14 General DNS Reference Information 433
14.1 Requests for Comment (RFCs) ........................................... 433
14.2 Notes .............................................................................. 438
14.3 Internet Drafts .................................................................. 438

15 Manual Pages 439
15.1 arpaname - translate IP addresses to the corresponding ARPA names ................................................. 439
15.2 ddns-confgen - TSIG key generation tool ...................................................................................... 439
15.3 delv - DNS lookup and validation utility ...................................................................................... 440
15.4 dig - DNS lookup utility ............................................................................................................ 445
15.5 dnssec-cds - change DS records for a child zone based on CDS/CDNSKEY .......................................... 455
15.6 dnssec-dsfromkey - DNSSEC DS RR generation tool .................................................................... 458
15.7 dnssec-importkey - import DNSKEY records from external systems so they can be managed ........... 460
15.8 dnssec-keyfromlabel - DNSSEC key generation tool ....................................................................... 462
15.9 dnssec-keygen: DNSSEC key generation tool .................................................................................. 465
15.10 dnssec-ksr - Create signed key response (SKR) files for offline KSK setups .................................. 469
15.11 dnssec-revoke - set the REVOKE bit on a DNSSEC key ................................................................. 471
15.12 dnssec-settime: set the key timing metadata for a DNSSEC key .................................................... 472
15.13 dnssec-signzone - DNSSEC zone signing tool .............................................................................. 476
15.14 dnssec-verify - DNSSEC zone verification tool ............................................................................. 481
15.15 dnstap-read - print dnstap data in human-readable form .............................................................. 483
15.16 filter-aaaa.so - filter AAAA in DNS responses when A is present .................................................. 483
15.17 host - DNS lookup utility ............................................................................................................. 485
15.18 mdig - DNS pipelined lookup utility .............................................................................................. 487
15.19 named-checkconf - named configuration file syntax checking tool ................................................... 492
15.20 named-checkzone - zone file validation tool .................................................................................. 493
15.21 named-compilezone - zone file converting tool ............................................................................ 496
15.22 named-journalprint - print zone journal in human-readable form .................................................. 499
15.23 named-nzd2nzf - convert an NZD database to NZF text format ...................................................... 500
15.24 named-rrchecker - syntax checker for individual DNS resource records ........................................ 500
15.25 named.conf - configuration file for named .................................................................................... 501
15.26 named - Internet domain name server ............................................................................................ 519
15.27 nsec3hash - generate NSEC3 hash ............................................................................................... 522
15.28 nslookup - query Internet name servers interactively .................................................................... 523
15.29 nsupdate - dynamic DNS update utility .......................................................................................... 526
15.30 rndc-confgen - rndc key generation tool ....................................................................................... 532
15.31 rndc.conf - rndc configuration file ............................................................................................... 533
15.32 rndc - name server control utility ................................................................................................. 535
15.33 tsig-keygen - TSIG key generation tool .......................................................................................... 543

Index 545
INTRODUCTION TO DNS AND BIND 9

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

This manual covers BIND version 9.20.0.

1.2 Organization of This Document

Introduction to DNS and BIND 9 introduces the basic DNS and BIND concepts. Some tutorial material on The Domain Name System (DNS) is presented for those unfamiliar with DNS. A DNS Security Overview is provided to allow BIND operators to implement appropriate security for their operational environment.

Resource Requirements describes the hardware and environment requirements for BIND 9 and lists both the supported and unsupported platforms.

Configurations and Zone Files is intended as a quickstart guide for newer users. Sample files are included for Authoritative Name Servers (both primary and secondary), as well as a simple Resolver (Caching Name Servers) and a Forwarding Resolver Configuration. Some reference material on the Zone File is included.

Name Server Operations covers basic BIND 9 software and DNS operations, including some useful tools, Unix signals, and plugins.

Advanced Configurations builds on the configurations of Configurations and Zone Files, adding functions and features the system administrator may need.

Security Configurations covers most aspects of BIND 9 security, including file permissions, running BIND 9 in a “jail,” and securing file transfers and dynamic updates.

DNSSEC describes the theory and practice of cryptographic authentication of DNS information. The DNSSEC Guide is a practical guide to implementing DNSSEC.
Configuration Reference gives exhaustive descriptions of all supported blocks, statements, and grammars used in BIND 9's named.conf configuration file.

Troubleshooting provides information on identifying and solving BIND 9 and DNS problems. Information about bug-reporting procedures is also provided.

Building BIND 9 is a definitive guide for those occasions where the user requires special options not provided in the standard Linux or Unix distributions.

The Appendices contain useful reference information, such as a bibliography and historic information related to BIND and the Domain Name System, as well as the current man pages for all the published tools.

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 text, or italics is also used for emphasis or clarity.

1.4 The Domain Name System (DNS)

This is a brief description of the functionality and organization of the Domain Name System (DNS). It is provided to familiarize users with the concepts involved, the (often confusing) terminology used, and how all the parts fit together to form an operational system.

All network systems operate with network addresses, such as IPv4 and IPv6. The vast majority of humans find it easier to work with names rather than seemingly endless strings of network address digits. The earliest ARPANET systems (from which the Internet evolved) mapped names to addresses using a hosts file that was distributed to all entities whenever changes occurred. Operationally, such a system became rapidly unsustainable once there were more than 100 networked entities, which led to the specification and implementation of the Domain Name System that we use today.
1.4.1 DNS Fundamentals

The DNS naming system is organized as a tree structure comprised of multiple levels and thus it naturally creates a distributed system. Each node in the tree is given a label which defines its Domain (its area or zone) of Authority. The topmost node in the tree is the Root Domain; it delegates to Domains at the next level which are generically known as the Top-Level Domains (TLDs). They in turn delegate to Second-Level Domains (SLDs), and so on. The Top-Level Domains (TLDs) include a special group of TLDs called the Country Code Top-Level Domains (ccTLDs), in which every country is assigned a unique two-character country code from ISO 3166 as its domain.

Note: The Domain Name System is controlled by ICANN (https://www.icann.org) (a 501c non-profit entity); their current policy is that any new TLD, consisting of three or more characters, may be proposed by any group of commercial sponsors and if it meets ICANN’s criteria will be added to the TLDs.

The concept of delegation and authority flows down the DNS tree (the DNS hierarchy) as shown:

![Delegation and Authority in the DNS Name Space](image)

A domain is the label of a node in the tree. A domain name uniquely identifies any node in the DNS tree and is written, left to right, by combining all the domain labels (each of which are unique within their parent’s zone or domain of authority), with a dot separating each component, up to the root domain. In the above diagram the following are all domain names:

```
example.com
b.com
ac.uk
us
org
```

The root has a unique label of “.” (dot), which is normally omitted when it is written as a domain name, but when it is written as a Fully Qualified Domain Name (FQDN) the dot must be present. Thus:

```
example.com    # domain name
test.example.com. # FQDN
```
1.4.2 Authority and Delegation

Each domain (node) has been delegated the authority from its parent domain. The delegated authority includes specific responsibilities to ensure that every domain it delegates has a unique name or label within its zone or domain of authority, and that it maintains an authoritative list of its delegated domains. The responsibilities further include an operational requirement to operate two (or more) name servers (which may be contracted to a third party) which will contain the authoritative data for all the domain labels within its zone of authority in a zone file. Again, the tree structure ensures that the DNS name space is naturally distributed.

The following diagram illustrates that Authoritative Name Servers exist for every level and every domain in the DNS name space:

```
Fig. 2: Authoritative Name Servers in the DNS Name Space
```

Note: The difference between a domain and a zone can appear confusing. Practically, the terms are generally used synonymously in the DNS. If, however, you are into directed graphs and tree structure theory or similar exotica, a zone can be considered as an arc through any node (or domain) with the domain at its apex. The zone therefore encompasses all the name space below the domain. This can, however, lead to the concept of subzones and these were indeed defined in the original DNS specifications. Thankfully the term subzone has been lost in the mists of time.

1.4.3 Root Servers

The root servers are a critical part of the DNS authoritative infrastructure. There are 13 root servers (a.root-servers.net to m.root-servers.net). The number 13 is historically based on the maximum amount of name and IPv4 data that could be packed into a 512-byte UDP message, and not a perverse affinity for a number that certain cultures treat as unlucky. The 512-byte UDP data limit is no longer a limiting factor and all root servers now support both IPv4 and IPv6. In addition, almost all the root servers use anycast, with well over 300 instances of the root servers now providing service worldwide (see further information at https://www.root-servers.org). The root servers are the starting point for all name resolution within the DNS.
1.4.4 Name Resolution

So far all the emphasis has been on how the DNS stores its authoritative domain (zone) data. End-user systems use names (an email address or a web address) and need to access this authoritative data to obtain an IP address, which they use to contact the required network resources such as web, FTP, or mail servers. The process of converting a domain name to a result (typically an IP address, though other types of data may be obtained) is generically called name resolution, and is handled by resolvers (also known as caching name servers and many other terms). The following diagram shows the typical name resolution process:

![Diagram of Name Resolution Process](image.png)

Fig. 3: Authoritative Name Servers and Name Resolution

An end-user application, such as a browser (1), when needing to resolve a name such as www.example.com, makes an internal system call to a minimal function resolution entity called a stub resolver (2). The stub resolver (using stored IP addresses) contacts a resolver (a caching name server or full-service resolver) (3), which in turn contacts all the necessary authoritative name servers (4, 5, and 6) to provide the answer that it then returns to the user (2, 1). To improve performance, all resolvers (including most stub resolvers) cache (store) their results such that a subsequent request for the same data is taken from the resolver’s cache, removing the need to repeat the name resolution process and use time-consuming resources. All communication between the stub resolver, the resolver, and the authoritative name servers uses the DNS protocol's query and response message pair.

1.4.5 DNS Protocol and Queries

DNS queries use the UDP protocol over the reserved port 53 (but both TCP and TLS can optionally be used in some parts of the network).

The following diagram shows the name resolution process expressed in terms of DNS queries and responses.

The stub resolver sends a recursive query message (with the required domain name in the QUESTION section of the query) (2) to the resolver. A recursive query simply requests the resolver to find the complete answer. A stub resolver only ever sends recursive queries and always needs the service of a resolver. The response to a recursive query can be:

1. The answer to the user’s QUESTION in the ANSWER section of the query response.
2. An error (such as NXDOMAIN - the name does not exist).
Recursive and Iterative Queries

www.example.com (QNAME)

1. Stub Resolver (cache)
2. PC
3. Area Resolver (ISP, SP, or Private Network)
4. DNS Resolver (Cache)
5. DNS example.com
6. DNS .com TLD
7. DNS root-servers

Note: Each numbered line represents a query/answer pair

Item (2) is a Recursive query; one question gives one complete answer. Items (3), (4), and (5) are Iterative queries which may return either a referral, an answer, or an error.

Fig. 4: Resolvers and Queries
The resolver, on receipt of the user’s recursive query, either responds immediately, if the ANSWER is in its cache, or accesses the DNS hierarchy to obtain the answer. The resolver always starts with root servers and sends an iterative query (4, 5, and 6). The response to an iterative query can be:

1. The answer to the resolver’s QUESTION in the ANSWER section of the query response.
2. A referral (indicated by an empty ANSWER section but data in the AUTHORITY section, and typically IP addresses in the ADDITIONAL section of the response).
3. An error (such as NXDOMAIN - the name does not exist).

