-d***DNSKEY TTL*] [**-m***max TTL*] [**-r***interval*] [**-c***compilezone path*] [**-k**] [**-z**] [zone...]

12.9.2 Description

`dnssec-coverage` verifies that the DNSSEC keys for a given zone or a set of zones have timing metadata set properly to ensure no future lapses in DNSSEC coverage.

If `zone` is specified, then keys found in the key repository matching that zone are scanned, and an ordered list is generated of the events scheduled for that key (i.e., publication, activation, inactivation, deletion). The list of events is walked in order of occurrence. Warnings are generated if any event is scheduled which could cause the zone to enter a state in which validation failures might occur: for example, if the number of published or active keys for a given algorithm drops to zero, or if a key is deleted from the zone too soon after a new key is rolled, and cached data signed by the prior key has not had time to expire from resolver caches.

If `zone` is not specified, then all keys in the key repository will be scanned, and all zones for which there are keys will be analyzed. (Note: This method of reporting is only accurate if all the zones that have keys in a given repository share the same TTL parameters.)

12.9.3 Options

-K *directory*

> Sets the directory in which keys can be found. Defaults to the current working directory.

-f *file*

> If a `file` is specified, then the zone is read from that file; the largest TTL and the DNSKEY TTL are determined directly from the zone data, and the `-m` and `-d` options do not need to be specified on the command line.

-l *duration*

> The length of time to check for DNSSEC coverage. Key events scheduled further into the future than `duration` will be ignored, and assumed to be correct.

> The value of `duration` can be set in seconds, or in larger units of time by adding a suffix: mi for minutes, h for hours, d for days, w for weeks, mo for months, y for years.

-m *maximum TTL*

> Sets the value to be used as the maximum TTL for the zone or zones being analyzed when determining whether there is a possibility of validation failure. When a zone-signing key is deactivated, there must be enough time for the record in the zone with the longest TTL to have expired from resolver caches before that key can be purged from the DNSKEY RRset. If that condition does not apply, a warning will be generated.

> The length of the TTL can be set in seconds, or in larger units of time by adding a suffix: mi for minutes, h for hours, d for days, w for weeks, mo for months, y for years.

> This option is not necessary if the `-f` has been used to specify a zone file. If `-f` has been specified, this option may still be used; it will override the value found in the file.

> If this option is not used and the maximum TTL cannot be retrieved from a zone file, a warning is generated and a default value of 1 week is used.

-d *DNSKEY TTL*

> Sets the value to be used as the DNSKEY TTL for the zone or zones being analyzed when determining whether there is a possibility of validation failure. When a key is rolled (that is, replaced with a new key), there must be enough time for the old DNSKEY RRset to have expired from resolver caches before the new key is activated and begins generating signatures. If that condition does not apply, a warning will be generated.

The length of the TTL can be set in seconds, or in larger units of time by adding a suffix: mi for minutes, h for hours, d for days, w for weeks, mo for months, y for years.

This option is not necessary if `-f` has been used to specify a zone file from which the TTL of the DNSKEY RRset can be read, or if a default key TTL was set using ith the `-L` to `dnssec-keygen`. If either of those is true, this option may still be used; it will override the values found in the zone file or the key file.

If this option is not used and the key TTL cannot be retrieved from the zone file or the key file, then a warning is generated and a default value of 1 day is used.

-r *resign interval*

Sets the value to be used as the resign interval for the zone or zones being analyzed when determining whether there is a possibility of validation failure. This value defaults to 22.5 days, which is also the default in `named`. However, if it has been changed by the `sig-validity-interval` option in named.conf, then it should also be changed here.

The length of the interval can be set in seconds, or in larger units of time by adding a suffix: mi for minutes, h for hours, d for days, w for weeks, mo for months, y for years.

-k

Only check KSK coverage; ignore ZSK events. Cannot be used with `-z`.

-z

Only check ZSK coverage; ignore KSK events. Cannot be used with `-k`.

-c *compilezone path*

Specifies a path to a `named-compilezone` binary. Used for testing.

12.9.4 See Also

`dnssec-checkds`(8), `dnssec-dsfromkey`(8), `dnssec-keygen`(8), `dnssec-signzone`(8)

12.10 dnssec-keymgr - Ensures correct DNSKEY coverage based on a defined policy

12.10.1 Synopsis

dnssec-keymgr [**-K***directory*] [**-c***file*] [**-f**] [**-k**] [**-q**] [**-v**] [**-z**] [**-g***path*] [**-s***path*] [zone…]

12.10.2 Description

`dnssec-keymgr` is a high level Python wrapper to facilitate the key rollover process for zones handled by BIND. It uses the BIND commands for manipulating DNSSEC key metadata: `dnssec-keygen` and `dnssec-settime`.

DNSSEC policy can be read from a configuration file (default /etc/dnssec-policy.conf), from which the key parameters, publication and rollover schedule, and desired coverage duration for any given zone can be determined. This file may be used to define individual DNSSEC policies on a per-zone basis, or to set a "default" policy used for all zones.

When `dnssec-keymgr` runs, it examines the DNSSEC keys for one or more zones, comparing their timing metadata against the policies for those zones. If key settings do not conform to the DNSSEC policy (for example, because the policy has been changed), they are automatically corrected.

A zone policy can specify a duration for which we want to ensure the key correctness (`coverage`). It can also specify a rollover period (`roll-period`). If policy indicates that a key should roll over before the coverage period ends, then a successor key will automatically be created and added to the end of the key series.

If zones are specified on the command line, `dnssec-keymgr` will examine only those zones. If a specified zone does not already have keys in place, then keys will be generated for it according to policy.

If zones are *not* specified on the command line, then `dnssec-keymgr` will search the key directory (either the current working directory or the directory set by the `-K` option), and check the keys for all the zones represented in the directory.

Key times that are in the past will not be updated unless the `-f` is used (see below). Key inactivation and deletion times that are less than five minutes in the future will be delayed by five minutes.

It is expected that this tool will be run automatically and unattended (for example, by `cron`).

12.10.3 Options

-c *file*

> If `-c` is specified, then the DNSSEC policy is read from `file`. (If not specified, then the policy is read from /etc/dnssec-policy.conf; if that file doesnt exist, a built-in global default policy is used.)

-f

> Force: allow updating of key events even if they are already in the past. This is not recommended for use with zones in which keys have already been published. However, if a set of keys has been generated all of which have publication and activation dates in the past, but the keys have not been published in a zone as yet, then this option can be used to clean them up and turn them into a proper series of keys with appropriate rollover intervals.

-g *keygen-path*

> Specifies a path to a `dnssec-keygen` binary. Used for testing. See also the `-s` option.

-h

> Print the `dnssec-keymgr` help summary and exit.

-K *directory*

> Sets the directory in which keys can be found. Defaults to the current working directory.

-k

> Only apply policies to KSK keys. See also the `-z` option.

-q

> Quiet: suppress printing of `dnssec-keygen` and `dnssec-settime`.

-s *settime-path*

> Specifies a path to a `dnssec-settime` binary. Used for testing. See also the `-g` option.

-v

> Print the `dnssec-keymgr` version and exit.

-z

> Only apply policies to ZSK keys. See also the `-k` option.

12.10.4 Policy Configuration

The dnssec-policy.conf file can specify three kinds of policies:

· *Policy classes* (`policy`*name*`{ ... };`) can be inherited by zone policies or other policy classes; these can be used to create sets of different security profiles. For example, a policy class `normal` might specify 1024-bit key sizes, but a class `extra` might specify 2048 bits instead; `extra` would be used for zones that had unusually high security needs.

· *Algorithm policies:* (`algorithm-policy`*algorithm*`{ ... };`) override default per-algorithm settings. For example, by default, RSASHA256 keys use 2048-bit key sizes for both KSK and ZSK. This can be modified using `algorithm-policy`, and the new key sizes would then be used for any key of type RSASHA256.

· *Zone policies:* (`zone`*name*`{ ... };`) set policy for a single zone by name. A zone policy can inherit a policy class by including a `policy` option. Zone names beginning with digits (i.e., 0-9) must be quoted. If a zone does not have its own policy then the "default" policy applies.

Options that can be specified in policies:

`algorithm` *name*;

The key algorithm. If no policy is defined, the default is RSASHA256.

`coverage` *duration*;

The length of time to ensure that keys will be correct; no action will be taken to create new keys to be activated after this time. This can be represented as a number of seconds, or as a duration using human-readable units (examples: "1y" or "6 months"). A default value for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is six months.

`directory` *path*;

Specifies the directory in which keys should be stored.

`key-size` *keytype size*;

Specifies the number of bits to use in creating keys. The keytype is either "zsk" or "ksk". A default value for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is 2048 bits for RSA keys.

`keyttl` *duration*;

The key TTL. If no policy is defined, the default is one hour.

`post-publish` *keytype duration*;

How long after inactivation a key should be deleted from the zone. Note: If `roll-period` is not set, this value is ignored. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. The default is one month.

`pre-publish` *keytype duration*;

How long before activation a key should be published. Note: If `roll-period` is not set, this value is ignored. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. The default is one month.

`roll-period` *keytype duration*;

How frequently keys should be rolled over. The keytype is either "zsk" or "ksk". A default duration for this option can be set in algorithm policies as well as in policy classes or zone policies. If no policy is configured, the default is one year for ZSKs. KSKs do not roll over by default.

`standby` *keytype number*;

Not yet implemented.

12.10.5 Remaining Work

· Enable scheduling of KSK rollovers using the `-P sync` and `-D sync` options to `dnssec-keygen` and `dnssec-settime`. Check the parent zone (as in `dnssec-checkds`) to determine when its safe for the key to roll.

· Allow configuration of standby keys and use of the REVOKE bit, for keys that use RFC 5011 semantics.

12.10.6 See Also

`dnssec-coverage`(8), `dnssec-keygen`(8), `dnssec-settime`(8), `dnssec-checkds`(8)

12.11 dnssec-keyfromlabel - DNSSEC key generation tool

12.11.1 Synopsis

dnssec-keyfromlabel {**-l** label} [**-3**] [**-a** algorithm] [**-A** date/offset] [**-c** class] [**-D** date/offset] [**-D** sync date/offset] [**-E** engine] [**-f** flag] [**-G**] [**-I** date/offset] [**-i** interval] [**-k**] [**-K** directory] [**-L** ttl] [**-n** nametype] [**-P** date/offset] [**-P** sync date/offset] [**-p** protocol] [**-R** date/offset] [**-S** key] [**-t** type] [**-v** level] [**-V**] [**-y**] {name}

12.11.2 Description

`dnssec-keyfromlabel` generates a pair of key files that reference a key object stored in a cryptographic hardware service module (HSM). The private key file can be used for DNSSEC signing of zone data as if it were a conventional signing key created by `dnssec-keygen`, but the key material is stored within the HSM and the actual signing takes place there.

The `name` of the key is specified on the command line. This must match the name of the zone for which the key is being generated.

12.11.3 Options

-a algorithm This option selects the cryptographic algorithm. The value of `algorithm` must be one of RSASHA1, NSEC3RSASHA1, RSASHA256, RSASHA512, ECDSAP256SHA256, ECDSAP384SHA384, ED25519, or ED448.

If no algorithm is specified, RSASHA1 is used by default unless the `-3` option is specified, in which case NSEC3RSASHA1 is used instead. (If `-3` is used and an algorithm is specified, that algorithm is checked for compatibility with NSEC3.)

These values are case-insensitive. In some cases, abbreviations are supported, such as ECDSA256 for ECDSAP256SHA256 and ECDSA384 for ECDSAP384SHA384. If RSASHA1 is specified along with the `-3` option, then NSEC3RSASHA1 is used instead.

Since BIND 9.12.0, this option is mandatory except when using the `-S` option, which copies the algorithm from the predecessory key. Previously, the default for newly generated keys was RSASHA1.

-3 This option uses an NSEC3-capable algorithm to generate a DNSSEC key. If this option is used with an algorithm that has both NSEC and NSEC3 versions, then the NSEC3 version is used; for example, `dnssec-keygen -3a RSASHA1` specifies the NSEC3RSASHA1 algorithm.

-E engine This option specifies the cryptographic hardware to use.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

-l label This option specifies the label for a key pair in the crypto hardware.

When BIND 9 is built with OpenSSL-based PKCS#11 support, the label is an arbitrary string that identifies a particular key. It may be preceded by an optional OpenSSL engine name, followed by a colon, as in `pkcs11:keylabel`.

When BIND 9 is built with native PKCS#11 support, the label is a PKCS#11 URI string in the format `pkcs11:keyword\ =value[;\ keyword\ =value;...]`. Keywords include `token`, which identifies the HSM; `object`, which identifies the key; and `pin-source`, which identifies a file from which the HSM's PIN code can be obtained. The label is stored in the on-disk `private` file.

If the label contains a `pin-source` field, tools using the generated key files are able to use the HSM for signing and other operations without any need for an operator to manually enter a PIN. Note: Making the HSM's PIN accessible in this manner may reduce the security advantage of using an HSM; use caution with this feature.

-n nametype This option specifies the owner type of the key. The value of `nametype` must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user (KEY)), or OTHER (DNSKEY). These values are case-insensitive.

-C This option enables compatibility mode, which generates an old-style key, without any metadata. By default, `dnssec-keyfromlabel` includes the key's creation date in the metadata stored with the private key; other dates may be set there as well, including publication date, activation date, etc. Keys that include this data may be incompatible with older versions of BIND; the `-C` option suppresses them.

-c class This option indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-f flag This option sets the specified flag in the `flag` field of the KEY/DNSKEY record. The only recognized flags are KSK (Key-Signing Key) and REVOKE.

-G This option generates a key, but does not publish it or sign with it. This option is incompatible with `-P` and `-A`.

-h This option prints a short summary of the options and arguments to `dnssec-keyfromlabel`.

-K directory This option sets the directory in which the key files are to be written.

-k This option generates KEY records rather than DNSKEY records.

-L ttl This option sets the default TTL to use for this key when it is converted into a DNSKEY RR. This is the TTL used when the key is imported into a zone, unless there was already a DNSKEY RRset in place, in which case the existing TTL would take precedence. Setting the default TTL to `0` or `none` removes it.

-p protocol This option sets the protocol value for the key. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in **RFC 2535** and its successors.

-S key This option generates a key as an explicit successor to an existing key. The name, algorithm, size, and type of the key are set to match the predecessor. The activation date of the new key is set to the inactivation date of the existing one. The publication date is set to the activation date minus the prepublication interval, which defaults to 30 days.

-t type This option indicates the type of the key. `type` must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF to the ability to encrypt data.

-v level This option sets the debugging level.

-V This option prints version information.

-y This option allows DNSSEC key files to be generated even if the key ID would collide with that of an existing key, in the event of either key being revoked. (This is only safe to enable if **RFC 5011** trust anchor maintenance is not used with either of the keys involved.)

12.11.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a + or −, it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use `none` or `never`.

-P date/offset This option sets the date on which a key is to be published to the zone. After that date, the key is included in the zone but is not used to sign it. If not set, and if the `-G` option has not been used, the default is the current date.

-P sync date/offset This option sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

-A date/offset This option sets the date on which the key is to be activated. After that date, the key is included in the zone and used to sign it. If not set, and if the `-G` option has not been used, the default is the current date.

-R date/offset This option sets the date on which the key is to be revoked. After that date, the key is flagged as revoked. It is included in the zone and is used to sign it.

-I date/offset This option sets the date on which the key is to be retired. After that date, the key is still included in the zone, but it is not used to sign it.

-D date/offset This option sets the date on which the key is to be deleted. After that date, the key is no longer included in the zone. (However, it may remain in the key repository.)

-D sync date/offset This option sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

-i interval This option sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date is not, the publication date defaults to this much time before the activation date; conversely, if the publication date is specified but not the activation date, activation is set to this much time after publication.

   If the key is being created as an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

   As with date offsets, if the argument is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

12.11.5 Generated Key Files

When `dnssec-keyfromlabel` completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key files it has generated.

- `nnnn` is the key name.

- `aaa` is the numeric representation of the algorithm.

- `iiiii` is the key identifier (or footprint).

`dnssec-keyfromlabel` creates two files, with names based on the printed string. `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNS KEY record that can be inserted into a zone file (directly or with an $INCLUDE statement).

The `.private` file contains algorithm-specific fields. For obvious security reasons, this file does not have general read permission.

12.11.6 See Also

dnssec-keygen(8), *dnssec-signzone(8)*, BIND 9 Administrator Reference Manual, **RFC 4034**, **RFC 7512**.

12.12 dnssec-keygen: DNSSEC key generation tool

12.12.1 Synopsis

dnssec-keygen [-**3**] [-**A** date/offset] [-**a** algorithm] [-**b** keysize] [-**C**] [-**c** class] [-**D** date/offset] [-**d** bits] [-**D** sync date/offset] [-**E** engine] [-**f** flag] [-**G**] [-**g** generator] [-**h**] [-**I** date/offset] [-**i** interval] [-**K** directory] [-**k** policy] [-**L** ttl] [-**l** file] [-**n** nametype] [-**P** date/offset] [-**P** sync date/offset] [-**p** protocol] [-**q**] [-**R** date/offset] [-**S** key] [-**s** strength] [-**T** rrtype] [-**t** type] [-**V**] [-**v** level] {name}

12.12.2 Description

`dnssec-keygen` generates keys for DNSSEC (Secure DNS), as defined in **RFC 2535** and **RFC 4034**. It can also generate keys for use with TSIG (Transaction Signatures) as defined in **RFC 2845**, or TKEY (Transaction Key) as defined in **RFC 2930**.

The `name` of the key is specified on the command line. For DNSSEC keys, this must match the name of the zone for which the key is being generated.

The `dnssec-keymgr` command acts as a wrapper around `dnssec-keygen`, generating and updating keys as needed to enforce defined security policies such as key rollover scheduling. Using `dnssec-keymgr` may be preferable to direct use of `dnssec-keygen`.

12.12.3 Options

-3 This option uses an NSEC3-capable algorithm to generate a DNSSEC key. If this option is used with an algorithm that has both NSEC and NSEC3 versions, then the NSEC3 version is selected; for example, `dnssec-keygen -3a RSASHA1` specifies the NSEC3RSASHA1 algorithm.

-a algorithm This option selects the cryptographic algorithm. For DNSSEC keys, the value of `algorithm` must be one of RSASHA1, NSEC3RSASHA1, RSASHA256, RSASHA512, ECDSAP256SHA256, ECD-SAP384SHA384, ED25519, or ED448. For TKEY, the value must be DH (Diffie-Hellman); specifying this value automatically sets the `-T KEY` option as well.

These values are case-insensitive. In some cases, abbreviations are supported, such as ECDSA256 for ECD-SAP256SHA256 and ECDSA384 for ECDSAP384SHA384. If RSASHA1 is specified along with the `-3` option, NSEC3RSASHA1 is used instead.

This parameter *must* be specified except when using the `-S` option, which copies the algorithm from the predecessor key.

In prior releases, HMAC algorithms could be generated for use as TSIG keys, but that feature was removed in BIND 9.13.0. Use `tsig-keygen` to generate TSIG keys.

-b keysize This option specifies the number of bits in the key. The choice of key size depends on the algorithm used: RSA keys must be between 1024 and 4096 bits; Diffie-Hellman keys must be between 128 and 4096 bits. Elliptic curve algorithms do not need this parameter.

If the key size is not specified, some algorithms have pre-defined defaults. For example, RSA keys for use as DNSSEC zone-signing keys have a default size of 1024 bits; RSA keys for use as key-signing keys (KSKs, generated with `-f KSK`) default to 2048 bits.

-C This option enables compatibility mode, which generates an old-style key, without any timing metadata. By default, `dnssec-keygen` includes the key's creation date in the metadata stored with the private key; other dates may be set there as well, including publication date, activation date, etc. Keys that include this data may be incompatible with older versions of BIND; the `-C` option suppresses them.

-c class This option indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-d bits This option specifies the key size in bits. For the algorithms RSASHA1, NSEC3RSASA1, RSASHA256, and RSASHA512 the key size must be between 1024 and 4096 bits; DH size is between 128 and 4096 bits. This option is ignored for algorithms ECDSAP256SHA256, ECDSAP384SHA384, ED25519, and ED448.

-E engine This option specifies the cryptographic hardware to use, when applicable.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

-f flag This option sets the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flags are KSK (Key-Signing Key) and REVOKE.

-G This option generates a key, but does not publish it or sign with it. This option is incompatible with `-P` and `-A`.

-g generator This option indicates the generator to use if generating a Diffie-Hellman key. Allowed values are 2 and 5. If no generator is specified, a known prime from **RFC 2539** is used if possible; otherwise the default is 2.

-h This option prints a short summary of the options and arguments to `dnssec-keygen`.

-K directory This option sets the directory in which the key files are to be written.

-k policy This option creates keys for a specific `dnssec-policy`. If a policy uses multiple keys, `dnssec-keygen` generates multiple keys. This also creates a ".state" file to keep track of the key state.

This option creates keys according to the `dnssec-policy` configuration, hence it cannot be used at the same time as many of the other options that `dnssec-keygen` provides.

-L ttl This option sets the default TTL to use for this key when it is converted into a DNSKEY RR. This is the TTL used when the key is imported into a zone, unless there was already a DNSKEY RRset in place, in which case the existing TTL takes precedence. If this value is not set and there is no existing DNSKEY RRset, the TTL defaults to the SOA TTL. Setting the default TTL to `0` or `none` is the same as leaving it unset.

-l file This option provides a configuration file that contains a `dnssec-policy` statement (matching the policy set with `-k`).

-n nametype This option specifies the owner type of the key. The value of `nametype` must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user (KEY)), or OTHER (DNSKEY). These values are case-insensitive. The default is ZONE for DNSKEY generation.

-p protocol This option sets the protocol value for the generated key, for use with `-T KEY`. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in **RFC 2535** and its successors.

-q This option sets quiet mode, which suppresses unnecessary output, including progress indication. Without this option, when `dnssec-keygen` is run interactively to generate an RSA or DSA key pair, it prints a string of symbols to `stderr` indicating the progress of the key generation. A `.` indicates that a random number has been found which passed an initial sieve test; `+` means a number has passed a single round of the Miller-Rabin primality test; and a space ( ) means that the number has passed all the tests and is a satisfactory key.

-S key This option creates a new key which is an explicit successor to an existing key. The name, algorithm, size, and type of the key are set to match the existing key. The activation date of the new key is set to the inactivation date of the existing one. The publication date is set to the activation date minus the prepublication interval, which defaults to 30 days.

-s strength This option specifies the strength value of the key. The strength is a number between 0 and 15, and currently has no defined purpose in DNSSEC.

-T rrtype This option specifies the resource record type to use for the key. `rrtype` must be either DNSKEY or KEY. The default is DNSKEY when using a DNSSEC algorithm, but it can be overridden to KEY for use with SIG(0).

-t type This option indicates the type of the key for use with `-T KEY`. `type` must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF to the ability to encrypt data.

-V This option prints version information.

-v level This option sets the debugging level.

12.12.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a + or −, it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use `none` or `never`.

-P date/offset This option sets the date on which a key is to be published to the zone. After that date, the key is included in the zone but is not used to sign it. If not set, and if the `-G` option has not been used, the default is the current date.

-P sync date/offset This option sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

-A date/offset This option sets the date on which the key is to be activated. After that date, the key is included in the zone and used to sign it. If not set, and if the `-G` option has not been used, the default is the current date. If set, and `-P` is not set, the publication date is set to the activation date minus the prepublication interval.

-R date/offset This option sets the date on which the key is to be revoked. After that date, the key is flagged as revoked. It is included in the zone and is used to sign it.

-I date/offset This option sets the date on which the key is to be retired. After that date, the key is still included in the zone, but it is not used to sign it.

-D date/offset This option sets the date on which the key is to be deleted. After that date, the key is no longer included in the zone. (However, it may remain in the key repository.)

-D sync date/offset This option sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

-i interval This option sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date is not, the publication date defaults to this much time before the activation date; conversely, if the publication date is specified but not the activation date, activation is set to this much time after publication.

If the key is being created as an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

As with date offsets, if the argument is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

12.12.5 Generated Keys

When `dnssec-keygen` completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key it has generated.

- `nnnn` is the key name.

- `aaa` is the numeric representation of the algorithm.

- `iiiii` is the key identifier (or footprint).

`dnssec-keygen` creates two files, with names based on the printed string. `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNSKEY or KEY record. When a zone is being signed by `named` or `dnssec-signzone -S`, DNSKEY records are included automatically. In other cases, the `.key` file can be inserted into a zone file manually or with an `$INCLUDE` statement.

The `.private` file contains algorithm-specific fields. For obvious security reasons, this file does not have general read permission.

12.12.6 Example

To generate an ECDSAP256SHA256 zone-signing key for the zone `example.com`, issue the command:

`dnssec-keygen -a ECDSAP256SHA256 example.com`

The command prints a string of the form:

`Kexample.com.+013+26160`

In this example, `dnssec-keygen` creates the files `Kexample.com.+013+26160.key` and `Kexample.com.+013+26160.private`.

To generate a matching key-signing key, issue the command:

`dnssec-keygen -a ECDSAP256SHA256 -f KSK example.com`

12.12.7 See Also

dnssec-signzone(8), BIND 9 Administrator Reference Manual, **RFC 2539**, **RFC 2845**, **RFC 4034**.

12.13 dnssec-revoke - set the REVOKED bit on a DNSSEC key

12.13.1 Synopsis

dnssec-revoke [**-hr**] [**-v** level] [**-V**] [**-K** directory] [**-E** engine] [**-f**] [**-R**] {keyfile}

12.13.2 Description

`dnssec-revoke` reads a DNSSEC key file, sets the REVOKED bit on the key as defined in **RFC 5011**, and creates a new pair of key files containing the now-revoked key.

12.13.3 Options

-h This option emits a usage message and exits.

-K directory This option sets the directory in which the key files are to reside.

-r This option indicates to remove the original keyset files after writing the new keyset files.

-v level This option sets the debugging level.

-V This option prints version information.

-E engine This option specifies the cryptographic hardware to use, when applicable.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

-f This option indicates a forced overwrite and causes `dnssec-revoke` to write the new key pair, even if a file already exists matching the algorithm and key ID of the revoked key.

-R This option prints the key tag of the key with the REVOKE bit set, but does not revoke the key.

12.13.4 See Also

dnssec-keygen(8), BIND 9 Administrator Reference Manual, **RFC 5011**.

12.14 dnssec-settime: set the key timing metadata for a DNSSEC key

12.14.1 Synopsis

dnssec-settime [**-f**] [**-K** directory] [**-L** ttl] [**-P** date/offset] [**-P** ds date/offset] [**-P** sync date/offset] [**-A** date/offset] [**-R** date/offset] [**-I** date/offset] [**-D** date/offset] [**-D** ds date/offset] [**-D** sync date/offset] [**-S** key] [**-i** interval] [**-h**] [**-V**] [**-v** level] [**-E** engine] {keyfile} [**-s**] [**-g** state] [**-d** state date/offset] [**-k** state date/offset] [**-r** state date/offset] [**-z** state date/offset]

12.14.2 Description

dnssec-settime reads a DNSSEC private key file and sets the key timing metadata as specified by the -P, -A, -R, -I, and -D options. The metadata can then be used by dnssec-signzone or other signing software to determine when a key is to be published, whether it should be used for signing a zone, etc.

If none of these options is set on the command line, dnssec-settime simply prints the key timing metadata already stored in the key.

When key metadata fields are changed, both files of a key pair (Knnnn.+aaa+iiiii.key and Knnnn.+aaa+iiiii.private) are regenerated.

Metadata fields are stored in the private file. A human-readable description of the metadata is also placed in comments in the key file. The private file's permissions are always set to be inaccessible to anyone other than the owner (mode 0600).

When working with state files, it is possible to update the timing metadata in those files as well with -s. With this option, it is also possible to update key states with -d (DS), -k (DNSKEY), -r (RRSIG of KSK), or -z (RRSIG of ZSK). Allowed states are HIDDEN, RUMOURED, OMNIPRESENT, and UNRETENTIVE.

The goal state of the key can also be set with -g. This should be either HIDDEN or OMNIPRESENT, representing whether the key should be removed from the zone or published.

It is NOT RECOMMENDED to manipulate state files manually, except for testing purposes.

12.14.3 Options

-f This option forces an update of an old-format key with no metadata fields. Without this option, dnssec-settime fails when attempting to update a legacy key. With this option, the key is recreated in the new format, but with the original key data retained. The key's creation date is set to the present time. If no other values are specified, then the key's publication and activation dates are also set to the present time.

-K directory This option sets the directory in which the key files are to reside.

-L ttl This option sets the default TTL to use for this key when it is converted into a DNSKEY RR. This is the TTL used when the key is imported into a zone, unless there was already a DNSKEY RRset in place, in which case the existing TTL takes precedence. If this value is not set and there is no existing DNSKEY RRset, the TTL defaults to the SOA TTL. Setting the default TTL to 0 or none removes it from the key.

-h This option emits a usage message and exits.

-V This option prints version information.

–v level This option sets the debugging level.

–E engine This option specifies the cryptographic hardware to use, when applicable.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

12.14.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. If the argument begins with a + or −, it is interpreted as an offset from the present time. For convenience, if such an offset is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, then the offset is computed in years (defined as 365 24-hour days, ignoring leap years), months (defined as 30 24-hour days), weeks, days, hours, or minutes, respectively. Without a suffix, the offset is computed in seconds. To explicitly prevent a date from being set, use `none` or `never`.

–P date/offset This option sets the date on which a key is to be published to the zone. After that date, the key is included in the zone but is not used to sign it.

–P ds date/offset This option Sets the date on which DS records that match this key have been seen in the parent zone.

–P sync date/offset This option sets the date on which CDS and CDNSKEY records that match this key are to be published to the zone.

–A date/offset This option sets the date on which the key is to be activated. After that date, the key is included in the zone and used to sign it.

–R date/offset This option sets the date on which the key is to be revoked. After that date, the key is flagged as revoked. It is included in the zone and is used to sign it.

–I date/offset This option sets the date on which the key is to be retired. After that date, the key is still included in the zone, but it is not used to sign it.

–D date/offset This option sets the date on which the key is to be deleted. After that date, the key is no longer included in the zone. (However, it may remain in the key repository.)

–D ds date/offset This option sets the date on which the DS records that match this key have been seen removed from the parent zone.

–D sync date/offset This option sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

–S predecessor key This option selects a key for which the key being modified is an explicit successor. The name, algorithm, size, and type of the predecessor key must exactly match those of the key being modified. The activation date of the successor key is set to the inactivation date of the predecessor. The publication date is set to the activation date minus the prepublication interval, which defaults to 30 days.

–i interval This option sets the prepublication interval for a key. If set, then the publication and activation dates must be separated by at least this much time. If the activation date is specified but the publication date is not, the publication date defaults to this much time before the activation date; conversely, if the publication date is specified but not the activation date, activation is set to this much time after publication.

If the key is being created as an explicit successor to another key, then the default prepublication interval is 30 days; otherwise it is zero.

As with date offsets, if the argument is followed by one of the suffixes `y`, `mo`, `w`, `d`, `h`, or `mi`, the interval is measured in years, months, weeks, days, hours, or minutes, respectively. Without a suffix, the interval is measured in seconds.

12.14.5 Key State Options

To test dnssec-policy it may be necessary to construct keys with artificial state information; these options are used by the testing framework for that purpose, but should never be used in production.

Known key states are HIDDEN, RUMOURED, OMNIPRESENT, and UNRETENTIVE.

–s This option indicates that when setting key timing data, the state file should also be updated.

–g state This option sets the goal state for this key. Must be HIDDEN or OMNIPRESENT.

–d state date/offset This option sets the DS state for this key as of the specified date, offset from the current date.

–k state date/offset This option sets the DNSKEY state for this key as of the specified date, offset from the current date.

–r state date/offset This option sets the RRSIG (KSK) state for this key as of the specified date, offset from the current date.

–z state date/offset This option sets the RRSIG (ZSK) state for this key as of the specified date, offset from the current date.

12.14.6 Printing Options

`dnssec-settime` can also be used to print the timing metadata associated with a key.

–u This option indicates that times should be printed in Unix epoch format.

–p C/P/Pds/Psync/A/R/I/D/Dds/Dsync/all This option prints a specific metadata value or set of metadata values. The `–p` option may be followed by one or more of the following letters or strings to indicate which value or values to print: `C` for the creation date, `P` for the publication date, `Pds`` for the DS publication date, ``Psync` for the CDS and CDNSKEY publication date, `A` for the activation date, `R` for the revocation date, `I` for the inactivation date, `D` for the deletion date, `Dds` for the DS deletion date, and `Dsync` for the CDS and CDNSKEY deletion date. To print all of the metadata, use `all`.

12.14.7 See Also

dnssec-keygen(8), *dnssec-signzone(8)*, BIND 9 Administrator Reference Manual, RFC 5011.

12.15 dnssec-signzone - DNSSEC zone signing tool

12.15.1 Synopsis

dnssec-signzone [**-a**] [**-c** class] [**-d** directory] [**-D**] [**-E** engine] [**-e** end-time] [**-f** output-file] [**-g**] [**-h**] [**-i** interval] [**-I** input-format] [**-j** jitter] [**-K** directory] [**-k** key] [**-L** serial] [**-M** maxttl] [**-N** soa-serial-format] [**-o** origin] [**-O** output-format] [**-P**] [**-Q**] [**-q**] [**-R**] [**-S**] [**-s** start-time] [**-T** ttl] [**-t**] [**-u**] [**-v** level] [**-V**] [**-X** extended end-time] [**-x**] [**-z**] [**-3** salt] [**-H** iterations] [**-A**] {zonefile} [key…]

12.15.2 Description

`dnssec-signzone` signs a zone; it generates NSEC and RRSIG records and produces a signed version of the zone. The security status of delegations from the signed zone (that is, whether the child zones are secure) is determined by the presence or absence of a `keyset` file for each child zone.

12.15.3 Options

–a This option verifies all generated signatures.

–c class This option specifies the DNS class of the zone.

–C This option sets compatibility mode, in which a `keyset-zonename` file is generated in addition to `dsset-zonename` when signing a zone, for use by older versions of `dnssec-signzone`.

–d directory This option indicates the directory where BIND 9 should look for `dsset-` or `keyset-` files.

–D This option indicates that only those record types automatically managed by `dnssec-signzone`, i.e., RRSIG, NSEC, NSEC3 and NSEC3PARAM records, should be included in the output. If smart signing (`-S`) is used, DNSKEY records are also included. The resulting file can be included in the original zone file with `$INCLUDE`. This option cannot be combined with `-O raw`, `-O map`, or serial-number updating.

–E engine This option specifies the hardware to use for cryptographic operations, such as a secure key store used for signing, when applicable.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

–g This option indicates that DS records for child zones should be generated from a `dsset-` or `keyset-` file. Existing DS records are removed.

–K directory This option specifies the directory to search for DNSSEC keys. If not specified, it defaults to the current directory.

–k key This option tells BIND 9 to treat the specified key as a key-signing key, ignoring any key flags. This option may be specified multiple times.

–M maxttl This option sets the maximum TTL for the signed zone. Any TTL higher than `maxttl` in the input zone is reduced to `maxttl` in the output. This provides certainty as to the largest possible TTL in the signed zone, which is useful to know when rolling keys. The maxttl is the longest possible time before signatures that have been retrieved by resolvers expire from resolver caches. Zones that are signed with this option should be configured to use a matching `max-zone-ttl` in `named.conf`. (Note: This option is incompatible with `-D`, because it modifies non-DNSSEC data in the output zone.)

–s start-time This option specifies the date and time when the generated RRSIG records become valid. This can be either an absolute or relative time. An absolute start time is indicated by a number in YYYYMMDDHHMMSS notation; 20000530144500 denotes 14:45:00 UTC on May 30th, 2000. A relative start time is indicated by +N, which is N seconds from the current time. If no `start-time` is specified, the current time minus 1 hour (to allow for clock skew) is used.

–e end-time This option specifies the date and time when the generated RRSIG records expire. As with `start-time`, an absolute time is indicated in YYYYMMDDHHMMSS notation. A time relative to the start time is indicated with +N, which is N seconds from the start time. A time relative to the current time is indicated with now+N. If no `end-time` is specified, 30 days from the start time is the default. `end-time` must be later than `start-time`.

-X extended end-time This option specifies the date and time when the generated RRSIG records for the DNSKEY RRset expire. This is to be used in cases when the DNSKEY signatures need to persist longer than signatures on other records; e.g., when the private component of the KSK is kept offline and the KSK signature is to be refreshed manually.

As with `end-time`, an absolute time is indicated in YYYYMMDDHHMMSS notation. A time relative to the start time is indicated with +N, which is N seconds from the start time. A time relative to the current time is indicated with `now+N`. If no `extended end-time` is specified, the value of `end-time` is used as the default. (`end-time`, in turn, defaults to 30 days from the start time.) `extended end-time` must be later than `start-time`.

-f output-file This option indicates the name of the output file containing the signed zone. The default is to append `.signed` to the input filename. If `output-file` is set to `-`, then the signed zone is written to the standard output, with a default output format of `full`.

-h This option prints a short summary of the options and arguments to `dnssec-signzone`.

-V This option prints version information.

-i interval This option indicates that, when a previously signed zone is passed as input, records may be re-signed. The `interval` option specifies the cycle interval as an offset from the current time, in seconds. If a RRSIG record expires after the cycle interval, it is retained; otherwise, it is considered to be expiring soon and it is replaced.

The default cycle interval is one quarter of the difference between the signature end and start times. So if neither `end-time` nor `start-time` is specified, `dnssec-signzone` generates signatures that are valid for 30 days, with a cycle interval of 7.5 days. Therefore, if any existing RRSIG records are due to expire in less than 7.5 days, they are replaced.

-I input-format This option sets the format of the input zone file. Possible formats are `text` (the default), `raw`, and `map`. This option is primarily intended to be used for dynamic signed zones, so that the dumped zone file in a non-text format containing updates can be signed directly. This option is not useful for non-dynamic zones.

-j jitter When signing a zone with a fixed signature lifetime, all RRSIG records issued at the time of signing expire simultaneously. If the zone is incrementally signed, i.e., a previously signed zone is passed as input to the signer, all expired signatures must be regenerated at approximately the same time. The `jitter` option specifies a jitter window that is used to randomize the signature expire time, thus spreading incremental signature regeneration over time.

Signature lifetime jitter also, to some extent, benefits validators and servers by spreading out cache expiration, i.e., if large numbers of RRSIGs do not expire at the same time from all caches, there is less congestion than if all validators need to refetch at around the same time.

-L serial When writing a signed zone to "raw" or "map" format, this option sets the "source serial" value in the header to the specified `serial` number. (This is expected to be used primarily for testing purposes.)

-n ncpus This option specifies the number of threads to use. By default, one thread is started for each detected CPU.

-N soa-serial-format This option sets the SOA serial number format of the signed zone. Possible formats are `keep` (the default), `increment`, `unixtime`, and `date`.

keep This format indicates that the SOA serial number should not be modified.

increment This format increments the SOA serial number using **RFC 1982** arithmetic.

unixtime This format sets the SOA serial number to the number of seconds since the beginning of the Unix epoch, unless the serial number is already greater than or equal to that value, in which case it is simply incremented by one.

date This format sets the SOA serial number to today's date, in YYYYMMDDNN format, unless the serial number is already greater than or equal to that value, in which case it is simply incremented by one.

-o origin This option sets the zone origin. If not specified, the name of the zone file is assumed to be the origin.

-O output-format This option sets the format of the output file containing the signed zone. Possible formats are text (the default), which is the standard textual representation of the zone; full, which is text output in a format suitable for processing by external scripts; and map, raw, and raw=N, which store the zone in binary formats for rapid loading by named. raw=N specifies the format version of the raw zone file: if N is 0, the raw file can be read by any version of named; if N is 1, the file can be read by release 9.9.0 or higher. The default is 1.

-P This option disables post-sign verification tests.

The post-sign verification tests ensure that for each algorithm in use there is at least one non-revoked self-signed KSK key, that all revoked KSK keys are self-signed, and that all records in the zone are signed by the algorithm. This option skips these tests.

-Q This option removes signatures from keys that are no longer active.

Normally, when a previously signed zone is passed as input to the signer, and a DNSKEY record has been removed and replaced with a new one, signatures from the old key that are still within their validity period are retained. This allows the zone to continue to validate with cached copies of the old DNSKEY RRset. The -Q option forces dnssec-signzone to remove signatures from keys that are no longer active. This enables ZSK rollover using the procedure described in **RFC 4641#4.2.1.1** ("Pre-Publish Key Rollover").

-q This option enables quiet mode, which suppresses unnecessary output. Without this option, when dnssec-signzone is run it prints three pieces of information to standard output: the number of keys in use; the algorithms used to verify the zone was signed correctly and other status information; and the filename containing the signed zone. With the option that output is suppressed, leaving only the filename.

-R This option removes signatures from keys that are no longer published.

This option is similar to -Q, except it forces dnssec-signzone to remove signatures from keys that are no longer published. This enables ZSK rollover using the procedure described in **RFC 4641#4.2.1.2** ("Double Signature Zone Signing Key Rollover").

-S This option enables smart signing, which instructs dnssec-signzone to search the key repository for keys that match the zone being signed, and to include them in the zone if appropriate.

When a key is found, its timing metadata is examined to determine how it should be used, according to the following rules. Each successive rule takes priority over the prior ones:

If no timing metadata has been set for the key, the key is published in the zone and used to sign the zone.

If the key's publication date is set and is in the past, the key is published in the zone.

If the key's activation date is set and is in the past, the key is published (regardless of publication date) and used to sign the zone.

If the key's revocation date is set and is in the past, and the key is published, then the key is revoked, and the revoked key is used to sign the zone.

If either the key's unpublication or deletion date is set and in the past, the key is NOT published or used to sign the zone, regardless of any other metadata.

If the key's sync publication date is set and is in the past, synchronization records (type CDS and/or CDNSKEY) are created.

If the key's sync deletion date is set and is in the past, synchronization records (type CDS and/or CDNSKEY) are removed.

-T ttl This option specifies a TTL to be used for new DNSKEY records imported into the zone from the key repository. If not specified, the default is the TTL value from the zone's SOA record. This option is ignored when signing without -S, since DNSKEY records are not imported from the key repository in that case. It is also ignored if there are any pre-existing DNSKEY records at the zone apex, in which case new records' TTL values are set to match

them, or if any of the imported DNSKEY records had a default TTL value. In the event of a conflict between TTL values in imported keys, the shortest one is used.