If the response is either an answer or an error, these are returned immediately to the user (and cached for future use). If the response is a referral, the resolver needs to take additional action to respond to the user’s recursive query.

A referral, in essence, indicates that the queried server does not know the answer (the ANSWER section of the response is empty), but it refers the resolver to the authoritative name servers (in the AUTHORITY section of the response) which it knows about in the domain name supplied in the QUESTION section of the query. Thus, if the QUESTION is for the domain name www.example.com, the root server to which the iterative query was sent adds a list of the .com authoritative name servers in the AUTHORITY section. The resolver selects one of the servers from the AUTHORITY section and sends an iterative query to it. Similarly, the .com authoritative name servers send a referral containing a list of the example.com authoritative name servers. This process continues down the DNS hierarchy until either an ANSWER or an error is received, at which point the user’s original recursive query is sent a response.

Note: The DNS hierarchy is always accessed starting at the root servers and working down; there is no concept of “up” in the DNS hierarchy. Clearly, if the resolver has already cached the list of .com authoritative name servers and the user’s recursive query QUESTION contains a domain name ending in .com, it can omit access to the root servers. However, that is simply an artifact (in this case a performance benefit) of caching and does not change the concept of top-down access within the DNS hierarchy.

The insatiably curious may find reading RFC 1034 and RFC 1035 a useful starting point for further information.

1.4.6 DNS and BIND 9

BIND 9 is a complete implementation of the DNS protocol. BIND 9 can be configured (using its named.conf file) as an authoritative name server, a resolver, and, on supported hosts, a stub resolver. While large operators usually dedicate DNS servers to a single function per system, smaller operators will find that BIND 9’s flexible configuration features support multiple functions, such as a single DNS server acting as both an authoritative name server and a resolver.

Example configurations of basic authoritative name servers and resolvers and forwarding resolvers, as well as advanced configurations and secure configurations, are provided.

1.5 DNS Security Overview

DNS is a communications protocol. All communications protocols are potentially vulnerable to both subversion and eavesdropping. It is important for users to audit their exposure to the various threats within their operational environment and implement the appropriate solutions. BIND 9, a specific implementation of the DNS protocol, provides an extensive set of security features. The purpose of this section is to help users to select from the range of available security features those required for their specific user environment.

A generic DNS network is shown below, followed by text descriptions. In general, the further one goes from the left-hand side of the diagram, the more complex the implementation.
Note: Historically, DNS data was regarded as public and security was concerned, primarily, with ensuring the integrity of DNS data. DNS data privacy is increasingly regarded as an important dimension of overall security, specifically DNS over TLS.

The following notes refer to the numbered elements in the above diagram.

1. A variety of system administration techniques and methods may be used to secure BIND 9’s local environment, including file permissions, running BIND 9 in a jail, and the use of Access Control Lists.

2. The remote name daemon control (rndc) program allows the system administrator to control the operation of a name server. The majority of BIND 9 packages or ports come preconfigured with local (loopback address) security preconfigured. If rndc is being invoked from a remote host, further configuration is required. The nsupdate tool uses Dynamic DNS (DDNS) features and allows users to dynamically change the contents of the zone file(s). nsupdate access and security may be controlled using named.conf statements or using TSIG or SIG(0) cryptographic methods. Clearly, if the remote hosts used for either rndc or DDNS lie within a network entirely under the user’s control, the security threat may be regarded as non-existent. Any implementation requirements, therefore, depend on the site’s security policy.

3. Zone transfer from a primary to one or more secondary authoritative name servers across a public network carries risk. The zone transfer may be secured using named.conf statements, TSIG cryptographic methods or TLS. Clearly, if the secondary authoritative name server(s) all lie within a network entirely under the user’s control, the security threat may be regarded as non-existent. Any implementation requirements again depend on the site’s security policy.

4. If the operator of an authoritative name server (primary or secondary) wishes to ensure that DNS responses to user-initiated queries about the zone(s) for which they are responsible can only have come from their server, that the data received by the user is the same as that sent, and that non-existent names are genuine, then DNSSEC is the only solution. DNSSEC requires configuration and operational changes both to the authoritative name servers and to any resolver which accesses those servers.

5. The typical Internet-connected end-user device (PCs, laptops, and even mobile phones) either has a stub resolver or operates via a DNS proxy. A stub resolver requires the services of an area or full-service resolver to completely
answer user queries. Stub resolvers on the majority of PCs and laptops typically have a caching capability to increase performance. At this time there are no standard stub resolvers or proxy DNS tools that implement DNSSEC. BIND 9 may be configured to provide such capability on supported Linux or Unix platforms. DNS over TLS may be configured to verify the integrity of the data between the stub resolver and area (or full-service) resolver. However, unless the resolver and the Authoritative Name Server implements DNSSEC, end-to-end integrity (from authoritative name server to stub resolver) cannot be guaranteed.
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

The current support status of BIND 9 versions across various platforms 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 modern C11 (or better) compiler, BSD-style sockets with RFC-compliant IPv6 support, POSIX-compliant threads, and the required libraries.

The following C11 features are required to compile BIND 9:

- Atomic operations support defined in `<stdatomic.h>`
- Thread Local Storage support defined in `<threads.h>`

Where it makes sense, BIND 9 uses C-standard fixes introduced by C17 update of the C11 standard.

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

Current versions of BIND 9 are fully supported and regularly tested on the following systems:

- Debian 11, 12
- Ubuntu LTS 20.04, 22.04, 24.04
- Fedora 40
- Red Hat Enterprise Linux / CentOS / Oracle Linux 8, 9
- FreeBSD 13.3, 14.0
- Alpine Linux 3.20

The amd64 CPU architecture is fully supported and regularly tested.

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. None of these are tested regularly by ISC.

- macOS 10.12+
- Solaris 11
- NetBSD
- OpenBSD
- Other Linux distributions still supported by their vendors, such as:
  - Ubuntu 22.10+
  - Gentoo
  - Arch Linux
- OpenWRT/LEDE 17.01+
- Other CPU architectures (arm, arm64, mips64, ppc64, s390x)
2.5.3 Community-Maintained

These systems may not all have the required dependencies for building BIND easily available, although it is 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, 18.04 (Ubuntu ESM releases are not supported)
  - Red Hat Enterprise Linux / CentOS / Oracle Linux 6, 7
  - Debian 8 Jessie, 9 Stretch, 10 Buster
  - FreeBSD 10.x, 11.x
- Less common CPU architectures (i386, i686, mips, mipsel, sparc, ppc, and others)

2.6 Unsupported Platforms

These are platforms on which current versions of BIND 9 are known not to build or run:

- Platforms without at least OpenSSL 1.0.2
- Windows
- Solaris 10 and older
- Platforms that do not support IPv6 Advanced Socket API (RFC 3542)
- Platforms that do not support atomic operations (via compiler or library)
- Linux without NPTL (Native POSIX Thread Library)
- Platforms on which libuv >= 1.34 cannot be compiled or is not available

2.7 Installing BIND 9

Building BIND 9 contains complete instructions for how to build BIND 9.

The ISC Knowledgebase contains many useful articles about installing BIND 9 on specific platforms.
3.1 Introduction

BIND 9 uses a single configuration file called named.conf, which is typically located in either /etc/namedb or /usr/local/etc/namedb.

Note: If rndc is being used locally (on the same host as BIND 9) then an additional file rndc.conf may be present, though rndc operates without this file. If rndc is being run from a remote host then an rndc.conf file must be present as it defines the link characteristics and properties.

Depending on the functionality of the system, one or more zone files is required.

The samples given throughout this and subsequent chapters use a standard base format for both the named.conf and the zone files for example.com. The intent is for the reader to see the evolution from a common base as features are added or removed.

3.1.1 named.conf Base File

This file illustrates the typical format and layout style used for named.conf and provides a basic logging service, which may be extended as required by the user.

```plaintext
// base named.conf file
// Recommended that you always maintain a change log in this file as shown here
// options clause defining the server-wide properties
options {
    // all relative paths use this directory as a base
directory "/var";
    // version statement for security to avoid hacking known weaknesses
    // if the real version number is revealed
    version "not currently available";
};

// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)
//
logging {
    channel example_log {
        // uses a relative path name and the directory statement to
```
The `logging` and `options` blocks and `category`, `channel`, `directory`, `file`, and `severity` statements are all described further in the appropriate sections of this ARM.

### 3.1.2 example.com base zone file

The following is a complete zone file for the domain `example.com`, which illustrates a number of common features. Comments in the file explain these features where appropriate. Zone files consist of Resource Records (RR), which describe the zone’s characteristics or properties.

```plaintext
; base zone file for example.com
$TTL 2d ; default TTL for zone
$ORIGIN example.com. ; base domain-name
; Start of Authority RR defining the key characteristics of the zone (domain)
@ IN SOA ns1.example.com. hostmaster.example.com. (
    2003080800 ; serial number
    12h ; refresh
    15m ; update retry
    3w ; expiry
    2h ; minimum
)
; name server RR for the domain
IN NS ns1.example.com.
; the second name server is external to this zone (domain)
IN NS ns2.example.net.
; mail server RRs for the zone (domain)
3w IN MX 10 mail.example.com.
; the second mail servers is external to the zone (domain)
IN MX 20 mail.example.net.
; domain hosts includes NS and MX records defined above
; plus any others required
; for instance a user query for the A RR of joe.example.com will
; return the IPv4 address 192.168.254.6 from this zone file
ns1 IN A 192.168.254.2
mail IN A 192.168.254.4
joe IN A 192.168.254.6
www IN A 192.168.254.7
; aliases ftp (ftp server) to an external domain
ftp IN CNAME ftp.example.net.
```

This type of zone file is frequently referred to as a forward-mapped zone file, since it maps domain names to some other value, while a reverse-mapped zone file maps an IP address to a domain name. The zone file is called `example.com` for no good reason except that it is the domain name of the zone it describes; as always, users are free to use whatever file-naming convention is appropriate to their needs.
3.1.3 Other Zone Files

Depending on the configuration additional zone files may or should be present. Their format and functionality are briefly described here.

3.1.4 localhost Zone File

All end-user systems are shipped with a hosts file (usually located in /etc). This file is normally configured to map the name localhost (the name used by applications when they run locally) to the loopback address. It is argued, reasonably, that a forward-mapped zone file for localhost is therefore not strictly required. This manual does use the BIND 9 distribution file localhost-forward.db (normally in /etc/namedb/master or /usr/local/etc/namedb/master) in all configuration samples for the following reasons:

1. Many users elect to delete the hosts file for security reasons (it is a potential target of serious domain name redirection/poisoning attacks).

2. Systems normally lookup any name (including domain names) using the hosts file first (if present), followed by DNS. However, the nsswitch.conf file (typically in /etc) controls this order (normally hosts: file dns), allowing the order to be changed or the file value to be deleted entirely depending on local needs. Unless the BIND administrator controls this file and knows its values, it is unsafe to assume that localhost is forward-mapped correctly.

3. As a reminder to users that unnecessary queries for localhost form a non-trivial volume of DNS queries on the public network, which affects DNS performance for all users.

Users may, however, elect at their discretion not to implement this file since, depending on the operational environment, it may not be essential.

The BIND 9 distribution file localhost-forward.db format is shown for completeness and provides for both IPv4 and IPv6 localhost resolution. The zone (domain) name is localhost.

```plaintext
$TTL 3h
@ IN SOA localhost. nobody.localhost. ( 2007091701 ; serial
30800 ; refresh
7200 ; retry
604800 ; expire
300 ) ; minimum
NS localhost.
A 127.0.0.1
AAAA ::1
```

Note: Readers of a certain age or disposition may note the reference in this file to the late, lamented Douglas Noel Adams.