−t This option prints statistics at completion.

−u This option updates the NSEC/NSEC3 chain when re-signing a previously signed zone. With this option, a zone signed with NSEC can be switched to NSEC3, or a zone signed with NSEC3 can be switched to NSEC or to NSEC3 with different parameters. Without this option, `dnssec-signzone` retains the existing chain when re-signing.

−v level This option sets the debugging level.

−x This option indicates that BIND 9 should only sign the DNSKEY, CDNSKEY, and CDS RRsets with key-signing keys, and should omit signatures from zone-signing keys. (This is similar to the `dnssec-dnskey-kskonly yes;` zone option in `named`.)

−z This option indicates that BIND 9 should ignore the KSK flag on keys when determining what to sign. This causes KSK-flagged keys to sign all records, not just the DNSKEY RRset. (This is similar to the `update-check-ksk no;` zone option in `named`.)

−3 salt This option generates an NSEC3 chain with the given hex-encoded salt. A dash (-) can be used to indicate that no salt is to be used when generating the NSEC3 chain.

−H iterations This option indicates that, when generating an NSEC3 chain, BIND 9 should use this many iterations. The default is 10.

−A This option indicates that, when generating an NSEC3 chain, BIND 9 should set the OPTOUT flag on all NSEC3 records and should not generate NSEC3 records for insecure delegations.

Using this option twice (i.e., `-AA`) turns the OPTOUT flag off for all records. This is useful when using the `-u` option to modify an NSEC3 chain which previously had OPTOUT set.

zonefile This option sets the file containing the zone to be signed.

key This option specifies which keys should be used to sign the zone. If no keys are specified, the zone is examined for DNSKEY records at the zone apex. If these records are found and there are matching private keys in the current directory, they are used for signing.

12.15.4 Example

The following command signs the `example.com` zone with the ECDSAP256SHA256 key generated by `dnssec-keygen` (Kexample.com.+013+17247). Because the `-S` option is not being used, the zone's keys must be in the master file (`db.example.com`). This invocation looks for `dsset` files in the current directory, so that DS records can be imported from them (`-g`).

```
% dnssec-signzone -g -o example.com db.example.com \
Kexample.com.+013+17247
db.example.com.signed
%
```

In the above example, `dnssec-signzone` creates the file `db.example.com.signed`. This file should be referenced in a zone statement in the `named.conf` file.

This example re-signs a previously signed zone with default parameters. The private keys are assumed to be in the current directory.

```
% cp db.example.com.signed db.example.com
% dnssec-signzone -o example.com db.example.com
db.example.com.signed
%
```

12.15.5 See Also

dnssec-keygen(8), BIND 9 Administrator Reference Manual, **RFC 4033**, **RFC 4641**.

12.16 dnssec-verify - DNSSEC zone verification tool

12.16.1 Synopsis

dnssec-verify [**-c** class] [**-E** engine] [**-I** input-format] [**-o** origin] [**-q**] [**-v** level] [**-V**] [**-x**] [**-z**] {zonefile}

12.16.2 Description

`dnssec-verify` verifies that a zone is fully signed for each algorithm found in the DNSKEY RRset for the zone, and that the NSEC/NSEC3 chains are complete.

12.16.3 Options

-c class This option specifies the DNS class of the zone.

-E engine This option specifies the cryptographic hardware to use, when applicable.

> When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

-I input-format This option sets the format of the input zone file. Possible formats are `text` (the default) and `raw`. This option is primarily intended to be used for dynamic signed zones, so that the dumped zone file in a non-text format containing updates can be verified independently. This option is not useful for non-dynamic zones.

-o origin This option indicates the zone origin. If not specified, the name of the zone file is assumed to be the origin.

-v level This option sets the debugging level.

-V This option prints version information.

-q This option sets quiet mode, which suppresses output. Without this option, when `dnssec-verify` is run it prints to standard output the number of keys in use, the algorithms used to verify the zone was signed correctly, and other status information. With this option, all non-error output is suppressed, and only the exit code indicates success.

-x This option verifies only that the DNSKEY RRset is signed with key-signing keys. Without this flag, it is assumed that the DNSKEY RRset is signed by all active keys. When this flag is set, it is not an error if the DNSKEY RRset is not signed by zone-signing keys. This corresponds to the `-x` option in `dnssec-signzone`.

-z This option indicates that the KSK flag on the keys should be ignored when determining whether the zone is correctly signed. Without this flag, it is assumed that there is a non-revoked, self-signed DNSKEY with the KSK flag set for each algorithm, and that RRsets other than DNSKEY RRset are signed with a different DNSKEY without the KSK flag set.

> With this flag set, BIND 9 only requires that for each algorithm, there be at least one non-revoked, self-signed DNSKEY, regardless of the KSK flag state, and that other RRsets be signed by a non-revoked key for the same algorithm that includes the self-signed key; the same key may be used for both purposes. This corresponds to the `-z` option in `dnssec-signzone`.

zonefile This option indicates the file containing the zone to be signed.

12.16.4 See Also

dnssec-signzone(8), BIND 9 Administrator Reference Manual, RFC 4033.

12.17 dnstap-read - print dnstap data in human-readable form

12.17.1 Synopsis

dnstap-read [**-m**] [**-p**] [**-x**] [**-y**] {file}

12.17.2 Description

dnstap-read reads dnstap data from a specified file and prints it in a human-readable format. By default, dnstap data is printed in a short summary format, but if the -y option is specified, a longer and more detailed YAML format is used.

12.17.3 Options

-m This option indicates trace memory allocations, and is used for debugging memory leaks.

-p This option prints the text form of the DNS message that was encapsulated in the dnstap frame, after printing the dnstap data.

-x This option prints a hex dump of the wire form of the DNS message that was encapsulated in the dnstap frame, after printing the dnstap data.

-y This option prints dnstap data in a detailed YAML format.

12.17.4 See Also

named(8), *rndc(8)*, BIND 9 Administrator Reference Manual.

12.18 filter-aaaa.so - filter AAAA in DNS responses when A is present

12.18.1 Synopsis

plugin query "filter-aaaa.so" [{ parameters }];

12.18.2 Description

`filter-aaaa.so` is a query plugin module for `named`, enabling `named` to omit some IPv6 addresses when responding to clients.

Until BIND 9.12, this feature was implemented natively in `named` and enabled with the `filter-aaaa` ACL and the `filter-aaaa-on-v4` and `filter-aaaa-on-v6` options. These options are now deprecated in `named.conf` but can be passed as parameters to the `filter-aaaa.so` plugin, for example:

```
plugin query "/usr/local/lib/filter-aaaa.so" {
        filter-aaaa-on-v4 yes;
        filter-aaaa-on-v6 yes;
        filter-aaaa { 192.0.2.1; 2001:db8:2::1; };
};
```

This module is intended to aid transition from IPv4 to IPv6 by withholding IPv6 addresses from DNS clients which are not connected to the IPv6 Internet, when the name being looked up has an IPv4 address available. Use of this module is not recommended unless absolutely necessary.

Note: This mechanism can erroneously cause other servers not to give AAAA records to their clients. If a recursing server with both IPv6 and IPv4 network connections queries an authoritative server using this mechanism via IPv4, it is denied AAAA records even if its client is using IPv6.

12.18.3 Options

filter-aaaa This option specifies a list of client addresses for which AAAA filtering is to be applied. The default is `any`.

filter-aaaa-on-v4 If set to `yes`, this option indicates that the DNS client is at an IPv4 address, in `filter-aaaa`. If the response does not include DNSSEC signatures, then all AAAA records are deleted from the response. This filtering applies to all responses, not only authoritative ones.

If set to `break-dnssec`, then AAAA records are deleted even when DNSSEC is enabled. As suggested by the name, this causes the response to fail to verify, because the DNSSEC protocol is designed to detect deletions.

This mechanism can erroneously cause other servers not to give AAAA records to their clients. If a recursing server with both IPv6 and IPv4 network connections queries an authoritative server using this mechanism via IPv4, it is denied AAAA records even if its client is using IPv6.

filter-aaaa-on-v6 This option is identical to `filter-aaaa-on-v4`, except that it filters AAAA responses to queries from IPv6 clients instead of IPv4 clients. To filter all responses, set both options to `yes`.

12.18.4 See Also

BIND 9 Administrator Reference Manual.

12.19 host - DNS lookup utility

12.19.1 Synopsis

host [**-aACdlnrsTUwv**] [**-c** class] [**-N** ndots] [**-p** port] [**-R** number] [**-t** type] [**-W** wait] [**-m** flag] [[**-4**] | [**-6**]] [**-v**] [**-V**] {name} [server]

12.19.2 Description

host is a simple utility for performing DNS lookups. It is normally used to convert names to IP addresses and vice versa. When no arguments or options are given, host prints a short summary of its command-line arguments and options.

name is the domain name that is to be looked up. It can also be a dotted-decimal IPv4 address or a colon-delimited IPv6 address, in which case host by default performs a reverse lookup for that address. server is an optional argument which is either the name or IP address of the name server that host should query instead of the server or servers listed in /etc/resolv.conf.

12.19.3 Options

-4 This option specifies that only IPv4 should be used for query transport. See also the -6 option.

-6 This option specifies that only IPv6 should be used for query transport. See also the -4 option.

-a The -a ("all") option is normally equivalent to -v -t ANY. It also affects the behavior of the -l list zone option.

-A The -A ("almost all") option is equivalent to -a, except that RRSIG, NSEC, and NSEC3 records are omitted from the output.

-c class This option specifies the query class, which can be used to lookup HS (Hesiod) or CH (Chaosnet) class resource records. The default class is IN (Internet).

-C This option indicates that named should check consistency, meaning that host queries the SOA records for zone name from all the listed authoritative name servers for that zone. The list of name servers is defined by the NS records that are found for the zone.

-d This option prints debugging traces, and is equivalent to the -v verbose option.

-l This option tells named to list the zone, meaning the host command performs a zone transfer of zone name and prints out the NS, PTR, and address records (A/AAAA).

Together, the -l -a options print all records in the zone.

-N ndots This option specifies the number of dots (ndots) that have to be in name for it to be considered absolute. The default value is that defined using the ndots statement in /etc/resolv.conf, or 1 if no ndots statement is present. Names with fewer dots are interpreted as relative names, and are searched for in the domains listed in the search or domain directive in /etc/resolv.conf.

-p port This option specifies the port to query on the server. The default is 53.

-r This option specifies a non-recursive query; setting this option clears the RD (recursion desired) bit in the query. This means that the name server receiving the query does not attempt to resolve name. The -r option enables host to mimic the behavior of a name server by making non-recursive queries, and expecting to receive answers to those queries that can be referrals to other name servers.

-R number This option specifies the number of retries for UDP queries. If number is negative or zero, the number of retries is silently set to 1. The default value is 1, or the value of the attempts option in /etc/resolv.conf, if set.

-s This option tells `named` *not* to send the query to the next nameserver if any server responds with a SERVFAIL response, which is the reverse of normal stub resolver behavior.

-t type This option specifies the query type. The `type` argument can be any recognized query type: CNAME, NS, SOA, TXT, DNSKEY, AXFR, etc.

When no query type is specified, `host` automatically selects an appropriate query type. By default, it looks for A, AAAA, and MX records. If the `-C` option is given, queries are made for SOA records. If `name` is a dotted-decimal IPv4 address or colon-delimited IPv6 address, `host` queries for PTR records.

If a query type of IXFR is chosen, the starting serial number can be specified by appending an equals sign (=), followed by the starting serial number, e.g., `-t IXFR=12345678`.

-T; -U This option specifies TCP or UDP. By default, `host` uses UDP when making queries; the `-T` option makes it use a TCP connection when querying the name server. TCP is automatically selected for queries that require it, such as zone transfer (AXFR) requests. Type `ANY` queries default to TCP, but can be forced to use UDP initially via `-U`.

-m flag This option sets memory usage debugging: the flag can be `record`, `usage`, or `trace`. The `-m` option can be specified more than once to set multiple flags.

-v This option sets verbose output, and is equivalent to the `-d` debug option. Verbose output can also be enabled by setting the `debug` option in `/etc/resolv.conf`.

-V This option prints the version number and exits.

-w This option sets "wait forever": the query timeout is set to the maximum possible. See also the `-W` option.

-W wait This options sets the length of the wait timeout, indicating that `named` should wait for up to `wait` seconds for a reply. If `wait` is less than 1, the wait interval is set to 1 second.

By default, `host` waits for 5 seconds for UDP responses and 10 seconds for TCP connections. These defaults can be overridden by the `timeout` option in `/etc/resolv.conf`.

See also the `-w` option.

12.19.4 IDN Support

If `host` has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. `host` appropriately converts character encoding of a domain name before sending a request to a DNS server or displaying a reply from the server. To turn off IDN support, define the `IDN_DISABLE` environment variable. IDN support is disabled if the variable is set when `host` runs.

12.19.5 Files

`/etc/resolv.conf`

12.19.6 See Also

dig(1), *named(8)*.

12.20 mdig - DNS pipelined lookup utility

12.20.1 Synopsis

mdig {@server} [**-f** filename] [**-h**] [**-v**] [[**-4**] | [**-6**]] [**-m**] [**-b** address] [**-p** port#] [**-c** class] [**-t** type] [**-i**] [**-x** addr] [plusopt…]

mdig {**-h**}

mdig [@server] {global-opt…} { {local-opt…} {query} …}

12.20.2 Description

mdig is a multiple/pipelined query version of dig: instead of waiting for a response after sending each query, it begins by sending all queries. Responses are displayed in the order in which they are received, not in the order the corresponding queries were sent.

mdig options are a subset of the dig options, and are divided into "anywhere options," which can occur anywhere, "global options," which must occur before the query name (or they are ignored with a warning), and "local options," which apply to the next query on the command line.

The @server option is a mandatory global option. It is the name or IP address of the name server to query. (Unlike dig, this value is not retrieved from /etc/resolv.conf.) It can be an IPv4 address in dotted-decimal notation, an IPv6 address in colon-delimited notation, or a hostname. When the supplied server argument is a hostname, mdig resolves that name before querying the name server.

mdig provides a number of query options which affect the way in which lookups are made and the results displayed. Some of these set or reset flag bits in the query header, some determine which sections of the answer get printed, and others determine the timeout and retry strategies.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option. These may be preceded by the string no to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form +keyword=value.

12.20.3 Anywhere Options

-f This option makes mdig operate in batch mode by reading a list of lookup requests to process from the file filename. The file contains a number of queries, one per line. Each entry in the file should be organized in the same way they would be presented as queries to mdig using the command-line interface.

-h This option causes mdig to print detailed help information, with the full list of options, and exit.

-v This option causes mdig to print the version number and exit.

12.20.4 Global Options

-4 This option forces mdig to only use IPv4 query transport.

-6 This option forces mdig to only use IPv6 query transport.

-b address This option sets the source IP address of the query to address. This must be a valid address on one of the host's network interfaces or "0.0.0.0" or "::". An optional port may be specified by appending "#<port>"

-m This option enables memory usage debugging.

-p port# This option is used when a non-standard port number is to be queried. `port#` is the port number that `mdig` sends its queries to, instead of the standard DNS port number 53. This option is used to test a name server that has been configured to listen for queries on a non-standard port number.

The global query options are:

+[no]additional This option displays [or does not display] the additional section of a reply. The default is to display it.

+[no]all This option sets or clears all display flags.

+[no]answer This option displays [or does not display] the answer section of a reply. The default is to display it.

+[no]authority This option displays [or does not display] the authority section of a reply. The default is to display it.

+[no]besteffort This option attempts to display [or does not display] the contents of messages which are malformed. The default is to not display malformed answers.

+burst This option delays queries until the start of the next second.

+[no]cl This option displays [or does not display] the CLASS when printing the record.

+[no]comments This option toggles the display of comment lines in the output. The default is to print comments.

+[no]continue This option toggles continuation on errors (e.g. timeouts).

+[no]crypto This option toggles the display of cryptographic fields in DNSSEC records. The contents of these fields are unnecessary to debug most DNSSEC validation failures and removing them makes it easier to see the common failures. The default is to display the fields. When omitted, they are replaced by the string "[omitted]"; in the DNSKEY case, the key ID is displayed as the replacement, e.g., `[ key id = value ]`.

+dscp[=value] This option sets the DSCP code point to be used when sending the query. Valid DSCP code points are in the range [0…63]. By default no code point is explicitly set.

+[no]multiline This option toggles printing of records, like the SOA records, in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the `mdig` output.

+[no]question This option prints [or does not print] the question section of a query when an answer is returned. The default is to print the question section as a comment.

+[no]rrcomments This option toggles the display of per-record comments in the output (for example, human-readable key information about DNSKEY records). The default is not to print record comments unless multiline mode is active.

+[no]short This option provides [or does not provide] a terse answer. The default is to print the answer in a verbose form.

+split=W This option splits long hex- or base64-formatted fields in resource records into chunks of W characters (where W is rounded up to the nearest multiple of 4). `+nosplit` or `+split=0` causes fields not to be split. The default is 56 characters, or 44 characters when multiline mode is active.

+[no]tcp This option uses [or does not use] TCP when querying name servers. The default behavior is to use UDP.

+[no]ttlid This option displays [or does not display] the TTL when printing the record.

+[no]ttlunits This option displays [or does not display] the TTL in friendly human-readable time units of "s", "m", "h", "d", and "w", representing seconds, minutes, hours, days, and weeks. This implies +ttlid.

+[no]vc This option uses [or does not use] TCP when querying name servers. This alternate syntax to `+[no]tcp` is provided for backwards compatibility. The `vc` stands for "virtual circuit".

12.20.5 Local Options

-c class This option sets the query class to `class`. It can be any valid query class which is supported in BIND 9. The default query class is "IN".

-t type This option sets the query type to `type`. It can be any valid query type which is supported in BIND 9. The default query type is "A", unless the `-x` option is supplied to indicate a reverse lookup with the "PTR" query type.

-x addr Reverse lookups - mapping addresses to names - are simplified by this option. `addr` is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. `mdig` automatically performs a lookup for a query name like `11.12.13.10.in-addr.arpa` and sets the query type and class to PTR and IN respectively. By default, IPv6 addresses are looked up using nibble format under the IP6.ARPA domain.

The local query options are:

+[no]aaflag This is a synonym for `+[no]aaonly`.

+[no]aaonly This sets the `aa` flag in the query.

+[no]adflag This sets [or does not set] the AD (authentic data) bit in the query. This requests the server to return whether all of the answer and authority sections have all been validated as secure, according to the security policy of the server. AD=1 indicates that all records have been validated as secure and the answer is not from a OPT-OUT range. AD=0 indicates that some part of the answer was insecure or not validated. This bit is set by default.

+bufsize=B This sets the UDP message buffer size advertised using EDNS0 to `B` bytes. The maximum and minimum sizes of this buffer are 65535 and 0 respectively. Values outside this range are rounded up or down appropriately. Values other than zero cause a EDNS query to be sent.

+[no]cdflag This sets [or does not set] the CD (checking disabled) bit in the query. This requests the server to not perform DNSSEC validation of responses.

+[no]cookie=#### This sends [or does not send] a COOKIE EDNS option, with an optional value. Replaying a COOKIE from a previous response allows the server to identify a previous client. The default is `+nocookie`.

+[no]dnssec This requests that DNSSEC records be sent by setting the DNSSEC OK (DO) bit in the OPT record in the additional section of the query.

+[no]edns[=#] This specifies [or does not specify] the EDNS version to query with. Valid values are 0 to 255. Setting the EDNS version causes an EDNS query to be sent. `+noedns` clears the remembered EDNS version. EDNS is set to 0 by default.

+[no]ednsflags[=#] This sets the must-be-zero EDNS flag bits (Z bits) to the specified value. Decimal, hex, and octal encodings are accepted. Setting a named flag (e.g. DO) is silently ignored. By default, no Z bits are set.

+[no]ednsopt[=code[:value]] This specifies [or does not specify] an EDNS option with code point `code` and an optional payload of `value` as a hexadecimal string. `+noednsopt` clears the EDNS options to be sent.

+[no]expire This toggles sending of an EDNS Expire option.

+[no]nsid This toggles inclusion of an EDNS name server ID request when sending a query.

+[no]recurse This toggles the setting of the RD (recursion desired) bit in the query. This bit is set by default, which means `mdig` normally sends recursive queries.

+retry=T This sets the number of times to retry UDP queries to server to `T` instead of the default, 2. Unlike `+tries`, this does not include the initial query.

+[no]subnet=addr[/prefix-length] This sends [or does not send] an EDNS Client Subnet option with the specified IP address or network prefix.

mdig +subnet=0.0.0.0/0, or simply mdig +subnet=0 This sends an EDNS client-subnet option with an empty address and a source prefix-length of zero, which signals a resolver that the client's address information must *not* be used when resolving this query.

+timeout=T This sets the timeout for a query to `T` seconds. The default timeout is 5 seconds for UDP transport and 10 for TCP. An attempt to set `T` to less than 1 results in a query timeout of 1 second being applied.

+tries=T This sets the number of times to try UDP queries to server to `T` instead of the default, 3. If `T` is less than or equal to zero, the number of tries is silently rounded up to 1.

+udptimeout=T This sets the timeout between UDP query retries to `T`.

+[no]unknownformat This prints [or does not print] all RDATA in unknown RR-type presentation format (see RFC 3597). The default is to print RDATA for known types in the type's presentation format.

+[no]yaml This toggles printing of the responses in a detailed YAML format.

+[no]zflag This sets [or does not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

12.20.6 See Also

dig(1), RFC 1035.

12.21 named-checkconf - named configuration file syntax checking tool

12.21.1 Synopsis

named-checkconf [**-chjlvz**] [**-p** [**-x**]] [**-t** directory] {filename}

12.21.2 Description

`named-checkconf` checks the syntax, but not the semantics, of a `named` configuration file. The file, along with all files included by it, is parsed and checked for syntax errors. If no file is specified, `/etc/named.conf` is read by default.

Note: files that `named` reads in separate parser contexts, such as `rndc.key` and `bind.keys`, are not automatically read by `named-checkconf`. Configuration errors in these files may cause `named` to fail to run, even if `named-checkconf` was successful. However, `named-checkconf` can be run on these files explicitly.

12.21.3 Options

-h This option prints the usage summary and exits.

-j When loading a zonefile, this option instructs `named` to read the journal if it exists.

-l This option lists all the configured zones. Each line of output contains the zone name, class (e.g. IN), view, and type (e.g. primary or secondary).

-c This option specifies that only the "core" configuration should be checked. This suppresses the loading of plugin modules, and causes all parameters to `plugin` statements to be ignored.

-i This option ignores warnings on deprecated options.

-p This option prints out the `named.conf` and included files in canonical form if no errors were detected. See also the `-x` option.

-t directory This option instructs `named` to chroot to `directory`, so that `include` directives in the configuration file are processed as if run by a similarly chrooted `named`.

–v This option prints the version of the `named-checkconf` program and exits.

–x When printing the configuration files in canonical form, this option obscures shared secrets by replacing them with strings of question marks (`?`). This allows the contents of `named.conf` and related files to be shared - for example, when submitting bug reports - without compromising private data. This option cannot be used without `-p`.

–z This option performs a test load of all zones of type `primary` found in `named.conf`.

filename This indicates the name of the configuration file to be checked. If not specified, it defaults to `/etc/named.conf`.

12.21.4 Return Values

`named-checkconf` returns an exit status of 1 if errors were detected and 0 otherwise.

12.21.5 See Also

named(8), *named-checkzone(8)*, BIND 9 Administrator Reference Manual.

12.22 named-checkzone - zone file validation tool

12.22.1 Synopsis

named–checkzone [-**d**] [-**h**] [-**j**] [-**q**] [-**v**] [-**c** class] [-**f** format] [-**F** format] [-**J** filename] [-**i** mode] [-**k** mode] [-**m** mode] [-**M** mode] [-**n** mode] [-**l** ttl] [-**L** serial] [-**o** filename] [-**r** mode] [-**s** style] [-**S** mode] [-**t** directory] [-**T** mode] [-**w** directory] [-**D**] [-**W** mode] {zonename} {filename}

12.22.2 Description

`named-checkzone` checks the syntax and integrity of a zone file. It performs the same checks as `named` does when loading a zone. This makes `named-checkzone` useful for checking zone files before configuring them into a name server.

12.22.3 Options

–d This option enables debugging.

–h This option prints the usage summary and exits.

–q This option sets quiet mode, which only sets an exit code to indicate successful or failed completion.

–v This option prints the version of the `named-checkzone` program and exits.

–j When loading a zone file, this option tells `named` to read the journal if it exists. The journal file name is assumed to be the zone file name with the string `.jnl` appended.

–J filename When loading the zone file, this option tells `named` to read the journal from the given file, if it exists. This implies `-j`.

–c class This option specifies the class of the zone. If not specified, `IN` is assumed.

−i mode This option performs post-load zone integrity checks. Possible modes are `full` (the default), `full-sibling`, `local`, `local-sibling`, and `none`.

Mode `full` checks that MX records refer to A or AAAA records (both in-zone and out-of-zone hostnames). Mode `local` only checks MX records which refer to in-zone hostnames.

Mode `full` checks that SRV records refer to A or AAAA records (both in-zone and out-of-zone hostnames). Mode `local` only checks SRV records which refer to in-zone hostnames.

Mode `full` checks that delegation NS records refer to A or AAAA records (both in-zone and out-of-zone hostnames). It also checks that glue address records in the zone match those advertised by the child. Mode `local` only checks NS records which refer to in-zone hostnames or verifies that some required glue exists, i.e., when the name server is in a child zone.

Modes `full-sibling` and `local-sibling` disable sibling glue checks, but are otherwise the same as `full` and `local`, respectively.

Mode `none` disables the checks.

−f format This option specifies the format of the zone file. Possible formats are `text` (the default), `raw`, and `map`.

−F format This option specifies the format of the output file specified. For `named-checkzone`, this does not have any effect unless it dumps the zone contents.

Possible formats are `text` (the default), which is the standard textual representation of the zone, and `map`, `raw`, and `raw=N`, which store the zone in a binary format for rapid loading by `named`. `raw=N` specifies the format version of the raw zone file: if `N` is 0, the raw file can be read by any version of `named`; if `N` is 1, the file can only be read by release 9.9.0 or higher. The default is 1.

−k mode This option performs `check-names` checks with the specified failure mode. Possible modes are `fail`, `warn` (the default), and `ignore`.

−l ttl This option sets a maximum permissible TTL for the input file. Any record with a TTL higher than this value causes the zone to be rejected. This is similar to using the `max-zone-ttl` option in `named.conf`.

−L serial When compiling a zone to `raw` or `map` format, this option sets the "source serial" value in the header to the specified serial number. This is expected to be used primarily for testing purposes.

−m mode This option specifies whether MX records should be checked to see if they are addresses. Possible modes are `fail`, `warn` (the default), and `ignore`.

−M mode This option checks whether a MX record refers to a CNAME. Possible modes are `fail`, `warn` (the default), and `ignore`.

−n mode This option specifies whether NS records should be checked to see if they are addresses. Possible modes are `fail`, `warn` (the default), and `ignore`.

−o filename This option writes the zone output to `filename`. If `filename` is −, then the zone output is written to standard output.

−r mode This option checks for records that are treated as different by DNSSEC but are semantically equal in plain DNS. Possible modes are `fail`, `warn` (the default), and `ignore`.

−s style This option specifies the style of the dumped zone file. Possible styles are `full` (the default) and `relative`. The `full` format is most suitable for processing automatically by a separate script. The relative format is more human-readable and is thus suitable for editing by hand. This does not have any effect unless it dumps the zone contents. It also does not have any meaning if the output format is not text.

−S mode This option checks whether an SRV record refers to a CNAME. Possible modes are `fail`, `warn` (the default), and `ignore`.

−t directory This option tells `named` to chroot to `directory`, so that `include` directives in the configuration file are processed as if run by a similarly chrooted `named`.

–T mode This option checks whether Sender Policy Framework (SPF) records exist and issues a warning if an SPF-formatted TXT record is not also present. Possible modes are `warn` (the default) and `ignore`.

–w directory This option instructs `named` to chdir to `directory`, so that relative filenames in master file `$INCLUDE` directives work. This is similar to the directory clause in `named.conf`.

–D This option dumps the zone file in canonical format.

–W mode This option specifies whether to check for non-terminal wildcards. Non-terminal wildcards are almost always the result of a failure to understand the wildcard matching algorithm ([RFC 4592](#)). Possible modes are `warn` (the default) and `ignore`.

zonename This indicates the domain name of the zone being checked.

filename This is the name of the zone file.

12.22.4 Return Values

`named-checkzone` returns an exit status of 1 if errors were detected and 0 otherwise.

12.22.5 See Also

named(8), *named-checkconf(8)*, *named-compilezone(8)*, [RFC 1035](#), BIND 9 Administrator Reference Manual.

12.23 named-compilezone - zone file converting tool

12.23.1 Synopsis

named–compilezone [**-d**] [**-h**] [**-j**] [**-q**] [**-v**] [**-c** class] [**-f** format] [**-F** format] [**-J** filename] [**-i** mode] [**-k** mode] [**-m** mode] [**-M** mode] [**-n** mode] [**-l** ttl] [**-L** serial] [**-r** mode] [**-s** style] [**-S** mode] [**-t** directory] [**-T** mode] [**-w** directory] [**-D**] [**-W** mode] {**-o** filename} {zonename} {filename}

12.23.2 Description

`named-compilezone` checks the syntax and integrity of a zone file, and dumps the zone contents to a specified file in a specified format. It applies strict check levels by default, since the dump output is used as an actual zone file loaded by `named`. When manually specified otherwise, the check levels must at least be as strict as those specified in the `named` configuration file.

12.23.3 Options

–d This option enables debugging.

–h This option prints the usage summary and exits.

–q This option sets quiet mode, which only sets an exit code to indicate successful or failed completion.

–v This option prints the version of the `named-checkzone` program and exits.

–j When loading a zone file, this option tells `named` to read the journal if it exists. The journal file name is assumed to be the zone file name with the string `.jnl` appended.

–J filename When loading the zone file, this option tells `named` to read the journal from the given file, if it exists. This implies `-j`.

–c class This option specifies the class of the zone. If not specified, `IN` is assumed.

–i mode This option performs post-load zone integrity checks. Possible modes are `full` (the default), `full-sibling`, `local`, `local-sibling`, and `none`.

> Mode `full` checks that MX records refer to A or AAAA records (both in-zone and out-of-zone hostnames). Mode `local` only checks MX records which refer to in-zone hostnames.

> Mode `full` checks that SRV records refer to A or AAAA records (both in-zone and out-of-zone hostnames). Mode `local` only checks SRV records which refer to in-zone hostnames.

> Mode `full` checks that delegation NS records refer to A or AAAA records (both in-zone and out-of-zone hostnames). It also checks that glue address records in the zone match those advertised by the child. Mode `local` only checks NS records which refer to in-zone hostnames or verifies that some required glue exists, i.e., when the name server is in a child zone.

> Modes `full-sibling` and `local-sibling` disable sibling glue checks, but are otherwise the same as `full` and `local`, respectively.

> Mode `none` disables the checks.

–f format This option specifies the format of the zone file. Possible formats are `text` (the default), `raw`, and `map`.

–F format This option specifies the format of the output file specified. For `named-checkzone`, this does not have any effect unless it dumps the zone contents.

> Possible formats are `text` (the default), which is the standard textual representation of the zone, and `map`, `raw`, and `raw=N`, which store the zone in a binary format for rapid loading by `named`. `raw=N` specifies the format version of the raw zone file: if `N` is 0, the raw file can be read by any version of `named`; if `N` is 1, the file can only be read by release 9.9.0 or higher. The default is 1.

–k mode This option performs `check-names` checks with the specified failure mode. Possible modes are `fail` (the default), `warn`, and `ignore`.

–l ttl This option sets a maximum permissible TTL for the input file. Any record with a TTL higher than this value causes the zone to be rejected. This is similar to using the `max-zone-ttl` option in `named.conf`.

–L serial When compiling a zone to `raw` or `map` format, this option sets the "source serial" value in the header to the specified serial number. This is expected to be used primarily for testing purposes.

–m mode This option specifies whether MX records should be checked to see if they are addresses. Possible modes are `fail`, `warn` (the default), and `ignore`.

–M mode This option checks whether a MX record refers to a CNAME. Possible modes are `fail`, `warn` (the default), and `ignore`.

–n mode This option specifies whether NS records should be checked to see if they are addresses. Possible modes are `fail` (the default), `warn`, and `ignore`.

–o filename This option writes the zone output to `filename`. If `filename` is –, then the zone output is written to standard output. This is mandatory for `named-compilezone`.

–r mode This option checks for records that are treated as different by DNSSEC but are semantically equal in plain DNS. Possible modes are `fail`, `warn` (the default), and `ignore`.

–s style This option specifies the style of the dumped zone file. Possible styles are `full` (the default) and `relative`. The `full` format is most suitable for processing automatically by a separate script. The relative format is more human-readable and is thus suitable for editing by hand.

–S mode This option checks whether an SRV record refers to a CNAME. Possible modes are `fail`, `warn` (the default), and `ignore`.

-t directory This option tells `named` to chroot to `directory`, so that `include` directives in the configuration file are processed as if run by a similarly chrooted `named`.

-T mode This option checks whether Sender Policy Framework (SPF) records exist and issues a warning if an SPF-formatted TXT record is not also present. Possible modes are `warn` (the default) and `ignore`.

-w directory This option instructs `named` to chdir to `directory`, so that relative filenames in master file `$IN-CLUDE` directives work. This is similar to the directory clause in `named.conf`.

-D This option dumps the zone file in canonical format. This is always enabled for `named-compilezone`.

-W mode This option specifies whether to check for non-terminal wildcards. Non-terminal wildcards are almost always the result of a failure to understand the wildcard matching algorithm (**RFC 4592**). Possible modes are `warn` (the default) and `ignore`.

zonename This indicates the domain name of the zone being checked.

filename This is the name of the zone file.

12.23.4 Return Values

`named-compilezone` returns an exit status of 1 if errors were detected and 0 otherwise.

12.23.5 See Also

named(8), *named-checkconf(8)*, *named-checkzone(8)*, **RFC 1035**, BIND 9 Administrator Reference Manual.

12.24 named-journalprint - print zone journal in human-readable form

12.24.1 Synopsis

named-journalprint [-c serial] [**-dux**] {journal}

12.24.2 Description

`named-journalprint` scans the contents of a zone journal file, printing it in a human-readable form, or, optionally, converting it to a different journal file format.

Journal files are automatically created by `named` when changes are made to dynamic zones (e.g., by `nsupdate`). They record each addition or deletion of a resource record, in binary format, allowing the changes to be re-applied to the zone when the server is restarted after a shutdown or crash. By default, the name of the journal file is formed by appending the extension `.jnl` to the name of the corresponding zone file.

`named-journalprint` converts the contents of a given journal file into a human-readable text format. Each line begins with `add` or `del`, to indicate whether the record was added or deleted, and continues with the resource record in master-file format.

The `-c` (compact) option provides a mechanism to reduce the size of a journal by removing (most/all) transactions prior to the specified serial number. Note: this option *must not* be used while `named` is running, and can cause data loss if the zone file has not been updated to contain the data being removed from the journal. Use with extreme caution.

The −x option causes additional data about the journal file to be printed at the beginning of the output and before each group of changes.

The −u (upgrade) and −d (downgrade) options recreate the journal file with a modified format version. The existing journal file is replaced. −d writes out the journal in the format used by versions of BIND up to 9.16.11; −u writes it out in the format used by versions since 9.16.13. (9.16.12 is omitted due to a journal-formatting bug in that release.) Note that these options *must not* be used while named is running.

12.24.3 See Also

named(8), *nsupdate(1)*, BIND 9 Administrator Reference Manual.

12.25 named-nzd2nzf - convert an NZD database to NZF text format

12.25.1 Synopsis

named−nzd2nzf {filename}

12.25.2 Description

named−nzd2nzf converts an NZD database to NZF format and prints it to standard output. This can be used to review the configuration of zones that were added to named via rndc addzone. It can also be used to restore the old file format when rolling back from a newer version of BIND to an older version.

12.25.3 Arguments

filename This is the name of the .nzd file whose contents should be printed.

12.25.4 See Also

BIND 9 Administrator Reference Manual.

12.26 named-rrchecker - syntax checker for individual DNS resource records

12.26.1 Synopsis

named−rrchecker [**-h**] [**-o** origin] [**-p**] [**-u**] [**-C**] [**-T**] [**-P**]

12.26.2 Description

`named-rrchecker` reads a individual DNS resource record from standard input and checks whether it is syntactically correct.

12.26.3 Options

-h This option prints out the help menu.

-o `origin` This option specifies the origin to be used when interpreting the record.

-p This option prints out the resulting record in canonical form. If there is no canonical form defined, the record is printed in unknown record format.

-u This option prints out the resulting record in unknown record form.

-C, -T, and -P These options print out the known class, standard type, and private type mnemonics, respectively.

12.26.4 See Also

RFC 1034, RFC 1035, *named(8)*.

12.27 named.conf - configuration file for named

12.27.1 Synopsis

named.conf

12.27.2 Description

`named.conf` is the configuration file for `named`. Statements are enclosed in braces and terminated with a semi-colon. Clauses in the statements are also semi-colon terminated. The usual comment styles are supported:

C style: /* */

 C++ style: // to end of line

Unix style: # to end of line

ACL

```
acl string { address_match_element; ... };
```

CONTROLS

```
controls {
      inet ( ipv4_address | ipv6_address |
          * ) [ port ( integer | * ) ] allow
          { address_match_element; ... } [
          keys { string; ... } ] [ read-only
          boolean ];
      unix quoted_string perm integer
          owner integer group integer [
          keys { string; ... } ] [ read-only
          boolean ];
};
```

DLZ

```
dlz string {
      database string;
      search boolean;
};
```

DNSSEC-POLICY

```
dnssec-policy string {
      dnskey-ttl duration;
      keys { ( csk | ksk | zsk ) [ ( key-directory ) ] lifetime
          duration_or_unlimited algorithm string [ integer ]; ... };
      max-zone-ttl duration;
      nsec3param [ iterations integer ] [ optout boolean ] [
          salt-length integer ];
      parent-ds-ttl duration;
      parent-propagation-delay duration;
      publish-safety duration;
      purge-keys duration;
      retire-safety duration;
      signatures-refresh duration;
      signatures-validity duration;
      signatures-validity-dnskey duration;
      zone-propagation-delay duration;
};
```

DYNDB

```
dyndb string quoted_string {
    unspecified-text };
```

KEY

```
key string {
      algorithm string;
      secret string;
};
```

LOGGING

```
logging {
      category string { string; ... };
      channel string {
              buffered boolean;
              file quoted_string [ versions ( unlimited | integer ) ]
                  [ size size ] [ suffix ( increment | timestamp ) ];
              null;
              print-category boolean;
              print-severity boolean;
              print-time ( iso8601 | iso8601-utc | local | boolean );
              severity log_severity;
              stderr;
              syslog [ syslog_facility ];
      };
};
```

MANAGED-KEYS

See DNSSEC-KEYS.

```
managed-keys { string ( static-key
    | initial-key | static-ds |
    initial-ds ) integer integer
    integer quoted_string; ... };, deprecated
```

MASTERS

```
masters string [ port integer ] [ dscp
    integer ] { ( remote-servers |
    ipv4_address [ port integer ] |
    ipv6_address [ port integer ] ) [ key
    string ]; ... };
```

OPTIONS

```
options {
    allow-new-zones boolean;
    allow-notify { address_match_element; ... };
    allow-query { address_match_element; ... };
    allow-query-cache { address_match_element; ... };
    allow-query-cache-on { address_match_element; ... };
    allow-query-on { address_match_element; ... };
    allow-recursion { address_match_element; ... };
    allow-recursion-on { address_match_element; ... };
    allow-transfer { address_match_element; ... };
    allow-update { address_match_element; ... };
    allow-update-forwarding { address_match_element; ... };
    also-notify [ port integer ] [ dscp integer ] { (
        remote-servers | ipv4_address [ port integer ] |
        ipv6_address [ port integer ] ) [ key string ]; ... };
    alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
    alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
        * ) ] [ dscp integer ];
    answer-cookie boolean;
    attach-cache string;
    auth-nxdomain boolean; // default changed
    auto-dnssec ( allow | maintain | off );
    automatic-interface-scan boolean;
    avoid-v4-udp-ports { portrange; ... };
    avoid-v6-udp-ports { portrange; ... };
    bindkeys-file quoted_string;
    blackhole { address_match_element; ... };
    cache-file quoted_string;// deprecated
    catalog-zones { zone string [ default-masters [ port integer ]
        [ dscp integer ] { ( remote-servers | ipv4_address [ port
        integer ] | ipv6_address [ port integer ] ) [ key
        string ]; ... } ] [ zone-directory quoted_string ] [
        in-memory boolean ] [ min-update-interval duration ]; ... };
    check-dup-records ( fail | warn | ignore );
    check-integrity boolean;
    check-mx ( fail | warn | ignore );
    check-mx-cname ( fail | warn | ignore );
    check-names ( primary | master |
        secondary | slave | response ) (
        fail | warn | ignore );
    check-sibling boolean;
    check-spf ( warn | ignore );
    check-srv-cname ( fail | warn | ignore );
    check-wildcard boolean;
    clients-per-query integer;
    cookie-algorithm ( aes | siphash24 );
    cookie-secret string;
    coresize ( default | unlimited | sizeval );
    datasize ( default | unlimited | sizeval );
    deny-answer-addresses { address_match_element; ... } [
        except-from { string; ... } ];
    deny-answer-aliases { string; ... } [ except-from { string; ...
        } ];
    dialup ( notify | notify-passive | passive | refresh | boolean );
```