3.1.5 localhost Reverse-Mapped Zone File

This zone file allows any query requesting the name associated with the loopback IP (127.0.0.1). This file is required to prevent unnecessary queries from reaching the public DNS hierarchy. The BIND 9 distribution file localhost.rev is shown for completeness:

```plaintext
$TTL 1D
@ IN SOA localhost. root.localhost. ( 2007091701 ; serial
30800 ; refresh
7200 ; retry
604800 ; expire
300 ) ; minimum
```

(continues on next page)
3.2 Authoritative Name Servers

These provide authoritative answers to user queries for the zones they support: for instance, the zone data describing the domain name example.com. An authoritative name server may support one or many zones.

Each zone may be defined as either a primary or a secondary. A primary zone reads its zone data directly from a file system. A secondary zone obtains its zone data from the primary zone using a process called zone transfer. Both the primary and the secondary zones provide authoritative data for their zone; there is no difference in the answer to a query from a primary or a secondary zone. An authoritative name server may support any combination of primary and secondary zones.

Note: The terms primary and secondary do not imply any access priority. Resolvers (name servers that provide the complete answers to user queries) are not aware of (and cannot find out) whether an authoritative answer comes from the primary or secondary name server. Instead, the resolver uses the list of authoritative servers for the zone (there must be at least two) and maintains a Round Trip Time (RTT) - the time taken to respond to the query - for each server in the list. The resolver uses the lowest-value server (the fastest) as its preferred server for the zone and continues to do so until its RTT becomes higher than the next lowest in its list, at which time that one becomes the preferred server.

For reasons of backward compatibility BIND 9 treats “primary” and “master” as synonyms, as well as “secondary” and “slave.”

The following diagram shows the relationship between the primary and secondary name servers. The text below explains the process in detail.

The numbers in parentheses in the following text refer to the numbered items in the diagram above.

1. The authoritative primary name server always loads (or reloads) its zone files from (1) a local or networked file store.

2. The authoritative secondary name server always loads its zone data from a primary via a zone transfer operation. Zone transfer may use AXFR (complete zone transfer) or IXFR (incremental zone transfer), but only if both primary and secondary name servers support the service. The zone transfer process (either AXFR or IXFR) works as follows:
   a) The secondary name server for the zone reads (3 and 4) the SOA RR periodically. The interval is defined by the refresh parameter of the Start of Authority (SOA) RR.
   b) The secondary compares the serial number parameter of the SOA RR received from the primary with the serial number in the SOA RR of its current zone data.
   c) If the received serial number is arithmetically greater (higher) than the current one, the secondary initiates a zone transfer (5) using AXFR or IXFR (depending on the primary and secondary configuration), using TCP over port 53 (6).

3. The typically recommended zone refresh times for the SOA RR (the time interval when the secondary reads or polls the primary for the zone SOA RR) are multiples of hours to reduce traffic loads. Worst-case zone change propagation can therefore take extended periods.

4. The optional NOTIFY (RFC 1996) feature (2) is automatically configured; use the notify statement to turn off the feature. Whenever the primary loads or reloads a zone, it sends a NOTIFY message to the configured secondary (or secondaries) and may optionally be configured to send the NOTIFY message to other hosts using the also-notify statement. The NOTIFY message simply indicates to the secondary that the primary has loaded or reloaded the zone. On receipt of the NOTIFY message, the secondary responds to indicate it has received the
3.2. Authoritative Name Servers

Fig. 1: Authoritative Primary and Secondary Name Servers
NOTIFY and immediately reads the SOA RR from the primary (as described in section 2 a. above). If the zone file has changed, propagation is practically immediate.

The authoritative samples all use NOTIFY but identify the statements used, so that they can be removed if not required.

### 3.2.1 Primary Authoritative Name Server

The zone files are unmodified *from the base samples* but the `named.conf` file has been modified as shown:

```plaintext
// authoritative primary named.conf file
// options clause defining the server-wide properties
options {
  // all relative paths use this directory as a base
directory "var";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is revealed
  version "not currently available";
  // This is the default - allows user queries from any IP
  allow-query { any; };
  // normal server operations may place items in the cache
  // this prevents any user query from accessing these items
  allow-query-cache { none; };
  // Do not provide recursive service to user queries
  recursion no;
};
// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)
//
logging {
  channel example_log {
    // uses a relative path name and the directory statement to
    // expand to /var/log/named/example.log
    file "log/named/example.log" versions 3 size 250k;
    // only log info and up messages - all others discarded
    severity info;
  };
  category default {
    example_log;
  };
};
// Provide forward mapping zone for localhost
// (optional)
zone "localhost" {
  type primary;
  file "master/localhost-forward.db";
  notify no;
};
// Provide reverse mapping zone for the loopback
// address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
  type primary;
  file "localhost.rev";
  notify no;
}
```

(continues on next page)
// We are the primary server for example.com
zone "example.com" {
    // this is the primary name server for the zone
type primary;
    file "example.com";
    // this is the default
    notify yes;
    // IP addresses of secondary servers allowed to
    // transfer example.com from this server
    allow-transfer {
        192.168.4.14;
        192.168.5.53;
    };
};

The added statements and blocks are commented in the above file.

The zone block, and allow-query, allow-query-cache, allow-transfer, file, notify, recursion, and type statements are described in detail in the appropriate sections.

### 3.2.2 Secondary Authoritative Name Server

The zone files local-host-forward.db and localhost.rev are unmodified from the base samples. The example.com zone file is not required (the zone file is obtained from the primary via zone transfer). The named.conf file has been modified as shown:

```bash
// authoritative secondary named.conf file
// options clause defining the server-wide properties
options {
    // all relative paths use this directory as a base
directory "/var";
    // version statement for security to avoid hacking known weaknesses
    // if the real version number is revealed
    version "not currently available";
    // This is the default - allows user queries from any IP
    allow-query { any; };
    // normal server operations may place items in the cache
    // this prevents any user query from accessing these items
    // only authoritative zone data will be returned
    allow-query-cache { none; };
    // Do not provide recursive service to user queries
    recursion no;
};
// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)
//
logging {
    channel example_log {
        // uses a relative path name and the directory statement to
        // expand to /var/log/named/example.log
        file "log/named/example.log" versions 3 size 250k;
    }
}(continues on next page)
// only log info and up messages - all others discarded
severity info;
);
category default { example_log;
);
// Provide forward mapping zone for localhost
// (optional)
zone "localhost" { type primary; file "master/localhost-forward.db"; notify no;
};
// Provide reverse mapping zone 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 secondary server for example.com
zone "example.com" {
  // this is a secondary server for the zone
type secondary;
  // the file statement here allows the secondary to save
  // each zone transfer so that in the event of a program restart
  // the zone can be loaded immediately and the server can start
  // to respond to queries without waiting for a zone transfer
  file "example.com.saved";
  // IP address of example.com primary server
  primaries { 192.168.254.2; }
};

The statements and blocks added are all commented in the above file.

The zone block, and allow-query, allow-query-cache, allow-transfer, file, primaries, recursion, and type statements are described in detail in the appropriate sections.

If NOTIFY is not being used, no changes are required in this named.conf file, since it is the primary that initiates the NOTIFY message.

Note: Just when the reader thought they understood primary and secondary, things can get more complicated. 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: in the zone block for the secondary causes named to only send NOTIFY messages for primary zones that it loads.
3.3 Resolver (Caching Name Servers)

Resolvers handle recursive user queries and provide complete answers; that is, they issue one or more iterative queries to the DNS hierarchy. Having obtained a complete answer (or an error), a resolver passes the answer to the user and places it in its cache. Subsequent user requests for the same query will be answered from the resolver’s cache until the TTL of the cached answer has expired, when it will be flushed from the cache; the next user query that requests the same information results in a new series of queries to the DNS hierarchy.

Resolvers are frequently referred to by a bewildering variety of names, including caching name servers, recursive name servers, forwarding resolvers, area resolvers, and full-service resolvers.

The following diagram shows how resolvers can function in a typical networked environment:

Resolver and Forwarding Resolver

1. End-user systems are all distributed with a local stub resolver as a standard feature. Today, the majority of stub resolvers also provide a local cache service to speed up user response times.

2. A stub resolver has limited functionality; specifically, it cannot follow referrals. When a stub resolver receives a request for a name from a local program, such as a browser, and the answer is not in its local cache, it sends a recursive user query (1) to a locally configured resolver (5), which may have the answer available in its cache. If it does not, it issues iterative queries (2) to the DNS hierarchy to obtain the answer. The resolver to which the local system sends the user query is configured, for Linux and Unix hosts, in /etc/resolv.conf; for Windows users it is configured or changed via the Control Panel or Settings interface.

3. Alternatively, the user query can be sent to a forwarding resolver (4). Forwarding resolvers on first glance look fairly pointless, since they appear to be acting as a simple pass-through and, like the stub resolver, require a full-service resolver (5). However, forwarding resolvers can be very powerful additions to a network for the following reasons:

   a) Cost and Performance. Each recursive user query (1) at the forwarding resolver (4) results in two messages - the query and its answer. The resolver (5) may have to issue three, four, or more query pairs (2) to get the required answer. Traffic is reduced dramatically, increasing performance or reducing cost (if the link is tariffed). Additionally, since the forwarding resolver is typically shared across multiple hosts, its cache is more likely to contain answers, again improving user performance.

   b) Network Maintenance. Forwarding resolvers (4) can be used to ease the burden of local administration by providing a single point at which changes to remote name servers can be managed, rather than having to update all hosts. Thus, all hosts in a particular network section or area can be configured to point to a forwarding resolver, which can be configured to stream DNS traffic as desired and changed over time with minimal effort.

   c) Sanitizing Traffic. Especially in larger private networks it may be sensible to stream DNS traffic using a forwarding resolver structure. The forwarding resolver (4) may be configured, for example, to handle all in-domain traffic (relatively safe) and forward all external traffic to a hardened resolver (5).

   d) Stealth Networks. Forwarding resolvers are extensively used in stealth or split networks.
4. Forwarding resolvers (4) can be configured to forward all traffic to a resolver (5), or to only forward selective traffic (5) while directly resolving other traffic (3).

**Attention:** While the diagram above shows recursive user queries arriving via interface (1), there is nothing to stop them from arriving via interface (2) via the public network. If no limits are placed on the source IPs that can send such queries, the resolver is termed an open resolver. Indeed, when the world was young this was the way things worked on the Internet. Much has changed and what seems to be a friendly, generous action can be used by rogue actors to cause all kinds of problems including Denial of Service (DoS) attacks. Resolvers should always be configured to limit the IP addresses that can use their services. BIND 9 provides a number of statements and blocks to simplify defining these IP limits and configuring a closed resolver. The resolver samples given here all configure closed resolvers using a variety of techniques.

### 3.3.1 Additional Zone Files

#### Root Servers (Hint) Zone File

Resolvers (although not necessarily forwarding resolvers) need to access the DNS hierarchy. To do this, they need to know the addresses (IPv4 and/or IPv6) of the 13 root servers. This is done by the provision of a root server zone file, which is contained in the standard BIND 9 distribution as the file named.root (normally found in /etc/namedb or /usr/local/namedb). This file may also be obtained from the IANA website (https://www.iana.org/domains/root/files).

**Note:** Many distributions rename this file for historical reasons. Consult the appropriate distribution documentation for the actual file name.

The hint zone file is referenced using the type hint statement and a zone (domain) name of “.” (the generally silent dot).

**Note:** The root server IP addresses have been stable for a number of years and are likely to remain stable for the near future. BIND 9 has a root-server list in its executable such that even if this file is omitted, out-of-date, or corrupt BIND 9 can still function. For this reason, many sample configurations omit the hints file. All the samples given here include the hints file primarily as a reminder of the functionality of the configuration, rather than as an absolute necessity.

#### Private IP Reverse Map Zone Files

Resolvers are configured to send iterative queries to the public DNS hierarchy when the information requested is not in their cache or not defined in any local zone file. Many networks make extensive use of private IP addresses (defined by RFC 1918, RFC 2193, RFC 5737, and RFC 6598). By their nature these IP addresses are forward-mapped in various user zone files. However, certain applications may issue reverse map queries (mapping an IP address to a name). If the private IP addresses are not defined in one or more reverse-mapped zone file(s), the resolver sends them to the DNS hierarchy where they are simply useless traffic, slowing down DNS responses for all users.

Private IP addresses may be defined using standard reverse-mapping techniques or using the empty-zones-enable statement. By default this statement is set to empty-zones-enable yes; and thus automatically prevents unnecessary DNS traffic by sending an NXDOMAIN error response (indicating the name does not exist) to any request. However, some applications may require a genuine answer to such reverse-mapped requests or they will fail to function. Mail systems in particular perform reverse DNS queries as a first-line spam check; in this case a reverse-mapped zone file is essential. The sample configuration files given here for both the resolver and the forwarding resolver provide a reverse-mapping zone file for the private IP address 192.168.254.4, which is the mail server address in the base zone.
file, as an illustration of the reverse-map technique. The file is named 192.168.254.rev and has a zone name of 254.168.192.in-addr.arpa.