(continues on next page)

```
    directory quoted_string;
    disable-algorithms string { string;
        ... };
    disable-ds-digests string { string;
        ... };
    disable-empty-zone string;
    dns64 netprefix {
            break-dnssec boolean;
            clients { address_match_element; ... };
            exclude { address_match_element; ... };
            mapped { address_match_element; ... };
            recursive-only boolean;
            suffix ipv6_address;
    };
    dns64-contact string;
    dns64-server string;
    dnskey-sig-validity integer;
    dnsrps-enable boolean;
    dnsrps-options { unspecified-text };
    dnssec-accept-expired boolean;
    dnssec-dnskey-kskonly boolean;
    dnssec-loadkeys-interval integer;
    dnssec-must-be-secure string boolean;
    dnssec-policy string;
    dnssec-secure-to-insecure boolean;
    dnssec-update-mode ( maintain | no-resign );
    dnssec-validation ( yes | no | auto );
    dnstap { ( all | auth | client | forwarder | resolver | update ) [
        ( query | response ) ]; ... };
    dnstap-identity ( quoted_string | none | hostname );
    dnstap-output ( file | unix ) quoted_string [ size ( unlimited |
        size ) ] [ versions ( unlimited | integer ) ] [ suffix (
        increment | timestamp ) ];
    dnstap-version ( quoted_string | none );
    dscp integer;
    dual-stack-servers [ port integer ] { ( quoted_string [ port
        integer ] [ dscp integer ] | ipv4_address [ port
        integer ] [ dscp integer ] | ipv6_address [ port
        integer ] [ dscp integer ] ); ... };
    dump-file quoted_string;
    edns-udp-size integer;
    empty-contact string;
    empty-server string;
    empty-zones-enable boolean;
    fetch-quota-params integer fixedpoint fixedpoint fixedpoint;
    fetches-per-server integer [ ( drop | fail ) ];
    fetches-per-zone integer [ ( drop | fail ) ];
    files ( default | unlimited | sizeval );
    flush-zones-on-shutdown boolean;
    forward ( first | only );
    forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
        | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
    fstrm-set-buffer-hint integer;
    fstrm-set-flush-timeout integer;
    fstrm-set-input-queue-size integer;
    fstrm-set-output-notify-threshold integer;
    fstrm-set-output-queue-model ( mpsc | spsc );
```

```
        fstrm-set-output-queue-size integer;
        fstrm-set-reopen-interval duration;
        geoip-directory ( quoted_string | none );
        glue-cache boolean;
        heartbeat-interval integer;
        hostname ( quoted_string | none );
        interface-interval duration;
        ixfr-from-differences ( primary | master | secondary | slave |
            boolean );
        keep-response-order { address_match_element; ... };
        key-directory quoted_string;
        lame-ttl duration;
        listen-on [ port integer ] [ dscp
            integer ] {
            address_match_element; ... };
        listen-on-v6 [ port integer ] [ dscp
            integer ] {
            address_match_element; ... };
        lmdb-mapsize sizeval;
        lock-file ( quoted_string | none );
        managed-keys-directory quoted_string;
        masterfile-format ( map | raw | text );
        masterfile-style ( full | relative );
        match-mapped-addresses boolean;
        max-cache-size ( default | unlimited | sizeval | percentage );
        max-cache-ttl duration;
        max-clients-per-query integer;
        max-ixfr-ratio ( unlimited | percentage );
        max-journal-size ( default | unlimited | sizeval );
        max-ncache-ttl duration;
        max-records integer;
        max-recursion-depth integer;
        max-recursion-queries integer;
        max-refresh-time integer;
        max-retry-time integer;
        max-rsa-exponent-size integer;
        max-stale-ttl duration;
        max-transfer-idle-in integer;
        max-transfer-idle-out integer;
        max-transfer-time-in integer;
        max-transfer-time-out integer;
        max-udp-size integer;
        max-zone-ttl ( unlimited | duration );
        memstatistics boolean;
        memstatistics-file quoted_string;
        message-compression boolean;
        min-cache-ttl duration;
        min-ncache-ttl duration;
        min-refresh-time integer;
        min-retry-time integer;
        minimal-any boolean;
        minimal-responses ( no-auth | no-auth-recursive | boolean );
        multi-master boolean;
        new-zones-directory quoted_string;
        no-case-compress { address_match_element; ... };
        nocookie-udp-size integer;
        notify ( explicit | master-only | primary-only | boolean );
```

```
    notify-delay integer;
    notify-rate integer;
    notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
        dscp integer ];
    notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
        [ dscp integer ];
    notify-to-soa boolean;
    nta-lifetime duration;
    nta-recheck duration;
    nxdomain-redirect string;
    parental-source ( ipv4_address | * ) [ port ( integer | * ) ] [
        dscp integer ];
    parental-source-v6 ( ipv6_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
    pid-file ( quoted_string | none );
    port integer;
    preferred-glue string;
    prefetch integer [ integer ];
    provide-ixfr boolean;
    qname-minimization ( strict | relaxed | disabled | off );
    query-source ( ( [ address ] ( ipv4_address | * ) [ port (
        integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
        port ( integer | * ) ) ) [ dscp integer ];
    query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
        integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
        port ( integer | * ) ) ) [ dscp integer ];
    querylog boolean;
    random-device ( quoted_string | none );
    rate-limit {
            all-per-second integer;
            errors-per-second integer;
            exempt-clients { address_match_element; ... };
            ipv4-prefix-length integer;
            ipv6-prefix-length integer;
            log-only boolean;
            max-table-size integer;
            min-table-size integer;
            nodata-per-second integer;
            nxdomains-per-second integer;
            qps-scale integer;
            referrals-per-second integer;
            responses-per-second integer;
            slip integer;
            window integer;
    };
    recursing-file quoted_string;
    recursion boolean;
    recursive-clients integer;
    request-expire boolean;
    request-ixfr boolean;
    request-nsid boolean;
    require-server-cookie boolean;
    reserved-sockets integer;
    resolver-nonbackoff-tries integer;
    resolver-query-timeout integer;
    resolver-retry-interval integer;
    response-padding { address_match_element; ... } block-size
```

```
            integer;
    response-policy { zone string [ add-soa boolean ] [ log
        boolean ] [ max-policy-ttl duration ] [ min-update-interval
        duration ] [ policy ( cname | disabled | drop | given | no-op
        | nodata | nxdomain | passthru | tcp-only quoted_string ) ] [
        recursive-only boolean ] [ nsip-enable boolean ] [
        nsdname-enable boolean ]; ... } [ add-soa boolean ] [
        break-dnssec boolean ] [ max-policy-ttl duration ] [
        min-update-interval duration ] [ min-ns-dots integer ] [
        nsip-wait-recurse boolean ] [ qname-wait-recurse boolean ]
        [ recursive-only boolean ] [ nsip-enable boolean ] [
        nsdname-enable boolean ] [ dnsrps-enable boolean ] [
        dnsrps-options { unspecified-text } ];
    reuseport boolean;
    root-delegation-only [ exclude { string; ... } ];
    root-key-sentinel boolean;
    rrset-order { [ class string ] [ type string ] [ name
        quoted_string ] string string; ... };
    secroots-file quoted_string;
    send-cookie boolean;
    serial-query-rate integer;
    serial-update-method ( date | increment | unixtime );
    server-id ( quoted_string | none | hostname );
    servfail-ttl duration;
    session-keyalg string;
    session-keyfile ( quoted_string | none );
    session-keyname string;
    sig-signing-nodes integer;
    sig-signing-signatures integer;
    sig-signing-type integer;
    sig-validity-interval integer [ integer ];
    sortlist { address_match_element; ... };
    stacksize ( default | unlimited | sizeval );
    stale-answer-client-timeout ( disabled | off | integer );
    stale-answer-enable boolean;
    stale-answer-ttl duration;
    stale-cache-enable boolean;
    stale-refresh-time duration;
    startup-notify-rate integer;
    statistics-file quoted_string;
    synth-from-dnssec boolean;
    tcp-advertised-timeout integer;
    tcp-clients integer;
    tcp-idle-timeout integer;
    tcp-initial-timeout integer;
    tcp-keepalive-timeout integer;
    tcp-listen-queue integer;
    tkey-dhkey quoted_string integer;
    tkey-domain quoted_string;
    tkey-gssapi-credential quoted_string;
    tkey-gssapi-keytab quoted_string;
    transfer-format ( many-answers | one-answer );
    transfer-message-size integer;
    transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
        dscp integer ];
    transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
```

```
        transfers-in integer;
        transfers-out integer;
        transfers-per-ns integer;
        trust-anchor-telemetry boolean; // experimental
        try-tcp-refresh boolean;
        update-check-ksk boolean;
        use-alt-transfer-source boolean;
        use-v4-udp-ports { portrange; ... };
        use-v6-udp-ports { portrange; ... };
        v6-bias integer;
        validate-except { string; ... };
        version ( quoted_string | none );
        zero-no-soa-ttl boolean;
        zero-no-soa-ttl-cache boolean;
        zone-statistics ( full | terse | none | boolean );
};
```

PARENTAL-AGENTS

```
parental-agents string [ port integer ] [
    dscp integer ] { ( remote-servers |
    ipv4_address [ port integer ] |
    ipv6_address [ port integer ] ) [ key
    string ]; ... };
```

PLUGIN

```
plugin ( query ) string [ { unspecified-text
    } ];
```

PRIMARIES

```
primaries string [ port integer ] [ dscp
    integer ] { ( remote-servers |
    ipv4_address [ port integer ] |
    ipv6_address [ port integer ] ) [ key
    string ]; ... };
```

SERVER

```
server netprefix {
    bogus boolean;
    edns boolean;
    edns-udp-size integer;
    edns-version integer;
    keys server_key;
    max-udp-size integer;
    notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
        dscp integer ];
```

```
        notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
            [ dscp integer ];
        padding integer;
        provide-ixfr boolean;
        query-source ( ( [ address ] ( ipv4_address | * ) [ port (
            integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
            port ( integer | * ) ) ) [ dscp integer ];
        query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
            integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
            port ( integer | * ) ) ) [ dscp integer ];
        request-expire boolean;
        request-ixfr boolean;
        request-nsid boolean;
        send-cookie boolean;
        tcp-keepalive boolean;
        tcp-only boolean;
        transfer-format ( many-answers | one-answer );
        transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
            dscp integer ];
        transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
            ] [ dscp integer ];
        transfers integer;
};
```

STATISTICS-CHANNELS

```
statistics-channels {
        inet ( ipv4_address | ipv6_address |
            * ) [ port ( integer | * ) ] [
            allow { address_match_element; ...
            } ];
};
```

TRUST-ANCHORS

```
trust-anchors { string ( static-key |
    initial-key | static-ds | initial-ds )
    integer integer integer
    quoted_string; ... };
```

TRUSTED-KEYS

Deprecated - see DNSSEC-KEYS.

```
trusted-keys { string integer
    integer integer
    quoted_string; ... };, deprecated
```

VIEW

```
view string [ class ] {
    allow-new-zones boolean;
    allow-notify { address_match_element; ... };
    allow-query { address_match_element; ... };
    allow-query-cache { address_match_element; ... };
    allow-query-cache-on { address_match_element; ... };
    allow-query-on { address_match_element; ... };
    allow-recursion { address_match_element; ... };
    allow-recursion-on { address_match_element; ... };
    allow-transfer { address_match_element; ... };
    allow-update { address_match_element; ... };
    allow-update-forwarding { address_match_element; ... };
    also-notify [ port integer ] [ dscp integer ] { (
        remote-servers | ipv4_address [ port integer ] |
        ipv6_address [ port integer ] ) [ key string ]; ... };
    alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
    alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
        * ) ] [ dscp integer ];
    attach-cache string;
    auth-nxdomain boolean; // default changed
    auto-dnssec ( allow | maintain | off );
    cache-file quoted_string;// deprecated
    catalog-zones { zone string [ default-masters [ port integer ]
        [ dscp integer ] { ( remote-servers | ipv4_address [ port
        integer ] | ipv6_address [ port integer ] ) [ key
        string ]; ... } ] [ zone-directory quoted_string ] [
        in-memory boolean ] [ min-update-interval duration ]; ... };
    check-dup-records ( fail | warn | ignore );
    check-integrity boolean;
    check-mx ( fail | warn | ignore );
    check-mx-cname ( fail | warn | ignore );
    check-names ( primary | master |
        secondary | slave | response ) (
        fail | warn | ignore );
    check-sibling boolean;
    check-spf ( warn | ignore );
    check-srv-cname ( fail | warn | ignore );
    check-wildcard boolean;
    clients-per-query integer;
    deny-answer-addresses { address_match_element; ... } [
        except-from { string; ... } ];
    deny-answer-aliases { string; ... } [ except-from { string; ...
        } ];
    dialup ( notify | notify-passive | passive | refresh | boolean );
    disable-algorithms string { string;
        ... };
    disable-ds-digests string { string;
        ... };
    disable-empty-zone string;
    dlz string {
            database string;
            search boolean;
    };
    dns64 netprefix {
```

(continues on next page)

```
                break-dnssec boolean;
                clients { address_match_element; ... };
                exclude { address_match_element; ... };
                mapped { address_match_element; ... };
                recursive-only boolean;
                suffix ipv6_address;
        };
        dns64-contact string;
        dns64-server string;
        dnskey-sig-validity integer;
        dnsrps-enable boolean;
        dnsrps-options { unspecified-text };
        dnssec-accept-expired boolean;
        dnssec-dnskey-kskonly boolean;
        dnssec-loadkeys-interval integer;
        dnssec-must-be-secure string boolean;
        dnssec-policy string;
        dnssec-secure-to-insecure boolean;
        dnssec-update-mode ( maintain | no-resign );
        dnssec-validation ( yes | no | auto );
        dnstap { ( all | auth | client | forwarder | resolver | update ) [
            ( query | response ) ]; ... };
        dual-stack-servers [ port integer ] { ( quoted_string [ port
            integer ] [ dscp integer ] | ipv4_address [ port
            integer ] [ dscp integer ] | ipv6_address [ port
            integer ] [ dscp integer ] ); ... };
        dyndb string quoted_string {
            unspecified-text };
        edns-udp-size integer;
        empty-contact string;
        empty-server string;
        empty-zones-enable boolean;
        fetch-quota-params integer fixedpoint fixedpoint fixedpoint;
        fetches-per-server integer [ ( drop | fail ) ];
        fetches-per-zone integer [ ( drop | fail ) ];
        forward ( first | only );
        forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
            | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
        glue-cache boolean;
        ixfr-from-differences ( primary | master | secondary | slave |
            boolean );
        key string {
                algorithm string;
                secret string;
        };
        key-directory quoted_string;
        lame-ttl duration;
        lmdb-mapsize sizeval;
        managed-keys { string (
            static-key | initial-key
            | static-ds | initial-ds
            ) integer integer
            integer
            quoted_string; ... };, deprecated
        masterfile-format ( map | raw | text );
        masterfile-style ( full | relative );
        match-clients { address_match_element; ... };
```

```
      match-destinations { address_match_element; ... };
      match-recursive-only boolean;
      max-cache-size ( default | unlimited | sizeval | percentage );
      max-cache-ttl duration;
      max-clients-per-query integer;
      max-ixfr-ratio ( unlimited | percentage );
      max-journal-size ( default | unlimited | sizeval );
      max-ncache-ttl duration;
      max-records integer;
      max-recursion-depth integer;
      max-recursion-queries integer;
      max-refresh-time integer;
      max-retry-time integer;
      max-stale-ttl duration;
      max-transfer-idle-in integer;
      max-transfer-idle-out integer;
      max-transfer-time-in integer;
      max-transfer-time-out integer;
      max-udp-size integer;
      max-zone-ttl ( unlimited | duration );
      message-compression boolean;
      min-cache-ttl duration;
      min-ncache-ttl duration;
      min-refresh-time integer;
      min-retry-time integer;
      minimal-any boolean;
      minimal-responses ( no-auth | no-auth-recursive | boolean );
      multi-master boolean;
      new-zones-directory quoted_string;
      no-case-compress { address_match_element; ... };
      nocookie-udp-size integer;
      notify ( explicit | master-only | primary-only | boolean );
      notify-delay integer;
      notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
          dscp integer ];
      notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
          [ dscp integer ];
      notify-to-soa boolean;
      nta-lifetime duration;
      nta-recheck duration;
      nxdomain-redirect string;
      parental-source ( ipv4_address | * ) [ port ( integer | * ) ] [
          dscp integer ];
      parental-source-v6 ( ipv6_address | * ) [ port ( integer | * )
          ] [ dscp integer ];
      plugin ( query ) string [ {
          unspecified-text } ];
      preferred-glue string;
      prefetch integer [ integer ];
      provide-ixfr boolean;
      qname-minimization ( strict | relaxed | disabled | off );
      query-source ( ( [ address ] ( ipv4_address | * ) [ port (
          integer | * ) ] ) | ( [ [ address ] ( ipv4_address | * ) ]
          port ( integer | * ) ) ) [ dscp integer ];
      query-source-v6 ( ( [ address ] ( ipv6_address | * ) [ port (
          integer | * ) ] ) | ( [ [ address ] ( ipv6_address | * ) ]
          port ( integer | * ) ) ) [ dscp integer ];
```

```
    rate-limit {
            all-per-second integer;
            errors-per-second integer;
            exempt-clients { address_match_element; ... };
            ipv4-prefix-length integer;
            ipv6-prefix-length integer;
            log-only boolean;
            max-table-size integer;
            min-table-size integer;
            nodata-per-second integer;
            nxdomains-per-second integer;
            qps-scale integer;
            referrals-per-second integer;
            responses-per-second integer;
            slip integer;
            window integer;
    };
    recursion boolean;
    request-expire boolean;
    request-ixfr boolean;
    request-nsid boolean;
    require-server-cookie boolean;
    resolver-nonbackoff-tries integer;
    resolver-query-timeout integer;
    resolver-retry-interval integer;
    response-padding { address_match_element; ... } block-size
        integer;
    response-policy { zone string [ add-soa boolean ] [ log
        boolean ] [ max-policy-ttl duration ] [ min-update-interval
        duration ] [ policy ( cname | disabled | drop | given | no-op
        | nodata | nxdomain | passthru | tcp-only quoted_string ) ] [
        recursive-only boolean ] [ nsip-enable boolean ] [
        nsdname-enable boolean ]; ... } [ add-soa boolean ] [
        break-dnssec boolean ] [ max-policy-ttl duration ] [
        min-update-interval duration ] [ min-ns-dots integer ] [
        nsip-wait-recurse boolean ] [ qname-wait-recurse boolean ]
        [ recursive-only boolean ] [ nsip-enable boolean ] [
        nsdname-enable boolean ] [ dnsrps-enable boolean ] [
        dnsrps-options { unspecified-text } ];
    root-delegation-only [ exclude { string; ... } ];
    root-key-sentinel boolean;
    rrset-order { [ class string ] [ type string ] [ name
        quoted_string ] string string; ... };
    send-cookie boolean;
    serial-update-method ( date | increment | unixtime );
    server netprefix {
            bogus boolean;
            edns boolean;
            edns-udp-size integer;
            edns-version integer;
            keys server_key;
            max-udp-size integer;
            notify-source ( ipv4_address | * ) [ port ( integer | *
                ) ] [ dscp integer ];
            notify-source-v6 ( ipv6_address | * ) [ port ( integer
                | * ) ] [ dscp integer ];
            padding integer;
```

```
        provide-ixfr boolean;
        query-source ( ( [ address ] ( ipv4_address | * ) [ port
            ( integer | * ) ] ) | ( [ [ address ] (
            ipv4_address | * ) ] port ( integer | * ) ) ) [
            dscp integer ];
        query-source-v6 ( ( [ address ] ( ipv6_address | * ) [
            port ( integer | * ) ] ) | ( [ [ address ] (
            ipv6_address | * ) ] port ( integer | * ) ) ) [
            dscp integer ];
        request-expire boolean;
        request-ixfr boolean;
        request-nsid boolean;
        send-cookie boolean;
        tcp-keepalive boolean;
        tcp-only boolean;
        transfer-format ( many-answers | one-answer );
        transfer-source ( ipv4_address | * ) [ port ( integer |
            * ) ] [ dscp integer ];
        transfer-source-v6 ( ipv6_address | * ) [ port (
            integer | * ) ] [ dscp integer ];
        transfers integer;
    };
    servfail-ttl duration;
    sig-signing-nodes integer;
    sig-signing-signatures integer;
    sig-signing-type integer;
    sig-validity-interval integer [ integer ];
    sortlist { address_match_element; ... };
    stale-answer-client-timeout ( disabled | off | integer );
    stale-answer-enable boolean;
    stale-answer-ttl duration;
    stale-cache-enable boolean;
    stale-refresh-time duration;
    synth-from-dnssec boolean;
    transfer-format ( many-answers | one-answer );
    transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
        dscp integer ];
    transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
    trust-anchor-telemetry boolean; // experimental
    trust-anchors { string ( static-key |
        initial-key | static-ds | initial-ds
        ) integer integer integer
        quoted_string; ... };
    trusted-keys { string
        integer integer
        integer
        quoted_string; ... };, deprecated
    try-tcp-refresh boolean;
    update-check-ksk boolean;
    use-alt-transfer-source boolean;
    v6-bias integer;
    validate-except { string; ... };
    zero-no-soa-ttl boolean;
    zero-no-soa-ttl-cache boolean;
    zone string [ class ] {
        allow-notify { address_match_element; ... };
```

```
            allow-query { address_match_element; ... };
            allow-query-on { address_match_element; ... };
            allow-transfer { address_match_element; ... };
            allow-update { address_match_element; ... };
            allow-update-forwarding { address_match_element; ... };
            also-notify [ port integer ] [ dscp integer ] { (
                remote-servers | ipv4_address [ port integer ] |
                ipv6_address [ port integer ] ) [ key string ];
                ... };
            alt-transfer-source ( ipv4_address | * ) [ port (
                integer | * ) ] [ dscp integer ];
            alt-transfer-source-v6 ( ipv6_address | * ) [ port (
                integer | * ) ] [ dscp integer ];
            auto-dnssec ( allow | maintain | off );
            check-dup-records ( fail | warn | ignore );
            check-integrity boolean;
            check-mx ( fail | warn | ignore );
            check-mx-cname ( fail | warn | ignore );
            check-names ( fail | warn | ignore );
            check-sibling boolean;
            check-spf ( warn | ignore );
            check-srv-cname ( fail | warn | ignore );
            check-wildcard boolean;
            database string;
            delegation-only boolean;
            dialup ( notify | notify-passive | passive | refresh |
                boolean );
            dlz string;
            dnskey-sig-validity integer;
            dnssec-dnskey-kskonly boolean;
            dnssec-loadkeys-interval integer;
            dnssec-policy string;
            dnssec-secure-to-insecure boolean;
            dnssec-update-mode ( maintain | no-resign );
            file quoted_string;
            forward ( first | only );
            forwarders [ port integer ] [ dscp integer ] { (
                ipv4_address | ipv6_address ) [ port integer ] [
                dscp integer ]; ... };
            in-view string;
            inline-signing boolean;
            ixfr-from-differences boolean;
            journal quoted_string;
            key-directory quoted_string;
            masterfile-format ( map | raw | text );
            masterfile-style ( full | relative );
            masters [ port integer ] [ dscp integer ] { (
                remote-servers | ipv4_address [ port integer ] |
                ipv6_address [ port integer ] ) [ key string ];
                ... };
            max-ixfr-ratio ( unlimited | percentage );
            max-journal-size ( default | unlimited | sizeval );
            max-records integer;
            max-refresh-time integer;
            max-retry-time integer;
            max-transfer-idle-in integer;
            max-transfer-idle-out integer;
```

```
              max-transfer-time-in integer;
              max-transfer-time-out integer;
              max-zone-ttl ( unlimited | duration );
              min-refresh-time integer;
              min-retry-time integer;
              multi-master boolean;
              notify ( explicit | master-only | primary-only | boolean );
              notify-delay integer;
              notify-source ( ipv4_address | * ) [ port ( integer | *
                  ) ] [ dscp integer ];
              notify-source-v6 ( ipv6_address | * ) [ port ( integer
                  | * ) ] [ dscp integer ];
              notify-to-soa boolean;
              parental-agents [ port integer ] [ dscp integer ] { (
                  remote-servers | ipv4_address [ port integer ] |
                  ipv6_address [ port integer ] ) [ key string ];
                  ... };
              parental-source ( ipv4_address | * ) [ port ( integer |
                  * ) ] [ dscp integer ];
              parental-source-v6 ( ipv6_address | * ) [ port (
                  integer | * ) ] [ dscp integer ];
              primaries [ port integer ] [ dscp integer ] { (
                  remote-servers | ipv4_address [ port integer ] |
                  ipv6_address [ port integer ] ) [ key string ];
                  ... };
              request-expire boolean;
              request-ixfr boolean;
              serial-update-method ( date | increment | unixtime );
              server-addresses { ( ipv4_address | ipv6_address ); ... };
              server-names { string; ... };
              sig-signing-nodes integer;
              sig-signing-signatures integer;
              sig-signing-type integer;
              sig-validity-interval integer [ integer ];
              transfer-source ( ipv4_address | * ) [ port ( integer |
                  * ) ] [ dscp integer ];
              transfer-source-v6 ( ipv6_address | * ) [ port (
                  integer | * ) ] [ dscp integer ];
              try-tcp-refresh boolean;
              type ( primary | master | secondary | slave | mirror |
                  delegation-only | forward | hint | redirect |
                  static-stub | stub );
              update-check-ksk boolean;
              update-policy ( local | { ( deny | grant ) string (
                  6to4-self | external | krb5-self | krb5-selfsub |
                  krb5-subdomain | ms-self | ms-selfsub | ms-subdomain |
                  name | self | selfsub | selfwild | subdomain | tcp-self
                  | wildcard | zonesub ) [ string ] rrtypelist; ... };
              use-alt-transfer-source boolean;
              zero-no-soa-ttl boolean;
              zone-statistics ( full | terse | none | boolean );
      };
      zone-statistics ( full | terse | none | boolean );
};
```