```plaintext
; reverse map zone file for 192.168.254.4 only
$TTL 2d ; 172800 seconds
$ORIGIN 254.168.192.IN-ADDR.ARPA.
@ IN SOA ns1.example.com. hostmaster.example.com. ( 2003080800 ; serial number
3h ; refresh
15m ; update retry
3w ; expiry
3h ; nx = nxdomain ttl )
; only one NS is required for this local file
; and is an out of zone name
   IN NS ns1.example.com.
; other IP addresses can be added as required
; this maps 192.168.254.4 as shown
 4 IN PTR mail.example.com. ; fully qualified domain name (FQDN)
```

### 3.3.2 Resolver Configuration

The resolver provides recursive query support to a defined set of IP addresses. It is therefore a closed resolver and cannot be used in wider network attacks.

```plaintext
// resolver named.conf file
// Two corporate subnets we wish to allow queries from
// defined in an acl clause
acl corpnets { 192.168.4.0/24;
                   192.168.7.0/24;
};

// options clause defining the server-wide properties
options {
    // all relative paths use this directory as a base
    directory "/var";
    // version statement for security to avoid hacking known weaknesses
    // if the real version number is revealed
    version "not currently available";
    // this is the default
    recursion yes;
    // recursive queries only allowed from these ips
    // and references the acl clause
    allow-query { corpnets; };
    // this ensures that any reverse map for private IPs
    // not defined in a zone file will *not* be passed to the public network
    // it is the default value
    empty-zones-enable yes;
};

// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)
```

(continues on next page)
The zone and acl blocks, and the allow-query, empty-zones-enable, file, notify, recursion, and type statements are described in detail in the appropriate sections.

As a reminder, the configuration of this resolver does not access the DNS hierarchy (does not use the public network) for any recursive query for which:

1. The answer is already in the cache.
2. The domain name is localhost (zone localhost).
3. Is a reverse-map query for 127.0.0.1 (zone 0.0.127.in-addr.arpa).
5. Is a reverse-map query for any local IP (`empty-zones-enable` statement).

All other recursive queries will result in access to the DNS hierarchy to resolve the query.

### 3.3.3 Forwarding Resolver Configuration

This forwarding resolver configuration forwards all recursive queries, other than those for the defined zones and those for which the answer is already in its cache, to a full-service resolver at the IP address 192.168.250.3, with an alternative at 192.168.230.27. The forwarding resolver will cache all responses from these servers. The configuration is closed, in that it defines those IPs from which it will accept recursive queries.

A second configuration in which selective forwarding occurs is also provided.

```plaintext
// forwarding named.conf file
// Two corporate subnets we wish to allow queries from
// defined in an acl clause
acl corpnets {
  192.168.4.0/24;
  192.168.7.0/24;
};

// options clause defining the server-wide properties
options {
  // all relative paths use this directory as a base
directory "/var";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is revealed
  version "not currently available";
  // this is the default
  recursion yes;
  // recursive queries only allowed from these ips
  // and references the acl clause
  allow-query { corpnets; };
  // this ensures that any reverse map for private IPs
  // not defined in a zone file will *not* be passed to the public network
  // it is the default value
  empty-zones-enable yes;
  // this defines the addresses of the resolvers to which queries will be forwarded
  forwarders {
    192.168.250.3;
    192.168.230.27;
  };  // indicates all queries will be forwarded other than for defined zones
  forward only;
};

// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)
//
logging {
  channel example_log {
    // uses a relative path name and the directory statement to
    // expand to /var/log/named/example.log
    file "log/named/example.log" versions 3 size 250k;
  }
} (continues on next page)
```
The `zone` and `acl` blocks, and the `allow-query`, `empty-zones-enable`, `file`, `forward`, `forwarders`, `notify`, `recursion`, and `type` statements are described in detail in the appropriate sections.

As a reminder, the configuration of this forwarding resolver does not forward any recursive query for which:

1. The answer is already in the cache.
2. The domain name is `localhost` (zone localhost).
3. Is a reverse-map query for 127.0.0.1 (zone 0.0.127.in-addr.arpa).
5. Is a reverse-map query for any local IP (`empty-zones-enable` statement).

All other recursive queries will be forwarded to resolve the query.
3.3.4 Selective Forwarding Resolver Configuration

This forwarding resolver configuration only forwards recursive queries for the zone example.com to the resolvers at 192.168.250.3 and 192.168.230.27. All other recursive queries, other than those for the defined zones and those for which the answer is already in its cache, are handled by this resolver. The forwarding resolver will cache all responses from both the public network and from the forwarded resolvers. The configuration is closed, in that it defines those IPs from which it will accept recursive queries.

```
// selective forwarding named.conf file
// Two corporate subnets we wish to allow queries from
// defined in an acl clause
acl corpnets {
  192.168.4.0/24;
  192.168.7.0/24;
};

// options clause defining the server-wide properties
options {
  // all relative paths use this directory as a base
directory "/var";
  // version statement for security to avoid hacking known weaknesses
  // if the real version number is revealed
  version "not currently available";
  // this is the default
  recursion yes;
  // recursive queries only allowed from these ips
  // and references the acl clause
  allow-query { corpnets; };
  // this ensures that any reverse map for private IPs
  // not defined in a zone file will *not* be passed to the public network
  // it is the default value
  empty-zones-enable yes;

  // forwarding is not global but selective by zone in this configuration
};

// logging clause
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// uses 3 files in rotation swaps files when size reaches 250K
// failure messages that occur before logging is established are
// in syslog (/var/log/messages)

logging {
  // uses a relative path name and the directory statement to
  // expand to /var/log/named/example.log
  channel example_log {
    file "log/named/example.log" versions 3 size 250k;
    // only log info and up messages - all others discarded
    severity info;
  }
  category default {
    example_log;
  }
};

// zone file for the root servers
// discretionary zone (see root server discussion above)
```

(continues on next page)
The zone and acl blocks, and the allow-query, empty-zones-enable, file, forward, forwarders, notify, recursion, and type statements are described in detail in the appropriate sections.

As a reminder, the configuration of this resolver does not access the DNS hierarchy (does not use the public network) for any recursive query for which:

1. The answer is already in the cache.
2. The domain name is localhost (zone localhost).
3. Is a reverse-map query for 127.0.0.1 (zone 0.0.127.in-addr.arpa).
5. Is a reverse-map query for any local IP (empty-zones-enable statement).
6. Is a query for the domain name example.com, in which case it will be forwarded to either 192.168.250.3 or 192.168.230.27 (zone example.com).
All other recursive queries will result in access to the DNS hierarchy to resolve the query.

### 3.4 Load Balancing

A primitive form of load balancing can be achieved in the DNS by using multiple resource records (RRs) in a **zone file** (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:

<table>
<thead>
<tr>
<th>Name</th>
<th>TTL</th>
<th>CLASS</th>
<th>TYPE</th>
<th>Resource Record (RR) Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>www</td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.2</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.3</td>
</tr>
</tbody>
</table>

When a resolver queries for these records, BIND rotates them and responds to the query with the records in a random 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, refer to the `rrset-order` statement in the `options` block.

### 3.5 Zone File

This section, largely borrowed from [RFC 1034](https://www.rfc-editor.org/rfc/rfc1034), describes the concept of a Resource Record (RR) and explains how to use them.

#### 3.5.1 Resource Records

A domain name identifies a node in the DNS tree namespace. 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 `sortlist` and `RRset Ordering`.

The components of a Resource Record are:

- **owner name**
  - The domain name where the RR is found.

- **RR type**
  - An encoded 16-bit value that specifies the type of the resource record. For a list of *types* of valid RRs, including those that have been obsoleted, please refer to [https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4](https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4).

- **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.
The following classes of resource records are currently valid in the DNS:

**IN**

The Internet. The only widely class used today.

**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:

<table>
<thead>
<tr>
<th>ISLEDU</th>
<th>MX</th>
<th>10 VENERA.ISLEDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>VENERA.ISLEDU</td>
<td>A</td>
<td>128.9.0.32</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>10.1.0.52</td>
</tr>
<tr>
<td>VAXA.ISLEDU</td>
<td>A</td>
<td>10.2.0.27</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>128.9.0.33</td>
</tr>
</tbody>
</table>

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:

```
<table>
<thead>
<tr>
<th>Domain</th>
<th>Class</th>
<th>Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX.LCS/MIT.EDU.</td>
<td>CH</td>
<td>A</td>
<td>10.0.0.44</td>
</tr>
<tr>
<td>XX.LCS/MIT.EDU.</td>
<td>CH</td>
<td>MIT.EDU</td>
<td>2420</td>
</tr>
</tbody>
</table>
```

This shows two addresses for XX.LCS/MIT.EDU, each of a different class.

### 3.5.2 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:

```
<table>
<thead>
<tr>
<th>Domain</th>
<th>Class</th>
<th>Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.com.</td>
<td>IN</td>
<td>MX</td>
<td>10 mail.example.com.</td>
</tr>
<tr>
<td>mail.example.com.</td>
<td>IN</td>
<td>MX</td>
<td>10 mail2.example.com.</td>
</tr>
<tr>
<td>mail.example.com.</td>
<td>IN</td>
<td>MX</td>
<td>20 mail.backup.org.</td>
</tr>
<tr>
<td>mail2.example.com.</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>mail2.example.com.</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.2</td>
</tr>
</tbody>
</table>
```

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.

### 3.5.3 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 minimum**

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. Further details can be found in RFC 2308.

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**.
3.5.4 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 3.2.1.0.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 2.10.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.

3.5.5 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 $INCLUDE Directive 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.

3.5.6 BIND Primary File Extension: the $GENERATE Directive

Syntax: $GENERATE range owner [ttl] [class] type rdata [comment]

$GENERATE is used to create a series of resource records that only differ from each other by an iterator.

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.

The owner string may include one or more $ (dollar sign) symbols, which will be replaced with the iterator value when generating records; see below for details.

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. As with owner, the rdata string may include one or more $ symbols, which are replaced with the iterator value. rdata may be quoted if there are spaces in the string; the quotation marks do not appear in the generated record.
Any single \$ (dollar sign) symbols within the owner or rdata strings are replaced by the iterator value. To get a \$ in the output, escape the \$ using a backslash \ e.g., \$. (For compatibility with earlier versions, $$ is also recognized as indicating a literal $ in the output.)

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 modifier cannot contain whitespace or newlines.

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.

Examples:

$GENERATE can be used to easily generate the sets of records required to support sub-/24 reverse delegations described in RFC 2317:

```bash
$ORIGIN 0.0.192.IN-ADDR.ARPA.
$GENERATE 1-2 @ NS SERVER$.EXAMPLE.
$GENERATE 1-127 $ CNAME $.0
```

is equivalent to

```plaintext
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.
```

This example creates a set of A and MX records. Note the MX's rdata is a quoted string; the quotes are stripped when $GENERATE is processed:

```bash
$ORIGIN EXAMPLE.
$GENERATE 1-127 HOST-\$ A 1.2.3.\$
$GENERATE 1-127 HOST-\$ MX "0 ".
```

is equivalent to

```plaintext
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.
```

This example generates A and AAAA records using modifiers; the AAAA owner names are generated using nibble mode:

```bash
$ORIGIN EXAMPLE.
$GENERATE 0-2 HOST-$\{0,4,d\} A 1.2.3.$\{1,0,d\}
$GENERATE 1024-1026 $\{0,3,n\} AAAA 2001:db8:$$\{0,4,x\}
```

is equivalent to:
The $GENERATE directive is a BIND extension and not part of the standard zone file format.

### 3.5.7 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.

For a primary server, a zone file in **raw** 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 **raw** format needs manual modification, it first must be converted to **text** format by the `named-compilezone` command, then converted back after editing. For example:

```bash
named-compilezone -f raw -F text -o zonefile.text <origin> zonefile.raw
[edit zonefile.text]
named-compilezone -f text -F raw -o zonefile.raw <origin> zonefile.text
```
4.1 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.

4.1.1 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.

4.1.2 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*.

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

**options**

Grammar:

```
options {
  default-key <string>;  
  default-port <integer>;  
  default-server <string>;  
  default-source-address ( <ipv4_address> | * );  
  default-source-address-v6 ( <ipv6_address> | * );
};
```

**Blocks:** topmost

**default-server**

Grammar: default-server <string>;

**Blocks:** options

*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**

Grammar: default-key <string>;

**Blocks:** options

*default-key* takes the name of a key as its argument, as defined by a *key* block.

**default-port**

Grammar: default-port <integer>;

**Blocks:** options
**default-port** specifies the port to which **rndc** should connect if no port is given on the command line or in a **server** block.

**default-source-address**

Grammar: `default-source-address ( <ipv4_address> | * );`

Blocks: `options`

**default-source-address-v6**

Grammar: `default-source-address-v6 ( <ipv6_address> | * );`

Blocks: `options`

**default-source-address** and **default-source-address-v6** specify the IPv4 and IPv6 source address used to communicate with the server if no address is given on the command line or in a **server** block.

**key**

Grammar server: `key <string> ;`

Grammar topmost:

```
key <string> { 
  algorithm <string> ;
  secret <string> ;
}; // may occur multiple times
```

Blocks: `topmost`, `server`

The **key** block 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** block has two statements: **algorithm** and **secret**.

**algorithm**

Grammar: `algorithm <string> ;`

Blocks: `key`

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.

**secret**

Grammar: `secret <string> ;`

Blocks: `key`

The secret is a Base64-encoded string as specified in **RFC 3548**.

**server**

Grammar:

```
server <string> { 
  addresses ( { <quoted_string> [ port <integer> ] | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] } ); ... );
  key <string> ;
  port <integer> ;
  source-address ( <ipv4_address> | * );
  source-address-v6 ( <ipv6_address> | * );
}; // may occur multiple times
```

4.1. Tools for Use With the Name Server Daemon
Blocks: topmost

The server block specifies connection parameters for a given server. The server can be specified as a host name or address.

addresses
  Grammar: addresses { ( <quoted_string> [ port <integer> ] | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) ; ... };
  Blocks: server
  Specifies one or more addresses to use when communicating with this server.

key
  Associates a key defined using the key statement with a server.

port
  Grammar: port <integer> ;
  Blocks: server
  Specifies the port rndc should connect to on the server.

source-address
  Grammar: source-address ( <ipv4_address> | * ) ;
  Blocks: server

source-address-v6
  Grammar: source-address-v6 ( <ipv6_address> | * ) ;
  Blocks: server
  Overrides default-source-address and default-source-address-v6 for this specific server.

A sample minimal configuration file is as follows:

```plaintext
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:

```plaintext
controls {
  inet 127.0.0.1
    allow { localhost; } keys { rndc_key; };
};
```
and it has an identical key block for \texttt{rndc\_key}.

Running the \texttt{rndc\_confgen} program conveniently creates an \texttt{rndc\_conf} file, and also displays the corresponding \texttt{controls} statement needed to add to \texttt{named.conf}. Alternatively, it is possible to run \texttt{rndc\_confgen -a} to set up an \texttt{rndc\_key} file and not modify \texttt{named.conf} at all.

4.2 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 \texttt{kill} command.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>Causes the server to read \texttt{named.conf} and reload the database.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Causes the server to clean up and exit.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>Causes the server to clean up and exit.</td>
</tr>
</tbody>
</table>

4.3 Plugins

Plugins are a mechanism to extend the functionality of \texttt{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 \texttt{filter-aaaa.so}, which replaces the \texttt{filter-aaaa} feature that previously existed natively as part of \texttt{named}. The code for this feature has been removed from \texttt{named} and can no longer be configured using standard \texttt{named.conf} syntax, but linking in the \texttt{filter-aaaa.so} plugin provides identical functionality.

4.4 Configuring Plugins

\texttt{plugin}

\textbf{Grammar}: plugin \{ query \} \textless string\textgreater \{ \{ \textless unspecified-text\textgreater \} \}; // may occur multiple times

\textbf{Blocks}: topmost, view

\textbf{Tags}: server

Configures plugins in \texttt{named.conf}.

A plugin is configured with the \texttt{plugin} statement in \texttt{named.conf}:

\begin{verbatim}
plugin query "library.so" { 
  parameters
};
\end{verbatim}

In this example, file \texttt{library.so} is the plugin library. \texttt{query} indicates that this is a query plugin.

Multiple \texttt{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.

4.5 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`.
DNS Security Extensions (DNSSEC) provide reliable protection from cache poisoning attacks. At the same time these extensions also provide other benefits: they limit the impact of random subdomain attacks on resolver caches and authoritative servers, and provide the foundation for modern applications like authenticated and private e-mail transfer.

To achieve this goal, DNSSEC adds digital signatures to DNS records in authoritative DNS zones, and DNS resolvers verify the validity of the signatures on the received records. If the signatures match the received data, the resolver can be sure that the data was not modified in transit.

Note: DNSSEC and transport-level encryption are complementary! Unlike typical transport-level encryption like DNS-over-TLS, DNS-over-HTTPS, or VPN, DNSSEC makes DNS records verifiable at all points of the DNS resolution chain.

This section focuses on ways to deploy DNSSEC using BIND. For a more in-depth discussion of DNSSEC principles (e.g. How Does DNSSEC Change DNS Lookup?) please see DNSSEC Guide.

5.1 Zone Signing

BIND offers several ways to generate signatures and maintain their validity during the lifetime of a DNS zone:

- **Fully Automated (Key and Signing Policy)** - strongly recommended
- **Manual Signing** - discouraged, use only for debugging

5.1.1 Zone keys

Regardless of the zone-signing method in use, cryptographic keys are stored in files named like Kdnssec.example.+013+12345.key and Kdnssec.example.+013+12345.private. 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. Additionally, the Fully Automated (Key and Signing Policy) method creates a third file, Kdnssec.example+013+12345.state, which is used to track DNSSEC key timings and to perform key rollovers safely.

These filenames contain:

- the key name, which always matches the zone name (dnssec.example.),
- the algorithm number (013 is ECDSAP256SHA256, 008 is RSASHA256, etc.),
- and the key tag, i.e. a non-unique key identifier (12345 in this case).
Warning: Private keys are required for full disaster recovery. Back up key files in a safe location and protect them from unauthorized access. Anyone with access to the private key can create fake but seemingly valid DNS data.

5.1.2 Fully Automated (Key and Signing Policy)

Key and Signing Policy (KASP) is a method of configuration that describes how to maintain DNSSEC signing keys and how to sign the zone.

This is the recommended, fully automated way to sign and maintain DNS zones. For most use cases users can simply use the built-in default policy, which applies up-to-date DNSSEC practices:

```plaintext
zone "dnssec.example" {
    type primary;
    file "dnssec.example.db";
    dnssec-policy default;
};
```

The `dnssec-policy` statement requires dynamic DNS to be set up, or `inline-signing` to be enabled. In the example above we use the latter, because the `default` policy uses `inline-signing`.

This is sufficient to create the necessary signing keys, and generate DNSKEY, RRSIG, and NSEC records for the zone. BIND also takes care of any DNSSEC maintenance for this zone, including replacing signatures that are about to expire and managing `Key Rollovers`.

Note: `dnssec-policy` needs write access to the zone. Please see `dnssec-policy` for more details about implications for zone storage.

The default policy creates one key that is used to sign the complete zone, and uses NSEC to enable authenticated denial of existence (a secure way to tell which records do not exist in a zone). This policy is recommended and typically does not need to be changed.

If needed, a custom policy can be defined by adding a `dnssec-policy` statement into the configuration:

```plaintext
dnssec-policy "custom" {
    dnskey-ttl 600;
    keys {
        ksk lifetime P1Y algorithm ecdsap384sha384;
        zsk lifetime 60d algorithm ecdsap384sha384;
    };
    nsec3param iterations 0 optout no salt-length 0;
}
```

This custom policy, for example:

- uses a very short DNSKEY TTL (600 seconds),
- uses two keys to sign the zone: a Key Signing Key (KSK) to sign the key related RRsets (DNSKEY, CDS, and CDNSKEY), and a Zone Signing Key (ZSK) to sign the rest of the zone. The KSK is automatically rotated after one year and the ZSK after 60 days.

Also:

- The configured keys have a lifetime set and use the ECDSAP384SHA384 algorithm.
- The last line instructs BIND to generate NSEC3 records for Proof of Non-Existence, using zero extra iterations and no salt. NSEC3 opt-out is disabled, meaning insecure delegations also get an NSEC3 record.
For more information about KASP configuration see `dnssec-policy`.

The **Advanced Discussions** section in the DNSSEC Guide discusses the various policy settings and may be useful for determining values for specific needs.

**Key Rollover**

When using a `dnssec-policy`, a key lifetime can be set to trigger key rollovers. ZSK rollovers are fully automatic, but for KSK and CSK rollovers a DS record needs to be submitted to the parent. See **Secure Delegation** for possible ways to do so.