ZONE

```
zone string [ class ] {
    allow-notify { address_match_element; ... };
    allow-query { address_match_element; ... };
    allow-query-on { address_match_element; ... };
    allow-transfer { address_match_element; ... };
    allow-update { address_match_element; ... };
    allow-update-forwarding { address_match_element; ... };
    also-notify [ port integer ] [ dscp integer ] { (
        remote-servers | ipv4_address [ port integer ] |
        ipv6_address [ port integer ] ) [ key string ]; ... };
    alt-transfer-source ( ipv4_address | * ) [ port ( integer | * )
        ] [ dscp integer ];
    alt-transfer-source-v6 ( ipv6_address | * ) [ port ( integer |
        * ) ] [ dscp integer ];
    auto-dnssec ( allow | maintain | off );
    check-dup-records ( fail | warn | ignore );
    check-integrity boolean;
    check-mx ( fail | warn | ignore );
    check-mx-cname ( fail | warn | ignore );
    check-names ( fail | warn | ignore );
    check-sibling boolean;
    check-spf ( warn | ignore );
    check-srv-cname ( fail | warn | ignore );
    check-wildcard boolean;
    database string;
    delegation-only boolean;
    dialup ( notify | notify-passive | passive | refresh | boolean );
    dlz string;
    dnskey-sig-validity integer;
    dnssec-dnskey-kskonly boolean;
    dnssec-loadkeys-interval integer;
    dnssec-policy string;
    dnssec-secure-to-insecure boolean;
    dnssec-update-mode ( maintain | no-resign );
    file quoted_string;
    forward ( first | only );
    forwarders [ port integer ] [ dscp integer ] { ( ipv4_address
        | ipv6_address ) [ port integer ] [ dscp integer ]; ... };
    in-view string;
    inline-signing boolean;
    ixfr-from-differences boolean;
    journal quoted_string;
    key-directory quoted_string;
    masterfile-format ( map | raw | text );
    masterfile-style ( full | relative );
    masters [ port integer ] [ dscp integer ] { ( remote-servers
        | ipv4_address [ port integer ] | ipv6_address [ port
        integer ] ) [ key string ]; ... };
    max-ixfr-ratio ( unlimited | percentage );
    max-journal-size ( default | unlimited | sizeval );
    max-records integer;
    max-refresh-time integer;
    max-retry-time integer;
    max-transfer-idle-in integer;
    max-transfer-idle-out integer;
```

(continues on next page)

```
        max-transfer-time-in integer;
        max-transfer-time-out integer;
        max-zone-ttl ( unlimited | duration );
        min-refresh-time integer;
        min-retry-time integer;
        multi-master boolean;
        notify ( explicit | master-only | primary-only | boolean );
        notify-delay integer;
        notify-source ( ipv4_address | * ) [ port ( integer | * ) ] [
            dscp integer ];
        notify-source-v6 ( ipv6_address | * ) [ port ( integer | * ) ]
            [ dscp integer ];
        notify-to-soa boolean;
        parental-agents [ port integer ] [ dscp integer ] { (
            remote-servers | ipv4_address [ port integer ] |
            ipv6_address [ port integer ] ) [ key string ]; ... };
        parental-source ( ipv4_address | * ) [ port ( integer | * ) ] [
            dscp integer ];
        parental-source-v6 ( ipv6_address | * ) [ port ( integer | * )
            ] [ dscp integer ];
        primaries [ port integer ] [ dscp integer ] { (
            remote-servers | ipv4_address [ port integer ] |
            ipv6_address [ port integer ] ) [ key string ]; ... };
        request-expire boolean;
        request-ixfr boolean;
        serial-update-method ( date | increment | unixtime );
        server-addresses { ( ipv4_address | ipv6_address ); ... };
        server-names { string; ... };
        sig-signing-nodes integer;
        sig-signing-signatures integer;
        sig-signing-type integer;
        sig-validity-interval integer [ integer ];
        transfer-source ( ipv4_address | * ) [ port ( integer | * ) ] [
            dscp integer ];
        transfer-source-v6 ( ipv6_address | * ) [ port ( integer | * )
            ] [ dscp integer ];
        try-tcp-refresh boolean;
        type ( primary | master | secondary | slave | mirror |
            delegation-only | forward | hint | redirect | static-stub |
            stub );
        update-check-ksk boolean;
        update-policy ( local | { ( deny | grant ) string ( 6to4-self |
            external | krb5-self | krb5-selfsub | krb5-subdomain | ms-self
            | ms-selfsub | ms-subdomain | name | self | selfsub | selfwild
            | subdomain | tcp-self | wildcard | zonesub ) [ string ]
            rrtypelist; ... };
        use-alt-transfer-source boolean;
        zero-no-soa-ttl boolean;
        zone-statistics ( full | terse | none | boolean );
};
```

12.27.3 Files

`/etc/named.conf`

12.27.4 See Also

ddns-confgen(8), *named(8)*, *named-checkconf(8)*, *rndc(8)*, *rndc-confgen(8)*, BIND 9 Administrator Reference Manual.

12.28 named - Internet domain name server

12.28.1 Synopsis

named [[**-4**] | [**-6**]] [**-c** config-file] [**-d** debug-level] [**-D** string] [**-E** engine-name] [**-f**] [**-g**] [**-L** logfile] [**-M** option] [**-m** flag] [**-n** #cpus] [**-p** port] [**-s**] [**-S** #max-socks] [**-t** directory] [**-U** #listeners] [**-u** user] [**-v**] [**-V**] [**-X** lock-file] [**-x** cache-file]

12.28.2 Description

`named` is a Domain Name System (DNS) server, part of the BIND 9 distribution from ISC. For more information on the DNS, see **RFC 1033**, **RFC 1034**, and **RFC 1035**.

When invoked without arguments, `named` reads the default configuration file `/etc/named.conf`, reads any initial data, and listens for queries.

12.28.3 Options

-4 This option tells `named` to use only IPv4, even if the host machine is capable of IPv6. `-4` and `-6` are mutually exclusive.

-6 This option tells `named` to use only IPv6, even if the host machine is capable of IPv4. `-4` and `-6` are mutually exclusive.

-c config-file This option tells `named` to use `config-file` as its configuration file instead of the default, `/etc/named.conf`. To ensure that the configuration file can be reloaded after the server has changed its working directory due to to a possible `directory` option in the configuration file, `config-file` should be an absolute pathname.

-d debug-level This option sets the daemon's debug level to `debug-level`. Debugging traces from `named` become more verbose as the debug level increases.

-D string This option specifies a string that is used to identify a instance of `named` in a process listing. The contents of `string` are not examined.

-E engine-name When applicable, this option specifies the hardware to use for cryptographic operations, such as a secure key store used for signing.

When BIND 9 is built with OpenSSL, this needs to be set to the OpenSSL engine identifier that drives the cryptographic accelerator or hardware service module (usually `pkcs11`). When BIND is built with native PKCS#11 cryptography (`--enable-native-pkcs11`), it defaults to the path of the PKCS#11 provider library specified via `--with-pkcs11`.

-f This option runs the server in the foreground (i.e., do not daemonize).

-g This option runs the server in the foreground and forces all logging to `stderr`.

-L logfile This option sets the log to the file `logfile` by default, instead of the system log.

-M option This option sets the default memory context options. If set to `external`, the internal memory manager is bypassed in favor of system-provided memory allocation functions. If set to `fill`, blocks of memory are filled with tag values when allocated or freed, to assist debugging of memory problems. `nofill` disables this behavior, and is the default unless `named` has been compiled with developer options.

-m flag This option turns on memory usage debugging flags. Possible flags are `usage`, `trace`, `record`, `size`, and `mctx`. These correspond to the `ISC_MEM_DEBUGXXXX` flags described in `<isc/mem.h>`.

-n #cpus This option creates `#cpus` worker threads to take advantage of multiple CPUs. If not specified, `named` tries to determine the number of CPUs present and creates one thread per CPU. If it is unable to determine the number of CPUs, a single worker thread is created.

-p port This option listens for queries on `port`. If not specified, the default is port 53.

-s This option writes memory usage statistics to `stdout` on exit.

Note: This option is mainly of interest to BIND 9 developers and may be removed or changed in a future release.

-S #max-socks This option allows `named` to use up to `#max-socks` sockets. The default value is 21000 on systems built with default configuration options, and 4096 on systems built with `configure --with-tuning=small`.

> **Warning:** This option should be unnecessary for the vast majority of users. The use of this option could even be harmful, because the specified value may exceed the limitation of the underlying system API. It is therefore set only when the default configuration causes exhaustion of file descriptors and the operational environment is known to support the specified number of sockets. Note also that the actual maximum number is normally slightly fewer than the specified value, because `named` reserves some file descriptors for its internal use.

-t directory This option tells `named` to chroot to `directory` after processing the command-line arguments, but before reading the configuration file.

> **Warning:** This option should be used in conjunction with the `-u` option, as chrooting a process running as root doesn't enhance security on most systems; the way `chroot` is defined allows a process with root privileges to escape a chroot jail.

-U #listeners This option tells `named` the number of `#listeners` worker threads to listen on, for incoming UDP packets on each address. If not specified, `named` calculates a default value based on the number of detected CPUs: 1 for 1 CPU, and the number of detected CPUs minus one for machines with more than 1 CPU. This cannot be increased to a value higher than the number of CPUs. If `-n` has been set to a higher value than the number of detected CPUs, then `-U` may be increased as high as that value, but no higher. On Windows, the number of UDP listeners is hardwired to 1 and this option has no effect.

-u user This option sets the setuid to `user` after completing privileged operations, such as creating sockets that listen on privileged ports.

Note: On Linux, `named` uses the kernel's capability mechanism to drop all root privileges except the ability to `bind` to a privileged port and set process resource limits. Unfortunately, this means that the `-u` option only works when `named` is run on kernel 2.2.18 or later, or kernel 2.3.99-pre3 or later, since previous kernels did not allow privileges to be retained after `setuid`.

-v This option reports the version number and exits.

-V This option reports the version number and build options, and exits.

-X lock-file This option acquires a lock on the specified file at runtime; this helps to prevent duplicate `named` instances from running simultaneously. Use of this option overrides the `lock-file` option in `named.conf`. If set to `none`, the lock file check is disabled.

-x cache-file This option loads data from `cache-file` into the cache of the default view.

> **Warning:** This option must not be used in normal operations. It is only of interest to BIND 9 developers and may be removed or changed in a future release.

12.28.4 Signals

In routine operation, signals should not be used to control the nameserver; `rndc` should be used instead.

SIGHUP This signal forces a reload of the server.

SIGINT, SIGTERM These signals shut down the server.

The result of sending any other signals to the server is undefined.

12.28.5 Configuration

The `named` configuration file is too complex to describe in detail here. A complete description is provided in the BIND 9 Administrator Reference Manual.

`named` inherits the `umask` (file creation mode mask) from the parent process. If files created by `named`, such as journal files, need to have custom permissions, the `umask` should be set explicitly in the script used to start the `named` process.

12.28.6 Files

/etc/named.conf The default configuration file.

/var/run/named/named.pid The default process-id file.

12.28.7 See Also

RFC 1033, RFC 1034, RFC 1035, `named-checkconf(8)`, `named-checkzone(8)`, `rndc(8)`, `named.conf(5)`, BIND 9 Administrator Reference Manual.

12.29 nsec3hash - generate NSEC3 hash

12.29.1 Synopsis

nsec3hash {salt} {algorithm} {iterations} {domain}

nsec3hash -r {algorithm} {flags} {iterations} {salt} {domain}

12.29.2 Description

`nsec3hash` generates an NSEC3 hash based on a set of NSEC3 parameters. This can be used to check the validity of NSEC3 records in a signed zone.

If this command is invoked as `nsec3hash -r`, it takes arguments in order, matching the first four fields of an NSEC3 record followed by the domain name: `algorithm`, `flags`, `iterations`, `salt`, `domain`. This makes it convenient to copy and paste a portion of an NSEC3 or NSEC3PARAM record into a command line to confirm the correctness of an NSEC3 hash.

12.29.3 Arguments

salt This is the salt provided to the hash algorithm.

algorithm This is a number indicating the hash algorithm. Currently the only supported hash algorithm for NSEC3 is SHA-1, which is indicated by the number 1; consequently "1" is the only useful value for this argument.

flags This is provided for compatibility with NSEC3 record presentation format, but is ignored since the flags do not affect the hash.

iterations This is the number of additional times the hash should be performed.

domain This is the domain name to be hashed.

12.29.4 See Also

BIND 9 Administrator Reference Manual, **RFC 5155**.

12.30 nslookup - query Internet name servers interactively

12.30.1 Synopsis

nslookup [-option] [name | -] [server]

12.30.2 Description

`nslookup` is a program to query Internet domain name servers. `nslookup` has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode prints just the name and requested information for a host or domain.

12.30.3 Arguments

Interactive mode is entered in the following cases:

    a. when no arguments are given (the default name server is used);

    b. when the first argument is a hyphen (-) and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

Options can also be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, with an initial timeout of 10 seconds, type:

```
nslookup -query=hinfo  -timeout=10
```

The `-version` option causes `nslookup` to print the version number and immediately exit.

12.30.4 Interactive Commands

host [server] This command looks up information for `host` using the current default server or using `server`, if specified. If `host` is an Internet address and the query type is A or PTR, the name of the host is returned. If `host` is a name and does not have a trailing period (`.`), the search list is used to qualify the name.

> To look up a host not in the current domain, append a period to the name.

server domain|lserver domain These commands change the default server to `domain`; `lserver` uses the initial server to look up information about `domain`, while `server` uses the current default server. If an authoritative answer cannot be found, the names of servers that might have the answer are returned.

root This command is not implemented.

finger This command is not implemented.

ls This command is not implemented.

view This command is not implemented.

help This command is not implemented.

? This command is not implemented.

exit This command exits the program.

set keyword[=value] This command is used to change state information that affects the lookups. Valid keywords are:

> **all** This keyword prints the current values of the frequently used options to `set`. Information about the current default server and host is also printed.

> **class=value** This keyword changes the query class to one of:
>
>> **IN** the Internet class
>>
>> **CH** the Chaos class
>>
>> **HS** the Hesiod class
>>
>> **ANY** wildcard
>
> The class specifies the protocol group of the information. The default is `IN`; the abbreviation for this keyword is `cl`.

> **nodebug** This keyword turns on or off the display of the full response packet, and any intermediate response packets, when searching. The default for this keyword is `nodebug`; the abbreviation for this keyword is `[no]deb`.

> **nod2** This keyword turns debugging mode on or off. This displays more about what nslookup is doing. The default is `nod2`.

> **domain=name** This keyword sets the search list to `name`.

nosearch If the lookup request contains at least one period, but does not end with a trailing period, this keyword appends the domain names in the domain search list to the request until an answer is received. The default is `search`.

port=value This keyword changes the default TCP/UDP name server port to `value` from its default, port 53. The abbreviation for this keyword is `po`.

querytype=value|type=value This keyword changes the type of the information query to `value`. The defaults are A and then AAAA; the abbreviations for these keywords are `q` and `ty`.

Please note that it is only possible to specify one query type. Only the default behavior looks up both when an alternative is not specified.

norecurse This keyword tells the name server to query other servers if it does not have the information. The default is `recurse`; the abbreviation for this keyword is `[no]rec`.

ndots=number This keyword sets the number of dots (label separators) in a domain that disables searching. Absolute names always stop searching.

retry=number This keyword sets the number of retries to `number`.

timeout=number This keyword changes the initial timeout interval to wait for a reply to `number`, in seconds.

novc This keyword indicates that a virtual circuit should always be used when sending requests to the server. `novc` is the default.

nofail This keyword tries the next nameserver if a nameserver responds with SERVFAIL or a referral (nofail), or terminates the query (fail) on such a response. The default is `nofail`.

12.30.5 Return Values

`nslookup` returns with an exit status of 1 if any query failed, and 0 otherwise.

12.30.6 IDN Support

If `nslookup` has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. `nslookup` appropriately converts character encoding of a domain name before sending a request to a DNS server or displaying a reply from the server. To turn off IDN support, define the `IDN_DISABLE` environment variable. IDN support is disabled if the variable is set when `nslookup` runs, or when the standard output is not a tty.

12.30.7 Files

`/etc/resolv.conf`

12.30.8 See Also

dig(1), *host(1)*, *named(8)*.

12.31 nsupdate - dynamic DNS update utility

12.31.1 Synopsis

nsupdate [**-d**] [**-D**] [**-i**] [**-L** level] [[**-g**] | [**-o**] | [**-l**] | [**-y** [hmac:]keyname:secret] | [**-k** keyfile]] [**-t** timeout] [**-u** udptimeout] [**-r** udpretries] [**-v**] [**-T**] [**-P**] [**-V**] [[**-4**] | [**-6**]] [filename]

12.31.2 Description

nsupdate is used to submit Dynamic DNS Update requests, as defined in **RFC 2136**, to a name server. This allows resource records to be added or removed from a zone without manually editing the zone file. A single update request can contain requests to add or remove more than one resource record.

Zones that are under dynamic control via nsupdate or a DHCP server should not be edited by hand. Manual edits could conflict with dynamic updates and cause data to be lost.

The resource records that are dynamically added or removed with nsupdate must be in the same zone. Requests are sent to the zone's primary server, which is identified by the MNAME field of the zone's SOA record.

Transaction signatures can be used to authenticate the Dynamic DNS updates. These use the TSIG resource record type described in **RFC 2845**, the SIG(0) record described in **RFC 2535** and **RFC 2931**, or GSS-TSIG as described in **RFC 3645**.

TSIG relies on a shared secret that should only be known to nsupdate and the name server. For instance, suitable key and server statements are added to /etc/named.conf so that the name server can associate the appropriate secret key and algorithm with the IP address of the client application that is using TSIG authentication. ddns-confgen can generate suitable configuration fragments. nsupdate uses the -y or -k options to provide the TSIG shared secret; these options are mutually exclusive.

SIG(0) uses public key cryptography. To use a SIG(0) key, the public key must be stored in a KEY record in a zone served by the name server.

GSS-TSIG uses Kerberos credentials. Standard GSS-TSIG mode is switched on with the -g flag. A non-standards-compliant variant of GSS-TSIG used by Windows 2000 can be switched on with the -o flag.

12.31.3 Options

-4 This option sets use of IPv4 only.

-6 This option sets use of IPv6 only.

-d This option sets debug mode, which provides tracing information about the update requests that are made and the replies received from the name server.

-D This option sets extra debug mode.

-i This option forces interactive mode, even when standard input is not a terminal.

-k keyfile This option indicates the file containing the TSIG authentication key. Keyfiles may be in two formats: a single file containing a named.conf-format key statement, which may be generated automatically by ddns-confgen; or a pair of files whose names are of the format K{name}.+157.+{random}.key and K{name}.+157.+{random}.private, which can be generated by dnssec-keygen. The -k option can also be used to specify a SIG(0) key used to authenticate Dynamic DNS update requests. In this case, the key specified is not an HMAC-MD5 key.

-l This option sets local-host only mode, which sets the server address to localhost (disabling the `server` so that the server address cannot be overridden). Connections to the local server use a TSIG key found in `/var/run/named/session.key`, which is automatically generated by `named` if any local `primary` zone has set `update-policy` to `local`. The location of this key file can be overridden with the `-k` option.

-L level This option sets the logging debug level. If zero, logging is disabled.

-p port This option sets the port to use for connections to a name server. The default is 53.

-P This option prints the list of private BIND-specific resource record types whose format is understood by `nsupdate`. See also the `-T` option.

-r udpretries This option sets the number of UDP retries. The default is 3. If zero, only one update request is made.

-t timeout This option sets the maximum time an update request can take before it is aborted. The default is 300 seconds. If zero, the timeout is disabled.

-T This option prints the list of IANA standard resource record types whose format is understood by `nsupdate`. `nsupdate` exits after the lists are printed. The `-T` option can be combined with the `-P` option.

Other types can be entered using `TYPEXXXXX` where `XXXXX` is the decimal value of the type with no leading zeros. The rdata, if present, is parsed using the UNKNOWN rdata format, (<backslash> <hash> <space> <length> <space> <hexstring>).

-u udptimeout This option sets the UDP retry interval. The default is 3 seconds. If zero, the interval is computed from the timeout interval and number of UDP retries.

-v This option specifies that TCP should be used even for small update requests. By default, `nsupdate` uses UDP to send update requests to the name server unless they are too large to fit in a UDP request, in which case TCP is used. TCP may be preferable when a batch of update requests is made.

-V This option prints the version number and exits.

-y [hmac:]keyname:secret This option sets the literal TSIG authentication key. `keyname` is the name of the key, and `secret` is the base64 encoded shared secret. `hmac` is the name of the key algorithm; valid choices are `hmac-md5`, `hmac-sha1`, `hmac-sha224`, `hmac-sha256`, `hmac-sha384`, or `hmac-sha512`. If `hmac` is not specified, the default is `hmac-md5`, or if MD5 was disabled, `hmac-sha256`.

NOTE: Use of the `-y` option is discouraged because the shared secret is supplied as a command-line argument in clear text. This may be visible in the output from ps1 or in a history file maintained by the user's shell.

12.31.4 Input Format

`nsupdate` reads input from `filename` or standard input. Each command is supplied on exactly one line of input. Some commands are for administrative purposes; others are either update instructions or prerequisite checks on the contents of the zone. These checks set conditions that some name or set of resource records (RRset) either exists or is absent from the zone. These conditions must be met if the entire update request is to succeed. Updates are rejected if the tests for the prerequisite conditions fail.

Every update request consists of zero or more prerequisites and zero or more updates. This allows a suitably authenticated update request to proceed if some specified resource records are either present or missing from the zone. A blank input line (or the `send` command) causes the accumulated commands to be sent as one Dynamic DNS update request to the name server.

The command formats and their meanings are as follows:

server servername port This command sends all dynamic update requests to the name server `servername`. When no server statement is provided, `nsupdate` sends updates to the primary server of the correct zone. The MNAME field of that zone's SOA record identify the primary server for that zone. `port` is the port number on

servername where the dynamic update requests are sent. If no port number is specified, the default DNS port number of 53 is used.

local address port This command sends all dynamic update requests using the local `address`. When no local statement is provided, `nsupdate` sends updates using an address and port chosen by the system. `port` can also be used to force requests to come from a specific port. If no port number is specified, the system assigns one.

zone zonename This command specifies that all updates are to be made to the zone `zonename`. If no `zone` statement is provided, `nsupdate` attempts to determine the correct zone to update based on the rest of the input.

class classname This command specifies the default class. If no `class` is specified, the default class is `IN`.

ttl seconds This command specifies the default time-to-live, in seconds, for records to be added. The value `none` clears the default TTL.

key hmac:keyname secret This command specifies that all updates are to be TSIG-signed using the `keyname`-`secret` pair. If `hmac` is specified, it sets the signing algorithm in use. The default is `hmac-md5`; if MD5 was disabled, the default is `hmac-sha256`. The `key` command overrides any key specified on the command line via `-y` or `-k`.

gsstsig This command uses GSS-TSIG to sign the updates. This is equivalent to specifying `-g` on the command line.

oldgsstsig This command uses the Windows 2000 version of GSS-TSIG to sign the updates. This is equivalent to specifying `-o` on the command line.

realm [realm_name] When using GSS-TSIG, this command specifies the use of `realm_name` rather than the default realm in `krb5.conf`. If no realm is specified, the saved realm is cleared.

check-names [yes_or_no] This command turns on or off check-names processing on records to be added. Check-names has no effect on prerequisites or records to be deleted. By default check-names processing is on. If check-names processing fails, the record is not added to the UPDATE message.

prereq nxdomain domain-name This command requires that no resource record of any type exist with the name `domain-name`.

prereq yxdomain domain-name This command requires that `domain-name` exist (as at least one resource record, of any type).

prereq nxrrset domain-name class type This command requires that no resource record exist of the specified `type`, `class`, and `domain-name`. If `class` is omitted, IN (Internet) is assumed.

prereq yxrrset domain-name class type This command requires that a resource record of the specified `type`, `class` and `domain-name` exist. If `class` is omitted, IN (internet) is assumed.

prereq yxrrset domain-name class type data With this command, the `data` from each set of prerequisites of this form sharing a common `type`, `class`, and `domain-name` are combined to form a set of RRs. This set of RRs must exactly match the set of RRs existing in the zone at the given `type`, `class`, and `domain-name`. The `data` are written in the standard text representation of the resource record's RDATA.

update delete domain-name ttl class type data This command deletes any resource records named `domain-name`. If `type` and `data` are provided, only matching resource records are removed. The Internet class is assumed if `class` is not supplied. The `ttl` is ignored, and is only allowed for compatibility.

update add domain-name ttl class type data This command adds a new resource record with the specified `ttl`, `class`, and `data`.

show This command displays the current message, containing all of the prerequisites and updates specified since the last send.

send This command sends the current message. This is equivalent to entering a blank line.

answer This command displays the answer.

debug This command turns on debugging.

---

version This command prints the version number.

help This command prints a list of commands.

Lines beginning with a semicolon (;) are comments and are ignored.

12.31.5 Examples

The examples below show how `nsupdate` can be used to insert and delete resource records from the `example.com` zone. Notice that the input in each example contains a trailing blank line, so that a group of commands is sent as one dynamic update request to the primary name server for `example.com`.

```
# nsupdate
> update delete oldhost.example.com A
> update add newhost.example.com 86400 A 172.16.1.1
> send
```

Any A records for `oldhost.example.com` are deleted, and an A record for `newhost.example.com` with IP address 172.16.1.1 is added. The newly added record has a TTL of 1 day (86400 seconds).

```
# nsupdate
> prereq nxdomain nickname.example.com
> update add nickname.example.com 86400 CNAME somehost.example.com
> send
```

The prerequisite condition tells the name server to verify that there are no resource records of any type for `nickname.example.com`. If there are, the update request fails. If this name does not exist, a CNAME for it is added. This ensures that when the CNAME is added, it cannot conflict with the long-standing rule in **RFC 1034** that a name must not exist as any other record type if it exists as a CNAME. (The rule has been updated for DNSSEC in **RFC 2535** to allow CNAMEs to have RRSIG, DNSKEY, and NSEC records.)

12.31.6 Files

/etc/resolv.conf Used to identify the default name server

/var/run/named/session.key Sets the default TSIG key for use in local-only mode

K{name}.+157.+{random}.key Base-64 encoding of the HMAC-MD5 key created by `dnssec-keygen`.

K{name}.+157.+{random}.private Base-64 encoding of the HMAC-MD5 key created by `dnssec-keygen`.

12.31.7 See Also

RFC 2136, **RFC 3007**, **RFC 2104**, **RFC 2845**, **RFC 1034**, **RFC 2535**, **RFC 2931**, *named(8)*, *ddns-confgen(8)*, *dnssec-keygen(8)*.

12.31.8 Bugs

The TSIG key is redundantly stored in two separate files. This is a consequence of `nsupdate` using the DST library for its cryptographic operations, and may change in future releases. pkcs11-destroy - destroy PKCS#11 objects

12.31.9 Synopsis

pkcs11-destroy [**-m** module] [**-s** slot] [**-i** ID] [**-l** label] [**-p** PIN] [**-w** seconds]

12.31.10 Description

`pkcs11-destroy` destroys keys stored in a PKCS#11 device, identified by their `ID` or `label`.

Matching keys are displayed before being destroyed. By default, there is a five-second delay to allow the user to interrupt the process before the destruction takes place.

12.31.11 Options

−m module This option specifies the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

−s slot This option opens the session with the given PKCS#11 slot. The default is slot 0.

−i ID This option destroys keys with the given object ID.

−l label This option destroys keys with the given label.

−p PIN This option specifies the `PIN` for the device. If no `PIN` is provided on the command line, `pkcs11-destroy` prompts for it.

−w seconds This option specifies how long, in seconds, to pause before carrying out key destruction. The default is 5 seconds. If set to `0`, destruction is immediate.

12.31.12 See Also

pkcs11-keygen(8), *pkcs11-list(8)*, *pkcs11-tokens(8)*

12.32 pkcs11-keygen - generate keys on a PKCS#11 device

12.32.1 Synopsis

pkcs11-keygen [**-a** algorithm] [**-b** keysize] [**-e**] [**-i** id] [**-m** module] [**-P**] [**-p** PIN] [**-q**] [**-S**] [**-s** slot] label

12.32.2 Description

`pkcs11-keygen` causes a PKCS#11 device to generate a new key pair with the given `label` (which must be unique) and with `keysize` bits of prime.

12.32.3 Options

-a algorithm This option specifies the key algorithm class: supported classes are RSA, DSA, DH, ECC, and ECX. In addition to these strings, the `algorithm` can be specified as a DNSSEC signing algorithm to be used with this key; for example, NSEC3RSASHA1 maps to RSA, ECDSAP256SHA256 maps to ECC, and ED25519 to ECX. The default class is `RSA`.

-b keysize This option creates the key pair with `keysize` bits of prime. For ECC keys, the only valid values are 256 and 384, and the default is 256. For ECX keys, the only valid values are 256 and 456, and the default is 256.

-e For RSA keys only, this option specifies use of a large exponent.

-i id This option creates key objects with `id`. The ID is either an unsigned short 2-byte or an unsigned long 4-byte number.

-m module This option specifies the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

-P This option sets the new private key to be non-sensitive and extractable, and allows the private key data to be read from the PKCS#11 device. The default is for private keys to be sensitive and non-extractable.

-p PIN This option specifies the `PIN` for the device. If no `PIN` is provided on the command line, `pkcs11-keygen` prompts for it.

-q This option sets quiet mode, which suppresses unnecessary output.

-S For Diffie-Hellman (DH) keys only, this option specifies use of a special prime of 768-, 1024-, or 1536-bit size and base (AKA generator) 2. If not specified, bit size defaults to 1024.

-s slot This option opens the session with the given PKCS#11 slot. The default is slot 0.

12.32.4 See Also

pkcs11-destroy(8), *pkcs11-list(8)*, *pkcs11-tokens(8)*, *dnssec-keyfromlabel(8)*

12.33 pkcs11-list - list PKCS#11 objects

pkcs11-list [**-P**] [**-m** module] [**-s** slot] [**-i** ID **]** [**-l** label] [**-p** PIN]

12.33.1 Description

`pkcs11-list` lists the PKCS#11 objects with `ID` or `label` or, by default, all objects. The object class, label, and ID are displayed for all keys. For private or secret keys, the extractability attribute is also displayed, as either `true`, `false`, or `never`.

12.33.2 Options

−P This option lists only the public objects. (Note that on some PKCS#11 devices, all objects are private.)

−m module This option specifies the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

−s slot This option opens the session with the given PKCS#11 slot. The default is slot 0.

−i ID This option lists only key objects with the given object ID.

−l label This option lists only key objects with the given label.

−p PIN This option specifies the `PIN` for the device. If no `PIN` is provided on the command line, `pkcs11-list` prompts for it.

12.33.3 See Also

pkcs11-destroy(8), *pkcs11-keygen(8)*, *pkcs11-tokens(8)*

12.34 pkcs11-tokens - list PKCS#11 available tokens

12.34.1 Synopsis

pkcs11-tokens [**-m** module] [**-v**]

12.34.2 Description

`pkcs11-tokens` lists the PKCS#11 available tokens with defaults from the slot/token scan performed at application initialization.

12.34.3 Options

−m module This option specifies the PKCS#11 provider module. This must be the full path to a shared library object implementing the PKCS#11 API for the device.

−v This option makes the PKCS#11 libisc initialization verbose.

12.34.4 See Also

pkcs11-destroy(8), *pkcs11-keygen(8)*, *pkcs11-list(8)*

12.35 rndc-confgen - rndc key generation tool

12.35.1 Synopsis

rndc-confgen [**-a**] [**-A** algorithm] [**-b** keysize] [**-c** keyfile] [**-h**] [**-k** keyname] [**-p** port] [**-s** address] [**-t** chrootdir] [**-u** user]

12.35.2 Description

`rndc-confgen` generates configuration files for `rndc`. It can be used as a convenient alternative to writing the `rndc.conf` file and the corresponding `controls` and `key` statements in `named.conf` by hand. Alternatively, it can be run with the `-a` option to set up a `rndc.key` file and avoid the need for a `rndc.conf` file and a `controls` statement altogether.

12.35.3 Options

-a This option sets automatic `rndc` configuration, which creates a file `rndc.key` in `/etc` (or a different `sysconfdir` specified when BIND was built) that is read by both `rndc` and `named` on startup. The `rndc.key` file defines a default command channel and authentication key allowing `rndc` to communicate with `named` on the local host with no further configuration.

If a more elaborate configuration than that generated by `rndc-confgen -a` is required, for example if rndc is to be used remotely, run `rndc-confgen` without the `-a` option and set up `rndc.conf` and `named.conf` as directed.

-A algorithm This option specifies the algorithm to use for the TSIG key. Available choices are: hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, and hmac-sha512. The default is hmac-sha256.

-b keysize This option specifies the size of the authentication key in bits. The size must be between 1 and 512 bits; the default is the hash size.

-c keyfile This option is used with the `-a` option to specify an alternate location for `rndc.key`.

-h This option prints a short summary of the options and arguments to `rndc-confgen`.

-k keyname This option specifies the key name of the `rndc` authentication key. This must be a valid domain name. The default is `rndc-key`.

-p port This option specifies the command channel port where `named` listens for connections from `rndc`. The default is 953.

-s address This option specifies the IP address where `named` listens for command-channel connections from `rndc`. The default is the loopback address 127.0.0.1.

-t chrootdir This option is used with the `-a` option to specify a directory where `named` runs chrooted. An additional copy of the `rndc.key` is written relative to this directory, so that it is found by the chrooted `named`.

-u user This option is used with the `-a` option to set the owner of the generated `rndc.key` file. If `-t` is also specified, only the file in the chroot area has its owner changed.

12.35.4 Examples

To allow `rndc` to be used with no manual configuration, run:

`rndc-confgen -a`

To print a sample `rndc.conf` file and the corresponding `controls` and `key` statements to be manually inserted into `named.conf`, run:

`rndc-confgen`

12.35.5 See Also

rndc(8), *rndc.conf(5)*, *named(8)*, BIND 9 Administrator Reference Manual.

12.36 rndc.conf - rndc configuration file

12.36.1 Synopsis

`rndc.conf`

12.36.2 Description

`rndc.conf` is the configuration file for `rndc`, the BIND 9 name server control utility. This file has a similar structure and syntax to `named.conf`. Statements are enclosed in braces and terminated with a semi-colon. Clauses in the statements are also semi-colon terminated. The usual comment styles are supported:

C style: /* */

C++ style: // to end of line

Unix style: # to end of line

`rndc.conf` is much simpler than `named.conf`. The file uses three statements: an options statement, a server statement, and a key statement.

The `options` statement contains five clauses. The `default-server` clause is followed by the name or address of a name server. This host is used when no name server is given as an argument to `rndc`. The `default-key` clause is followed by the name of a key, which is identified by a `key` statement. If no `keyid` is provided on the rndc command line, and no `key` clause is found in a matching `server` statement, this default key is used to authenticate the server's commands and responses. The `default-port` clause is followed by the port to connect to on the remote name server. If no `port` option is provided on the rndc command line, and no `port` clause is found in a matching `server` statement, this default port is used to connect. The `default-source-address` and `default-source-address-v6` clauses can be used to set the IPv4 and IPv6 source addresses respectively.

After the `server` keyword, the server statement includes a string which is the hostname or address for a name server. The statement has three possible clauses: `key`, `port`, and `addresses`. The key name must match the name of a key statement in the file. The port number specifies the port to connect to. If an `addresses` clause is supplied, these addresses are used instead of the server name. Each address can take an optional port. If a `source-address` or `source-address-v6` is supplied, it is used to specify the IPv4 and IPv6 source address, respectively.

The `key` statement begins with an identifying string, the name of the key. The statement has two clauses. `algorithm` identifies the authentication algorithm for `rndc` to use; currently only HMAC-MD5 (for compatibility), HMAC-SHA1, HMAC-SHA224, HMAC-SHA256 (default), HMAC-SHA384, and HMAC-SHA512 are supported. This is followed by

a secret clause which contains the base-64 encoding of the algorithm's authentication key. The base-64 string is enclosed in double quotes.

There are two common ways to generate the base-64 string for the secret. The BIND 9 program `rndc-confgen` can be used to generate a random key, or the `mmencode` program, also known as `mimencode`, can be used to generate a base-64 string from known input. `mmencode` does not ship with BIND 9 but is available on many systems. See the Example section for sample command lines for each.

12.36.3 Example

```
options {
  default-server  localhost;
  default-key     samplekey;
};
```

```
server localhost {
  key             samplekey;
};
```

```
server testserver {
  key     testkey;
  addresses   { localhost port 5353; };
};
```

```
key samplekey {
  algorithm       hmac-sha256;
  secret          "6FMfj43Osz4lyb24OIe2iGEz9lf1llJO+lz";
};
```

```
key testkey {
  algorithm   hmac-sha256;
  secret      "R3HI8P6BKw9ZwXwN3VZKuQ==";
};
```

In the above example, `rndc` by default uses the server at localhost (127.0.0.1) and the key called "samplekey". Commands to the localhost server use the "samplekey" key, which must also be defined in the server's configuration file with the same name and secret. The key statement indicates that "samplekey" uses the HMAC-SHA256 algorithm and its secret clause contains the base-64 encoding of the HMAC-SHA256 secret enclosed in double quotes.

If `rndc -s testserver` is used, then `rndc` connects to the server on localhost port 5353 using the key "testkey".

To generate a random secret with `rndc-confgen`:

```
rndc-confgen
```

A complete `rndc.conf` file, including the randomly generated key, is written to the standard output. Commented-out `key` and `controls` statements for `named.conf` are also printed.

To generate a base-64 secret with `mmencode`:

```
echo "known plaintext for a secret" | mmencode
```

12.36.4 Name Server Configuration

The name server must be configured to accept rndc connections and to recognize the key specified in the `rndc.conf` file, using the controls statement in `named.conf`. See the sections on the `controls` statement in the BIND 9 Administrator Reference Manual for details.

12.36.5 See Also

rndc(8), *rndc-confgen(8)*, *mmencode(1)*, BIND 9 Administrator Reference Manual.

12.37 rndc - name server control utility

12.37.1 Synopsis

rndc [**-b** source-address] [**-c** config-file] [**-k** key-file] [**-s** server] [**-p** port] [**-q**] [**-r**] [**-V**] [**-y** key_id] [[**-4**] | [**-6**]] {command}

12.37.2 Description

`rndc` controls the operation of a name server; it supersedes the `ndc` utility. If `rndc` is invoked with no command line options or arguments, it prints a short summary of the supported commands and the available options and their arguments.

`rndc` communicates with the name server over a TCP connection, sending commands authenticated with digital signatures. In the current versions of `rndc` and `named`, the only supported authentication algorithms are HMAC-MD5 (for compatibility), HMAC-SHA1, HMAC-SHA224, HMAC-SHA256 (default), HMAC-SHA384, and HMAC-SHA512. They use a shared secret on each end of the connection, which provides TSIG-style authentication for the command request and the name server's response. All commands sent over the channel must be signed by a key_id known to the server.

`rndc` reads a configuration file to determine how to contact the name server and decide what algorithm and key it should use.

12.37.3 Options

-4 This option indicates use of IPv4 only.

-6 This option indicates use of IPv6 only.

-b source-address This option indicates `source-address` as the source address for the connection to the server. Multiple instances are permitted, to allow setting of both the IPv4 and IPv6 source addresses.

-c config-file This option indicates `config-file` as the configuration file instead of the default, `/etc/rndc.conf`.

-k key-file This option indicates `key-file` as the key file instead of the default, `/etc/rndc.key`. The key in `/etc/rndc.key` is used to authenticate commands sent to the server if the config-file does not exist.

-s server `server` is the name or address of the server which matches a server statement in the configuration file for `rndc`. If no server is supplied on the command line, the host named by the default-server clause in the options statement of the `rndc` configuration file is used.

-p port This option instructs BIND 9 to send commands to TCP port `port` instead of its default control channel port, 953.

-q This option sets quiet mode, where message text returned by the server is not printed unless there is an error.

-r This option instructs `rndc` to print the result code returned by `named` after executing the requested command (e.g., ISC_R_SUCCESS, ISC_R_FAILURE, etc.).

-V This option enables verbose logging.

-y key_id This option indicates use of the key `key_id` from the configuration file. For control message validation to succeed, `key_id` must be known by `named` with the same algorithm and secret string. If no `key_id` is specified, `rndc` first looks for a key clause in the server statement of the server being used, or if no server statement is present for that host, then in the default-key clause of the options statement. Note that the configuration file contains shared secrets which are used to send authenticated control commands to name servers, and should therefore not have general read or write access.

12.37.4 Commands

A list of commands supported by `rndc` can be seen by running `rndc` without arguments.

Currently supported commands are:

addzone *zone* [*class* [*view*]] *configuration* This command adds a zone while the server is running. This command requires the `allow-new-zones` option to be set to `yes`. The configuration string specified on the command line is the zone configuration text that would ordinarily be placed in `named.conf`.

The configuration is saved in a file called `viewname.nzf` (or, if `named` is compiled with liblmdb, an LMDB database file called `viewname.nzd`). `viewname` is the name of the view, unless the view name contains characters that are incompatible with use as a file name, in which case a cryptographic hash of the view name is used instead. When `named` is restarted, the file is loaded into the view configuration so that zones that were added can persist after a restart.

This sample `addzone` command adds the zone `example.com` to the default view:

```
rndc addzone example.com '{ type master; file "example.com.db"; };'
```

(Note the brackets around and semi-colon after the zone configuration text.)

See also `rndc delzone` and `rndc modzone`.

delzone [**-clean**] *zone* [*class* [*view*]] This command deletes a zone while the server is running.

If the `-clean` argument is specified, the zone's master file (and journal file, if any) are deleted along with the zone. Without the `-clean` option, zone files must be deleted manually. (If the zone is of type `secondary` or `stub`, the files needing to be removed are reported in the output of the `rndc delzone` command.)

If the zone was originally added via `rndc addzone`, then it is removed permanently. However, if it was originally configured in `named.conf`, then that original configuration remains in place; when the server is restarted or reconfigured, the zone is recreated. To remove it permanently, it must also be removed from `named.conf`.

See also `rndc addzone` and `rndc modzone`.

dnssec (**-status** | **-rollover -key** *id* [**-alg** *algorithm*] [**-when** *time*] | **-checkds** [**-key** *id* [**-alg** *algorithm*]] [**-when** *time*] (*published* | *with* This command allows you to interact with the "dnssec-policy" of a given zone.

`rndc dnssec -status` show the DNSSEC signing state for the specified zone.

`rndc dnssec -rollover` allows you to schedule key rollover for a specific key (overriding the original key lifetime).

`rndc dnssec -checkds` will let `named` know that the DS for the given key has been seen published into or withdrawn from the parent. This is required in order to complete a KSK rollover. If the `-key id` argument is specified, look for the key with the given identifier, otherwise if there is only one key acting as a KSK in the zone, assume the DS of that key (if there are multiple keys with the same tag, use `-alg algorithm` to select the

correct algorithm). The time that the DS has been published or withdrawn is set to now, unless otherwise specified with the argument `-when time`.

dnstap (-reopen | -roll [*number*]) This command closes and re-opens DNSTAP output files. `rndc dnstap -reopen` allows the output file to be renamed externally, so that `named` can truncate and re-open it. `rndc dnstap -roll` causes the output file to be rolled automatically, similar to log files. The most recent output file has ".0" appended to its name; the previous most recent output file is moved to ".1", and so on. If `number` is specified, then the number of backup log files is limited to that number.

dumpdb [-all | -cache | -zones | -adb | -bad | -expired | -fail] [*view ...*] This command dumps the server's caches (default) and/or zones to the dump file for the specified views. If no view is specified, all views are dumped. (See the `dump-file` option in the BIND 9 Administrator Reference Manual.)

flush This command flushes the server's cache.

flushname *name* [*view*] This command flushes the given name from the view's DNS cache and, if applicable, from the view's nameserver address database, bad server cache, and SERVFAIL cache.

flushtree *name* [*view*] This command flushes the given name, and all of its subdomains, from the view's DNS cache, address database, bad server cache, and SERVFAIL cache.

freeze [*zone* [*class* [*view*]]] This command suspends updates to a dynamic zone. If no zone is specified, then all zones are suspended. This allows manual edits to be made to a zone normally updated by dynamic update, and causes changes in the journal file to be synced into the master file. All dynamic update attempts are refused while the zone is frozen.

See also `rndc thaw`.

halt [-p] This command stops the server immediately. Recent changes made through dynamic update or IXFR are not saved to the master files, but are rolled forward from the journal files when the server is restarted. If `-p` is specified, `named`'s process ID is returned. This allows an external process to determine when `named` has completed halting.

See also `rndc stop`.

loadkeys [*zone* [*class* [*view*]]] This command fetches all DNSSEC keys for the given zone from the key directory. If they are within their publication period, they are merged into the zone's DNSKEY RRset. Unlike `rndc sign`, however, the zone is not immediately re-signed by the new keys, but is allowed to incrementally re-sign over time.

This command requires that the zone be configured with a `dnssec-policy`, or that the `auto-dnssec` zone option be set to `maintain`, and also requires the zone to be configured to allow dynamic DNS. (See "Dynamic Update Policies" in the Administrator Reference Manual for more details.)

managed-keys (*status* | *refresh* | *sync* | *destroy*) [*class* [*view*]] This command inspects and controls the "managed-keys" database which handles **RFC 5011** DNSSEC trust anchor maintenance. If a view is specified, these commands are applied to that view; otherwise, they are applied to all views.

- When run with the `status` keyword, this prints the current status of the managed-keys database.

- When run with the `refresh` keyword, this forces an immediate refresh query to be sent for all the managed keys, updating the managed-keys database if any new keys are found, without waiting the normal refresh interval.

- When run with the `sync` keyword, this forces an immediate dump of the managed-keys database to disk (in the file `managed-keys.bind` or (`viewname.mkeys`). This synchronizes the database with its journal file, so that the database's current contents can be inspected visually.

- When run with the `destroy` keyword, the managed-keys database is shut down and deleted, and all key maintenance is terminated. This command should be used only with extreme caution.

  Existing keys that are already trusted are not deleted from memory; DNSSEC validation can continue after this command is used. However, key maintenance operations cease until `named` is restarted or reconfigured, and all existing key maintenance states are deleted.

Running `rndc reconfig` or restarting `named` immediately after this command causes key maintenance to be reinitialized from scratch, just as if the server were being started for the first time. This is primarily intended for testing, but it may also be used, for example, to jumpstart the acquisition of new keys in the event of a trust anchor rollover, or as a brute-force repair for key maintenance problems.

modzone *zone* [*class* [*view*]] *configuration* This command modifies the configuration of a zone while the server is running. This command requires the `allow-new-zones` option to be set to `yes`. As with `addzone`, the configuration string specified on the command line is the zone configuration text that would ordinarily be placed in `named.conf`.

If the zone was originally added via `rndc addzone`, the configuration changes are recorded permanently and are still in effect after the server is restarted or reconfigured. However, if it was originally configured in `named.conf`, then that original configuration remains in place; when the server is restarted or reconfigured, the zone reverts to its original configuration. To make the changes permanent, it must also be modified in `named.conf`.

See also `rndc addzone` and `rndc delzone`.

notify *zone* [*class* [*view*]] This command resends NOTIFY messages for the zone.

notrace This command sets the server's debugging level to 0.

See also `rndc trace`.

nta [(**-class** *class* | **-dump** | **-force** | **-remove** | **-lifetime** *duration*)] *domain* [*view*] This command sets a DNSSEC negative trust anchor (NTA) for `domain`, with a lifetime of `duration`. The default lifetime is configured in `named.conf` via the `nta-lifetime` option, and defaults to one hour. The lifetime cannot exceed one week.

A negative trust anchor selectively disables DNSSEC validation for zones that are known to be failing because of misconfiguration rather than an attack. When data to be validated is at or below an active NTA (and above any other configured trust anchors), `named` aborts the DNSSEC validation process and treats the data as insecure rather than bogus. This continues until the NTA's lifetime has elapsed.

NTAs persist across restarts of the `named` server. The NTAs for a view are saved in a file called `name.nta`, where `name` is the name of the view; if it contains characters that are incompatible with use as a file name, a cryptographic hash is generated from the name of the view.

An existing NTA can be removed by using the `-remove` option.

An NTA's lifetime can be specified with the `-lifetime` option. TTL-style suffixes can be used to specify the lifetime in seconds, minutes, or hours. If the specified NTA already exists, its lifetime is updated to the new value. Setting `lifetime` to zero is equivalent to `-remove`.

If `-dump` is used, any other arguments are ignored and a list of existing NTAs is printed. Note that this may include NTAs that are expired but have not yet been cleaned up.

Normally, `named` periodically tests to see whether data below an NTA can now be validated (see the `nta-recheck` option in the Administrator Reference Manual for details). If data can be validated, then the NTA is regarded as no longer necessary and is allowed to expire early. The `-force` parameter overrides this behavior and forces an NTA to persist for its entire lifetime, regardless of whether data could be validated if the NTA were not present.

The view class can be specified with `-class`. The default is class `IN`, which is the only class for which DNSSEC is currently supported.

All of these options can be shortened, i.e., to `-l`, `-r`, `-d`, `-f`, and `-c`.

Unrecognized options are treated as errors. To refer to a domain or view name that begins with a hyphen, use a double-hyphen (–) on the command line to indicate the end of options.

querylog [(*on* | *off*)] This command enables or disables query logging. For backward compatibility, this command can also be used without an argument to toggle query logging on and off.

Query logging can also be enabled by explicitly directing the `queries category` to a `channel` in the `logging` section of `named.conf`, or by specifying `querylog yes;` in the `options` section of `named.conf`.

reconfig This command reloads the configuration file and loads new zones, but does not reload existing zone files even if they have changed. This is faster than a full `reload` when there is a large number of zones, because it avoids the need to examine the modification times of the zone files.

recursing This command dumps the list of queries `named` is currently recursing on, and the list of domains to which iterative queries are currently being sent.

The first list includes all unique clients that are waiting for recursion to complete, including the query that is awaiting a response and the timestamp (seconds since the Unix epoch) of when named started processing this client query.

The second list comprises of domains for which there are active (or recently active) fetches in progress. It reports the number of active fetches for each domain and the number of queries that have been passed (allowed) or dropped (spilled) as a result of the `fetches-per-zone` limit. (Note: these counters are not cumulative over time; whenever the number of active fetches for a domain drops to zero, the counter for that domain is deleted, and the next time a fetch is sent to that domain, it is recreated with the counters set to zero).

refresh *zone* [*class* [*view*]] This command schedules zone maintenance for the given zone.

reload This command reloads the configuration file and zones.

reload *zone* [*class* [*view*]] This command reloads the given zone.

retransfer *zone* [*class* [*view*]] This command retransfers the given secondary zone from the primary server.

If the zone is configured to use `inline-signing`, the signed version of the zone is discarded; after the retransfer of the unsigned version is complete, the signed version is regenerated with new signatures.

scan This command scans the list of available network interfaces for changes, without performing a full `reconfig` or waiting for the `interface-interval` timer.

secroots [-] [*view ...*] This command dumps the security roots (i.e., trust anchors configured via `trust-anchors`, or the `managed-keys` or `trusted-keys` statements [both deprecated], or `dnssec-validation auto`) and negative trust anchors for the specified views. If no view is specified, all views are dumped. Security roots indicate whether they are configured as trusted keys, managed keys, or initializing managed keys (managed keys that have not yet been updated by a successful key refresh query).

If the first argument is −, then the output is returned via the `rndc` response channel and printed to the standard output. Otherwise, it is written to the secroots dump file, which defaults to `named.secroots`, but can be overridden via the `secroots-file` option in `named.conf`.

See also `rndc managed-keys`.

serve-stale (on | off | reset | status) [*class* [*view*]] This command enables, disables, resets, or reports the current status of the serving of stale answers as configured in `named.conf`.

If serving of stale answers is disabled by `rndc-serve-stale off`, then it remains disabled even if `named` is reloaded or reconfigured. `rndc serve-stale reset` restores the setting as configured in `named.conf`.

`rndc serve-stale status` reports whether serving of stale answers is currently enabled, disabled by the configuration, or disabled by `rndc`. It also reports the values of `stale-answer-ttl` and `max-stale-ttl`.

showzone *zone* [*class* [*view*]] This command prints the configuration of a running zone.

See also `rndc zonestatus`.

sign *zone* [*class* [*view*]] This command fetches all DNSSEC keys for the given zone from the key directory (see the `key-directory` option in the BIND 9 Administrator Reference Manual). If they are within their publication period, they are merged into the zone's DNSKEY RRset. If the DNSKEY RRset is changed, then the zone is automatically re-signed with the new key set.

This command requires that the zone be configured with a `dnssec-policy`, or that the `auto-dnssec` zone option be set to `allow` or `maintain`, and also requires the zone to be configured to allow dynamic DNS. (See "Dynamic Update Policies" in the BIND 9 Administrator Reference Manual for more details.)

See also `rndc loadkeys`.

signing [(-list | -clear *keyid/algorithm* **| -clear** *all* **| -nsec3param (** *parameters* **| none) | -serial** *value* **)** *zone* **[***class* **[***view***]]**
This command lists, edits, or removes the DNSSEC signing-state records for the specified zone. The status of ongoing DNSSEC operations, such as signing or generating NSEC3 chains, is stored in the zone in the form of DNS resource records of type `sig-signing-type`. `rndc signing -list` converts these records into a human-readable form, indicating which keys are currently signing or have finished signing the zone, and which NSEC3 chains are being created or removed.

`rndc signing -clear` can remove a single key (specified in the same format that `rndc signing -list` uses to display it), or all keys. In either case, only completed keys are removed; any record indicating that a key has not yet finished signing the zone is retained.

`rndc signing -nsec3param` sets the NSEC3 parameters for a zone. This is the only supported mechanism for using NSEC3 with `inline-signing` zones. Parameters are specified in the same format as an NSEC3PARAM resource record: `hash algorithm`, `flags`, `iterations`, and `salt`, in that order.

Currently, the only defined value for `hash algorithm` is `1`, representing SHA-1. The `flags` may be set to `0` or `1`, depending on whether the opt-out bit in the NSEC3 chain should be set. `iterations` defines the number of additional times to apply the algorithm when generating an NSEC3 hash. The `salt` is a string of data expressed in hexadecimal, a hyphen (*-') if no salt is to be used, or the keyword `auto`, which causes `named` to generate a random 64-bit salt.

So, for example, to create an NSEC3 chain using the SHA-1 hash algorithm, no opt-out flag, 10 iterations, and a salt value of "FFFF", use: `rndc signing -nsec3param 1 0 10 FFFF zone`. To set the opt-out flag, 15 iterations, and no salt, use: `rndc signing -nsec3param 1 1 15 - zone`.

`rndc signing -nsec3param none` removes an existing NSEC3 chain and replaces it with NSEC.

`rndc signing -serial value` sets the serial number of the zone to `value`. If the value would cause the serial number to go backwards, it is rejected. The primary use of this parameter is to set the serial number on inline signed zones.

stats This command writes server statistics to the statistics file. (See the `statistics-file` option in the BIND 9 Administrator Reference Manual.)

status This command displays the status of the server. Note that the number of zones includes the internal `bind/CH` zone and the default `./IN` hint zone, if there is no explicit root zone configured.

stop -p This command stops the server, making sure any recent changes made through dynamic update or IXFR are first saved to the master files of the updated zones. If `-p` is specified, `named(8)`'s process ID is returned. This allows an external process to determine when ``named` has completed stopping.

See also `rndc halt`.

sync -clean [*zone* **[***class* **[***view***]]]** This command syncs changes in the journal file for a dynamic zone to the master file. If the "-clean" option is specified, the journal file is also removed. If no zone is specified, then all zones are synced.

tcp-timeouts [*initial idle keepalive advertised***]** When called without arguments, this command displays the current values of the `tcp-initial-timeout`, `tcp-idle-timeout`, `tcp-keepalive-timeout`, and `tcp-advertised-timeout` options. When called with arguments, these values are updated. This allows an administrator to make rapid adjustments when under a denial-of-service (DoS) attack. See the descriptions of these options in the BIND 9 Administrator Reference Manual for details of their use.

thaw [*zone* **[***class* **[***view***]]]** This command enables updates to a frozen dynamic zone. If no zone is specified, then all frozen zones are enabled. This causes the server to reload the zone from disk, and re-enables dynamic updates after

the load has completed. After a zone is thawed, dynamic updates are no longer refused. If the zone has changed and the `ixfr-from-differences` option is in use, the journal file is updated to reflect changes in the zone. Otherwise, if the zone has changed, any existing journal file is removed.

See also `rndc freeze`.

trace This command increments the server's debugging level by one.

trace *level* This command sets the server's debugging level to an explicit value.

See also `rndc notrace`.

tsig-delete *keyname* [*view*] This command deletes a given TKEY-negotiated key from the server. This does not apply to statically configured TSIG keys.

tsig-list This command lists the names of all TSIG keys currently configured for use by `named` in each view. The list includes both statically configured keys and dynamic TKEY-negotiated keys.

validation (on | off | status) [*view ...*]`` This command enables, disables, or checks the current status of DNSSEC validation. By default, validation is enabled.

The cache is flushed when validation is turned on or off to avoid using data that might differ between states.

zonestatus *zone* [*class* [*view*]] This command displays the current status of the given zone, including the master file name and any include files from which it was loaded, when it was most recently loaded, the current serial number, the number of nodes, whether the zone supports dynamic updates, whether the zone is DNSSEC signed, whether it uses automatic DNSSEC key management or inline signing, and the scheduled refresh or expiry times for the zone.

See also `rndc showzone`.

`rndc` commands that specify zone names, such as `reload`, `retransfer`, or `zonestatus`, can be ambiguous when applied to zones of type `redirect`. Redirect zones are always called `.`, and can be confused with zones of type `hint` or with secondary copies of the root zone. To specify a redirect zone, use the special zone name `-redirect`, without a trailing period. (With a trailing period, this would specify a zone called "-redirect".)

12.37.5 Limitations

There is currently no way to provide the shared secret for a `key_id` without using the configuration file.

Several error messages could be clearer.

12.37.6 See Also

rndc.conf(5), *rndc-confgen(8)*, *named(8)*, *named.conf(5)*, *ndc(8)*, BIND 9 Administrator Reference Manual.

12.38 tsig-keygen - TSIG key generation tool

12.38.1 Synopsis

tsig-keygen [**-a** algorithm] [**-h**] [name]

12.38.2 Description

`tsig-keygen` is an utility that generates keys for use in TSIG signing. The resulting keys can be used, for example, to secure dynamic DNS updates to a zone, or for the `rndc` command channel.

A domain name can be specified on the command line to be used as the name of the generated key. If no name is specified, the default is `tsig-key`.

12.38.3 Options

−a algorithm This option specifies the algorithm to use for the TSIG key. Available choices are: hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384, and hmac-sha512. The default is hmac-sha256. Options are case-insensitive, and the "hmac-" prefix may be omitted.

−h This option prints a short summary of options and arguments.

12.38.4 See Also

nsupdate(1), *named.conf(5)*, *named(8)*, BIND 9 Administrator Reference Manual.

S

Y