Once the DS is in the parent (and the DS of the predecessor key is withdrawn), BIND needs to be told that this event has happened. This can be done automatically by configuring parental agents:

```bind
zone "dnssec.example" {
    type primary;
    file "dnssec.example.db";
    dnssec-policy default;
    parental-agents { 192.0.2.1; }
    checkds explicit;
};
```

Here one server, `192.0.2.1`, is configured for BIND to send DS queries to, to check the DS RRset for `dnssec-example` during key rollovers. This needs to be a trusted server, because BIND does not validate the response. The `checkds` option makes BIND use the explicitly configured parental agents, rather than looking them up by querying for the parent NS records.

If setting up a parental agent is undesirable, it is also possible to tell BIND that the DS is published in the parent with: `rndc dnssec -checkds -key 12345 published dnssec.example`. and the DS for the predecessor key has been removed with: `rndc dnssec -checkds -key 54321 withdrawn dnssec.example`. where 12345 and 54321 are the key tags of the successor and predecessor key, respectively.

To roll a key sooner than scheduled, or to roll a key that has an unlimited lifetime, use: `rndc dnssec -rollover -key 12345 dnssec.example`.

To revert a signed zone back to an insecure zone, change the zone configuration to use the built-in “insecure” policy. Detailed instructions are described in **Reverting to Unsigned**.

**Multi-Signer Model**

Dynamic zones provide the ability to sign a zone by multiple providers, meaning each provider signs and serves the same zone independently, as is described in **RFC 8901**. BIND 9 is able to support Model 2, where each provider has their own KSK and ZSK (or CSK). The keys from the other provider can be imported via Dynamic Update. For each active KSK there must be a corresponding DS record in the parent zone. Key rollovers require coordination in order to update the DS and DNSKEY RRset.
5.1.3 Manual Signing

There are several tools available to manually sign a zone.

Warning: Please note manual procedures are available mainly for backwards compatibility and should be used only by expert users with specific needs.

To set up a DNSSEC secure zone manually, a series of steps must be followed. Please see chapter Manual Signing in the DNSSEC Guide for more information.

5.1.4 Monitoring with 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

5.2 Secure Delegation

Once a zone is signed on the authoritative servers, the last remaining step is to establish chain of trust between the parent zone (example.) and the local zone (dnssec.example.).

Generally the procedure is:

- Wait for stale data to expire from caches. The amount of time required is equal to the maximum TTL value used in the zone before signing. This step ensures that unsigned data expire from caches and resolvers do not get confused by missing signatures.
- Insert/update DS records in the parent zone (dnssec.example. DS record).

1 For further details on how the chain of trust is used in practice, see The 12-Step DNSSEC Validation Process (Simplified) in the DNSSEC Guide.
There are multiple ways to update DS records in the parent zone. Refer to the documentation for the parent zone to find out which options are applicable to a given case zone. Generally the options are, from most-to least-recommended:

- Automatically update the DS record in the parent zone using CDS/CDNSKEY records automatically generated by BIND. This requires support for RFC 7344 in either parent zone, registry, or registrar. In that case, configure BIND to monitor DS records in the parent zone and everything will happen automatically at the right time.

- Query the zone for automatically generated CDS or CDNSKEY records using dig, and then insert these records into the parent zone using the method specified by the parent zone (web form, e-mail, API, …).

- Generate DS records manually using the dnssec-dsfromkey utility on zone keys, and then insert them into the parent zone.

5.3 DNSSEC Validation

The BIND resolver validates answers from authoritative servers by default. This behavior is controlled by the configuration statement `dnssec-validation`.

By default a trust anchor for the DNS root zone is used. This trust anchor is provided as part of BIND and is kept up-to-date using Dynamic Trust Anchor Management.

Note: DNSSEC validation works “out of the box” and does not require additional configuration. Additional configuration options are intended only for special cases.

To validate answers, the resolver needs at least one trusted starting point, a “trust anchor.” Essentially, trust anchors are copies of DNSKEY RRs for zones that are used to form the first link in the cryptographic chain of trust. Alternative trust anchors can be specified using `trust-anchors`, but this setup is very unusual and is recommended only for expert use. For more information, see Trust Anchors in the DNSSEC Guide.

The BIND authoritative server does not verify signatures on load, so zone keys for authoritative zones do not need to be specified in the configuration file.

5.3.1 Validation Failures

When DNSSEC validation is 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.

For more information see Basic DNSSEC Troubleshooting.

5.3.2 Coexistence With Unsigned (Insecure) Zones

Zones not protected by DNSSEC are called “insecure,” and these zones seamlessly coexist with signed zones.

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.”

5.4 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.

5.4.1 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 the trust-anchors statement description.

5.4.2 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.

5.5 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.

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.

BIND 9 access PKCS#11 libraries via OpenSSL extensions. The extension for OpenSSL 3 and newer is `pkcs11-provider`. And for the older OpenSSL versions engine_pkcs11 from the OpenSC project can be used.

In both cases the extension is dynamically loaded into OpenSSL and the HSM is operated indirectly; any cryptographic operations not supported by the HSM can be carried out by OpenSSL instead.

5.5.1 Prerequisites

See the documentation provided by the HSM vendor for information about installing, initializing, testing, and troubleshooting the HSM.

5.5.2 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
```

5.5.3 OpenSSL 1.x.x with engine_pkcs11

OpenSSL engine-based PKCS#11 uses engine_pkcs11 OpenSSL engine from libp11 project.

engine_pkcs11 tries to fit the PKCS#11 API within the engine API of OpenSSL. That is, it provides a gateway between PKCS#11 modules and the OpenSSL engine API. One has to register the engine with OpenSSL and one has to provide the path to the PKCS#11 module which should be gatewayed to. This can be done by editing the OpenSSL configuration file, by engine specific controls, or by using the p11-kit proxy module.

It is recommended, that libp11 >= 0.4.12 is used.
For more detailed howto including the examples, we recommend reading:


When using engine_pkcs11, all BIND binaries potentially need the keys require `-E pkcs11` argument to activate the engine support.

Even though OpenSSL 3 has compatibility support for Engine API it is not recommended to be used due to bugs in OpenSSL and libp11.

It is not possible to generate new keys via the engine_pkcs11 and therefore it is not recommended to use it in a dnssec-policy setup (although it is possible to put previously generated keys in the key-directory and let the key manager select those keys when a key rollover is started.

### 5.5.4 Configuring engine_pkcs11

The canonical documentation for configuring engine_pkcs11 is in the libp11/README.md, but here’s copy of working configuration for your convenience:

We are going to use our own custom copy of OpenSSL configuration, again it’s driven by an environment variable, this time called OPENSSL_CONF. We are going to copy the global OpenSSL configuration (often found in etc/ssl/openssl.conf) and customize it to use engines_pkcs11.

```bash
cp /etc/ssl/openssl.cnf /opt/bind9/etc/openssl.cnf
```

and export the environment variable:

```bash
export OPENSSL_CONF=/opt/bind9/etc/openssl.cnf
```

Now add the following line at the top of file, before any sections (in square brackets) are defined:

```bash
openssl_conf = openssl1_init
```

And make sure there are no other `openssl_conf = ...` lines in the file.

Add following lines at the bottom of the file:

```bash
[openssl1_init]
engines=engine_section

[engine_section]
pkcs11 = pkcs11_section

[pkcs11_section]
engine_id = pkcs11
dynamic_path = <PATHTO>/pkcs11.so
MODULE_PATH = <FULL_PATH_TO_HSM_MODULE>
# if automatic logging to the token is needed, PIN can be specified as below
#PIN = 1234
init = 0
```
5.5.5 Enabling the OpenSSL Engine in BIND commands

When using OpenSSL Engine-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. This engine name matches the 'engine_id' in the openssl.cnf created in previous section.

The zone signing commences as usual, with only one small difference. We need to provide the name of the OpenSSL engine using the -E command line option.

```
dnssec-signzone -E pkcs11 -S -o example.net example.net
```

5.5.6 OpenSSL 3 with pkcs11-provider

OpenSSL provider-based PKCS#11 uses pkcs11-provider project.

pkcs11-provider tries to fit the PKCS#11 API within the Provider API of OpenSSL. That is, it provides a gateway between PKCS#11 modules and the OpenSSL Provider API. One has to register the engine with OpenSSL and one has to provide the path to the PKCS#11 module which should be gatewayed to. This can be done by editing the OpenSSL configuration file, by engine specific controls, or by using the p11-kit proxy module.

It is required to use pkcs11-provider git commit 2e8c26b4157fd21422c66f0b4d7b26cf8c320570 from October 2, 2023 or later.

BIND support for pkcs11-provider is built in and the -E command line option explained above should not be used.

5.5.7 Configuring pkcs11-provider

The canonical documentation for configuring pkcs11-provider is in the provider-pkcs11.7 manual page, but here’s copy of working configuration for your convenience:

We are going to use our own custom copy of OpenSSL configuration, again it’s driven by an environment variable, this time called OPENSSL_CONF. We are going to copy the global OpenSSL configuration (often found in etc/ssl/openssl.conf) and customize it to use pkcs11-provider.

```
cp /etc/ssl/openssl.cnf /opt/bind9/etc/openssl.cnf
```

and export the environment variable:

```
export OPENSSL_CONF=/opt/bind9/etc/openssl.cnf
```

Now add the following line at the top of file, before any sections (in square brackets) are defined:

```
oppenssl_conf = openssl_init
```

And make sure there are no other ‘openssl_conf = …’ lines in the file.

Add following lines at the bottom of the file:

```
[openssl_init]
providers = provider_init

[provider_init]
default = default_init
pkcs11 = pkcs11_init

[default_init]  
```

(continues on next page)
activate = 1

[pkcs11_init]
module = <PATH_TO>/pkcs11.so
pkcs11-module-path = <FULL_PATH_TO_HSM_MODULE>
# bind uses the digest+sign api. this is broken with the default load behaviour,
# but works with early load. see: https://github.com/latchset/pkcs11-provider/issues/
->266
pkcs11-module-load-behavior = early
# no-deinit quirk is needed if you use softhsm2
#pkcs11-module-quirks = no-deinit
# if automatic logging to the token is needed, PIN can be specified as below
# the file referenced should contain just the PIN
#pkcs11-module-token-pin = file:/etc/pki/pin.txt
activate = 1

5.5.8 Key Generation

HSM keys can now be created and used. We are going to assume that you already have a BIND 9 installed, either from a package, or from the sources, and the tools are readily available in the $PATH.

For generating the keys, we are going to use pkcs11-tool available from the OpenSC suite. On both DEB-based and RPM-based distributions, the package is called opensc.

We need to generate at least two RSA keys:

```
pkcs11-tool --module <FULL_PATH_TO_HSM_MODULE> -l -k --key-type rsa:2048 --label...
--example.net-ksk --pin <PIN>
pkcs11-tool --module <FULL_PATH_TO_HSM_MODULE> -l -k --key-type rsa:2048 --label...
--example.net-zsk --pin <PIN>
```

Remember that each key should have unique label and we are going to use that label to reference the private key.

Convert the RSA keys stored in the HSM into a format that BIND 9 understands. The `dnssec-keyfromlabel` tool from BIND 9 can link the raw keys stored in the HSM with the \( \text{K<zone>+<alg>+<id>} \) files.

You'll need to provide the OpenSSL engine name (pkcs11) if using the engine and the algorithm (RSASHA256). The key is referenced with the PKCS#11 URI scheme and it can contain the PKCS#11 token label (we assume that it has been initialized as bind9), and the PKCS#11 object label (called label when generating the keys using `pkcs11-tool`) and the HSM PIN. Refer to RFC 7512 for the full PKCS#11 URI specification.

Convert the KSK:

```
dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l "pkcs11:token=bind9;object=example.net-\n\nksk;pin-value=0000" -f KSK example.net
```

and ZSK:

```
dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l "pkcs11:token=bind9;object=example.net-\n\nzsk;pin-value=0000" example.net
```

NOTE: you can use PIN stored on disk, by specifying `pin-source=<path_to>/<file>`, f.e.:

```
(umask 0700 && echo \n0000 > /opt/bind9/etc/pin.txt)
```

and then use in the label specification:
pin-source=/opt/bind9/etc/pin.txt

Confirm that you have one KSK and one ZSK present in the current directory:

```
ls -l K*
```

The output should look like this (the second number will be different):

```
Rexample.net.+008+31729.key
Rexexample.net.+008+31729.private
Rexexample.net.+008+42231.key
Rexexample.net.+008+42231.private
```

A note on generating ECDSA keys: there is a bug in libp11 when looking up a key, that function compares keys only on their ID, not the label. So when looking up a key it returns the first key, rather than the matching key. The workaround for this is when creating ECDSA keys, you should specify a unique ID:

```
ksk=$(echo "example.net-ksk" | openssl sha1 -r | awk '{print $1}')
zsk=$(echo "example.net-zsk" | openssl sha1 -r | awk '{print $1}')
```

```
pkcs11-tool --module <FULL_PATH_TO_HSM_MODULE> -l -k --key-type EC:prime256v1 --id $ksk --label example.net-ksk --pin <PIN>
pkcs11-tool --module <FULL_PATH_TO_HSM_MODULE> -l -k --key-type EC:prime256v1 --id $zsk --label example.net-zsk --pin <PIN>
```

### 5.5.9 Running named With Automatic Zone Re-signing

The zone can also be signed automatically by named. Again, we need to provide the name of the OpenSSL engine using the `-E` command line option, if using OpenSSL 1.x.x with engine_pkcs11, and this is not needed when using OpenSSL 3.x.x providers.

```
named -E pkcs11 -c named.conf
```

and the logs should have lines like:

```
Fetching example.net/RSASHA256/31729 (SKS) from key repository.
DNSKEY example.net/RSASHA256/31729 (SKS) is now published
DNSKEY example.net/RSASHA256/31729 (SKS) is now active
Fetching example.net/RSASHA256/42231 (ZSK) from key repository.
DNSKEY example.net/RSASHA256/42231 (ZSK) is now published
DNSKEY example.net/RSASHA256/42231 (ZSK) is now active
```

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`).

See OpenSSL extension specific documentation on how to configure the PIN on global level. Doing so 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.)

### 5.5. PKCS#11 (Cryptoki) Support
6.1 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.

6.1.1 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**.

### 6.2 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 message 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` and `namedconf:ref`also-notify` statements. 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.

### 6.3 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.

### 6.4 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.
6.4.1 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 Configurations and Zone Files.
Internal DNS server config:

```plaintext
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; }
}
```

(continues on next page)
allow-transfer { internals; }
};

External (bastion host) DNS server configuration:

```plaintext
cacl internals { 172.16.72.0/24; 192.168.1.0/24; }

cacl 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";
  primaries { another_bastion_host_maybe; }
  allow-transfer { internals; externals; }
};

In the resolv.conf (or equivalent) on the bastion host(s):

```plaintext
search ...
nameserver 172.16.72.2
nameserver 172.16.72.3
nameserver 172.16.72.4
```

### 6.5 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](https://tools.ietf.org/html/rfc3363) 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 \texttt{ip6.arpa} domain, as well as the older, deprecated \texttt{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.

### 6.5.1 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:

\begin{verbatim}
$ORIGIN example.com.
host 3600 IN AAAA 2001:db8::1
\end{verbatim}

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.

### 6.5.2 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 \texttt{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:

\begin{verbatim}
$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)
\end{verbatim}

### 6.6 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 modules exist for several different database backends, including MySQL and LDAP, and can be written for any other.

The DLZ module provides data to \texttt{named} in text format, which is then converted to DNS wire format by \texttt{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.

### 6.6.1 Configuring DLZ

\texttt{dlz}

\textbf{Grammar zone (primary, redirect, secondary):} \texttt{dlz <string>;}

\textbf{Grammar topmost, view:}

\begin{verbatim}
dlz <string> { 
   database <string>;
   search <boolean>;
}; // may occur multiple times
\end{verbatim}
Blocks: topmost, view, zone (primary, redirect, secondary)
Tags: zone

Configures a Dynamically Loadable Zone (DLZ) database in named.conf.

A DLZ database is configured with a `dlz` statement in named.conf:

```plaintext
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.

**search**

Grammar: `search <boolean>`;

Blocks: dlz, view.dlz

Tags: query

Specifies whether a Dynamically Loadable Zone (DLZ) module is queried for an answer to a query name.

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:

```plaintext
dlz other {
    database "dlopen driver.so args";
    search no;
};
zone "." {
    type redirect;
    dlz other;
};
```

### 6.6.2 Sample DLZ Module

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:

```plaintext
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.

### 6.7 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`.

#### 6.7.1 Configuring DynDB

```
dyndb
  Grammar: dyndb <string> <quoted_string> { <unspecified-text> } // may occur multiple times
  Blocks: topmost, view
  Tags: zone

  Configures a DynDB database in `named.conf`.
```

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.

### 6.7.2 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.

### 6.8 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 [RFC 9432](https://tools.ietf.org/html/rfc9432).

#### 6.8.1 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 primaries 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.

### 6.8.2 Configuring Catalog Zones

catalog-zones

Grammar:
catalog-zones { zone <string> [ default-primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... } ] [ zone-directory <quoted_string> ] [ in-memory <boolean> ] [ min-update-interval <duration> ]; ... };

Blocks: options, view

Tags: zone

Configures catalog zones in named.conf.

Catalog zones are configured with a catalog-zones statement in the options or view section of named.conf. For example:

```plaintext
catalog-zones {
  zone "catalog.example"
    default-primaries { 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**

Synonym for `default-primaries`.

**default-primaries**

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.

### 6.8.3 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.

```plaintext
IN SOA . . 2016022901 900 600 86400 1
IN NS invalid.
IN TXT "2"
```

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. Currently BIND supports catalog zone schema versions “1” and “2”.

Also note that the catalog zone must have an NS record in order to be a valid DNS zone, and using the value “invalid” for NS is recommended.

A member zone is added by including a PTR resource record in the `zones` sub-domain of the catalog zone. The record label can be any unique label. The target of the PTR record is the member zone name. For example, to add member zones `domain.example` and `domain2.example`:

```plaintext
IN PTR domain.example. 8960775ba38e7a4e9263fc06e7c00569b6a05c.zones.catalog.example.
IN PTR domain2.example. uniquelabel.zones.catalog.example.
```

The label is necessary to identify custom properties (see below) for a specific member zone. Also, the zone state can be reset by changing its label, in which case BIND will remove the member zone and add it back.
6.8.4 Catalog Zone Custom Properties

BIND uses catalog zones custom properties to define different properties which can be set either globally for the whole catalog zone or for a single member zone. Global custom properties override the settings in the configuration file, and member zone custom properties override global custom properties.

For the version “1” of the schema custom properties must be placed without a special suffix.

For the version “2” of the schema custom properties must be placed under the “.ext” suffix.

Global custom properties are set at the apex of the catalog zone, e.g.:

```
primaries.ext.catalog.example. IN AAAA 2001::db8::1
```

BIND currently supports the following custom properties:

- **A simple primaries definition:**

  ```
  primaries.ext.catalog.example. IN A 192.0.2.1
  ```

  This custom property 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.

  Note: masters can be used as a synonym for primaries.

- **A primaries with a TSIG key defined:**

  ```
  label.primaries.ext.catalog.example. IN A 192.0.2.2
  label.primaries.ext.catalog.example. IN TXT "tsig_key_name"
  ```

  This custom property 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.

  Note: masters can be used as a synonym for primaries.

- **allow-query and allow-transfer ACLs:**

  ```
  allow-query.ext.catalog.example. IN APL 1:10.0.0.0/24
  allow-transfer.ext.catalog.example. IN APL !1:10.0.0.0/32 1:10.0.0.0/24
  ```

  These custom properties are the equivalents of allow-query and allow-transfer options 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.

The member zone-specific custom properties are defined the same way as global custom properties, but in the member zone subdomain:

```
primaries.ext.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN A 192.0.2.2
label.primaries.ext.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN AAAA 2001::db8::2
label.primaries.ext.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN TXT "tsig_key_name"
allow-query.ext.5960775ba382e7a4e09263fc06e7c00569b6a05c.zones.catalog.example. IN APL 1:10.0.0.0/24
primaries.ext.uniquelabel.zones.catalog.example. IN A 192.0.2.3
```

Custom properties defined for a specific zone override the global custom properties defined in the catalog zone. These in turn override the global options defined in the catalog-zones statement in the configuration file.
6.8.5 Change of Ownership (coo)

BIND supports the catalog zones “Change of Ownership” (coo) property. When the same entry which exists in one catalog zone is added into another catalog zone, the default behavior for BIND is to ignore it, and continue serving the zone using the catalog zone where it was originally existed, unless it is removed from there, then it can be added into the new one.

Using the coo property it is possible to gracefully move a zone from one catalog zone into another, by letting the catalog consumers know that it is permitted to do so. To do that, the original catalog zone should be updated with a new record with coo custom property:

```
uniquelabel.zones.catalog.example. IN PTR domain2.example.
coo.uniquelabel.zones.catalog.example. IN PTR catalog2.example.
```

Here, the catalog.example catalog zone gives permission for the member zone with label “uniquelabel” to be transferred into catalog2.example catalog zone. Catalog consumers which support the coo property will then take note, and when the zone is finally added into catalog2.example catalog zone, catalog consumers will change the ownership of the zone from catalog.example to catalog2.example. BIND's implementation simply deletes the zone from the old catalog zone and adds it back into the new catalog zone, which also means that all associated state for the just migrated zone will be reset, including when the unique label is the same.

The record with coo custom property can be later deleted by the catalog zone operator after confirming that all the consumers have received it and have successfully changed the ownership of the zone.

6.9 DNS Firewalls and Response Policy Zones

A DNS firewall examines DNS traffic and allows some responses to pass through while blocking others. This examination can be based on several criteria, including the name requested, the data (such as an IP address) associated with that name, or the name or IP address of the name server that is authoritative for the requested name. Based on these criteria, a DNS firewall can be configured to discard, modify, or replace the original response, allowing administrators more control over what systems can access or be accessed from their networks.

DNS Response Policy Zones (RPZ) are a form of DNS firewall in which the firewall rules are expressed within the DNS itself - encoded in an open, vendor-neutral format as records in specially constructed DNS zones.

Using DNS zones to configure policy allows policy to be shared from one server to another using the standard DNS zone transfer mechanism. This allows a DNS operator to maintain their own firewall policies and share them easily amongst their internal name servers, or to subscribe to external firewall policies such as commercial or cooperative “threat feeds,” or both.

`named` can subscribe to up to 64 Response Policy Zones, each of which encodes a separate policy rule set. Each rule is stored in a DNS resource record set (RRset) within the RPZ, and consists of a trigger and an action. There are five types of triggers and six types of actions.

A response policy rule in a DNS RPZ can be triggered as follows:

- by the IP address of the client
- by the query name
- by an address which would be present in a truthful response
- by the name or address of an authoritative name server responsible for publishing the original response
A response policy action can be one of the following:

- to synthesize a “domain does not exist” (NXDOMAIN) response
- to synthesize a “name exists but there are no records of the requested type” (NODATA) response
- to drop the response
- to switch to TCP by sending a truncated UDP response that requires the DNS client to try again with TCP
- to replace/override the response's data with specific data (provided within the response policy zone)
- to exempt the response from further policy processing

The most common use of a DNS firewall is to “poison” a domain name, IP address, name server name, or name server IP address. Poisoning is usually done by forcing a synthetic “domain does not exist” (NXDOMAIN) response. This means that if an administrator maintains a list of known “phishing” domains, these names can be made unreachable by customers or end users just by adding a firewall policy into the recursive DNS server, with a trigger for each known “phishing” domain, and an action in every case forcing a synthetic NXDOMAIN response. It is also possible to use a data-replacement action such as answering for these known “phishing” domains with the name of a local web server that can display a warning page. Such a web server would be called a “walled garden.”

Note: Authoritative name servers can be responsible for many different domains. If DNS RPZ is used to poison all domains served by some authoritative name server name or address, the effects can be quite far-reaching. Users are advised to ensure that such authoritative name servers do not also serve domains that should not be poisoned.

### 6.9.1 Why Use a DNS Firewall?

Criminal and network abuse traffic on the Internet often uses the Domain Name System (DNS), so protection against these threats should include DNS firewalling. A DNS firewall can selectively intercept DNS queries for known network assets including domain names, IP addresses, and name servers. Interception can mean rewriting a DNS response to direct a web browser to a “walled garden,” or simply making any malicious network assets invisible and unreachable.

### 6.9.2 What Can a DNS Firewall Do?

Firewalls work by applying a set of rules to a traffic flow, where each rule consists of a trigger and an action. Triggers determine which messages within the traffic flow are handled specially, and actions determine what that special handling is. For a DNS firewall, the traffic flow to be controlled consists of responses sent by a recursive DNS server to its end-user clients. Some true responses are not safe for all clients, so the policy rules in a DNS firewall allow some responses to be intercepted and replaced with safer content.

### 6.9.3 Creating and Maintaining RPZ Rule Sets

In DNS RPZ, the DNS firewall policy rule set is stored in a DNS zone, which is maintained and synchronized using the same tools and methods as for any other DNS zone. The primary name server for a DNS RPZ may be an internal server, if an administrator is creating and maintaining their own DNS policy zone, or it may be an external name server (such as a security vendor's server), if importing a policy zone published externally. The primary copy of the DNS firewall policy can be a DNS “zone file” which is either edited by hand or generated from a database. A DNS zone can also be edited indirectly using DNS dynamic updates (for example, using the “nsupdate” shell level utility).

DNS RPZ allows firewall rules to be expressed in a DNS zone format and then carried to subscribers as DNS data. A recursive DNS server which is capable of processing DNS RPZ synchronizes these DNS firewall rules using the same standard DNS tools and protocols used for secondary name service. The DNS policy information is then promoted to the DNS control plane inside the customer’s DNS resolver, making that server into a DNS firewall.
A security company whose products include threat intelligence feeds can use a DNS Response Policy Zone (RPZ) as a delivery channel to customers. Threats can be expressed as known-malicious IP addresses and subnets, known-malicious domain names, and known-malicious domain name servers. By feeding this threat information directly into customers' local DNS resolvers, providers can transform these DNS servers into a distributed DNS firewall.

When a customer's DNS resolver is connected by a real-time subscription to a threat intelligence feed, the provider can protect the customer's end users from malicious network elements (including IP addresses and subnets, domain names, and name servers) immediately as they are discovered. While it may take days or weeks to “take down” criminal and abusive infrastructure once reported, a distributed DNS firewall can respond instantly.

Other distributed TCP/IP firewalls have been on the market for many years, and enterprise users are now comfortable importing real-time threat intelligence from their security vendors directly into their firewalls. This intelligence can take the form of known-malicious IP addresses or subnets, or of patterns which identify known-malicious email attachments, file downloads, or web addresses (URLs). In some products it is also possible to block DNS packets based on the names or addresses they carry.

### 6.9.4 Limitations of DNS RPZ

We're often asked if DNS RPZ could be used to set up redirection to a CDN. For example, if “mydomain.com” is a normal domain with SOA, NS, MX, TXT records etc., then if someone sends an A or AAAA query for “mydomain.com”, can we use DNS RPZ on an authoritative nameserver to return “CNAME mydomain.com.my-cdn-provider.net”?

The problem with this suggestion is that there is no way to CNAME only A and AAAA queries, not even with RPZ.

The underlying reason is that if the authoritative server answers with a CNAME, the recursive server making that query will cache the response. Thereafter (while the CNAME is still in cache), it assumes that there are no records of any non-CNAME type for the name that was being queried, and directs subsequent queries for all other types directly to the target name of the CNAME record.

To be clear, this is not a limitation of RPZ; it is a function of the way the DNS protocol works. It’s simply not possible to use “partial” CNAMEs to help when setting up CDNs because doing this will break other functionality such as email routing.

Similarly, following the DNS protocol definition, wildcards in the form of *.example records might behave in unintuitive ways. For a detailed definition of wildcards in DNS, please see [RFC 4592](https://tools.ietf.org/html/rfc4592), especially section 2.

### 6.9.5 DNS Firewall Usage Examples

Here are some scenarios in which a DNS firewall might be useful.

Some known threats are based on an IP address or subnet (IP address range). For example, an analysis may show that all addresses in a “class C” network are used by a criminal gang for “phishing” web servers. With a DNS firewall based on DNS RPZ, a firewall policy can be created such as “if a DNS lookup would result in an address from this class C network, then answer instead with an NXDOMAIN indication.” That simple rule would prevent any end users inside customers' networks from being able to look up any domain name used in this phishing attack – without having to know in advance what those names might be.

Other known threats are based on domain names. An analysis may determine that a certain domain name or set of domain names is being or will shortly be used for spamming, phishing, or other Internet-based attacks which all require working domain names. By adding name-triggered rules to a distributed DNS firewall, providers can protect customers' end users from any attacks which require them to be able to look up any of these malicious names. The names can be wildcards (for example, *.evil.com), and these wildcards can have exceptions if some domains are not as malicious as others (if *.evil.com is bad, then not.evil.com might be an exception).

Alongside growth in electronic crime has come growth of electronic criminal expertise. Many criminal gangs now maintain their own extensive DNS infrastructure to support a large number of domain names and a diverse set of IP addressing resources. Analyses show in many cases that the only truly fixed assets criminal organizations have are their name servers,
which are by nature slightly less mobile than other network assets. In such cases, DNS administrators can anchor their DNS firewall policies in the abusive name server names or name server addresses, and thus protect their customers' end users from threats where neither the domain name nor the IP address of that threat is known in advance.

Electronic criminals rely on the full resiliency of DNS just as the rest of digital society does. By targeting criminal assets at the DNS level we can deny these criminals the resilience they need. A distributed DNS firewall can leverage the high skills of a security company to protect a large number of end users. DNS RPZ, as the first open and vendor-neutral distributed DNS firewall, can be an effective way to deliver threat intelligence to customers.

A Real-World Example of DNS RPZ's Value

The Conficker malware worm (https://en.wikipedia.org/wiki/Conficker) was first detected in 2008. Although it is no longer an active threat, the techniques described here can be applied to other DNS threats.

Conficker used a domain generation algorithm (DGA) to choose up to 50,000 command and control domains per day. It would be impractical to create an RPZ that contains so many domain names and changes so much on a daily basis. Instead, we can trigger RPZ rules based on the names of the name servers that are authoritative for the command and control domains, rather than trying to trigger on each of 50,000 different (daily) query names. Since the well-known name server names for Conficker's domain names are never used by nonmalicious domains, it is safe to poison all lookups that rely on these name servers. Here is an example that achieves this result:

```
$ORIGIN rpz.example.com.
ns.0xc0f1c3a5.com.rpz-nsdname CNAME *.walled-garden.example.com.
nss.0xc0f1c3a5.net.rpz-nsdname CNAME *.walled-garden.example.com.
nss.0xc0f1c3a5.org.rpz-nsdname CNAME *.walled-garden.example.com.
```

The * at the beginning of these CNAME target names is special, and it causes the original query name to be prepended to the CNAME target. So if a user tries to visit the Conficker command and control domain racaldftn.com.ai (which was a valid Conficker command and control domain name on 19-October-2011), the RPZ-configured recursive name server will send back this answer:

```
```

This example presumes that the following DNS content has also been created, which is not part of the RPZ zone itself but is in another domain:

```
$ORIGIN walled-garden.example.com.
* A 192.168.50.3
```

Assuming that we're running a web server listening on 192.168.50.3 that always displays a warning message no matter what uniform resource identifier (URI) is used, the above RPZ configuration will instruct the web browser of any infected end user to connect to a “server name” consisting of their original lookup name (racaldftn.com.ai) prepended to the walled garden domain name (walled-garden.example.com). This is the name that will appear in the web server's log file, and having the full name in that log file will facilitate an analysis as to which users are infected with what virus.
6.9.6 Keeping Firewall Policies Updated

It is vital for overall system performance that incremental zone transfers (see RFC 1995) and real-time change notification (see RFC 1996) be used to synchronize DNS firewall rule sets between the publisher’s primary copy of the rule set and the subscribers’ working copies of the rule set.

If DNS dynamic updates are used to maintain a DNS RPZ rule set, the name server automatically calculates a stream of deltas for use when sending incremental zone transfers to the subscribing name servers. Sending a stream of deltas – known as an “incremental zone transfer” or IXFR – is usually much faster than sending the full zone every time it changes, so it’s worth the effort to use an editing method that makes such incremental transfers possible.

Administrators who edit or periodically regenerate a DNS RPZ rule set and whose primary name server uses BIND can enable the `ixfr-from-differences` option, which tells the primary name server to calculate the differences between each new zone and the preceding version, and to make these differences available as a stream of deltas for use in incremental zone transfers to the subscribing name servers. This will look something like the following:

```
options {
    // ...
    ixfr-from-differences yes;
    // ...
};
```

As mentioned above, the simplest and most common use of a DNS firewall is to poison domain names known to be purely malicious, by simply making them disappear. All DNS RPZ rules are expressed as resource record sets (RRsets), and the way to express a “force a name-does-not-exist condition” is by adding a CNAME pointing to the root domain (.). In practice this looks like:

```
$ORIGIN rpz.example.com.
malicious1.org CNAME .
*.malicious1.org CNAME .
malicious2.org CNAME .
*.malicious2.org CNAME .
```

Two things are noteworthy in this example. First, the malicious names are made relative within the response policy zone. Since there is no trailing dot following “.org” in the above example, the actual RRsets created within this response policy zone are, after expansion:

```
malicious1.org.rpz.example.com. CNAME .
malicious2.org.rpz.example.com. CNAME .
```

Second, for each name being poisoned, a wildcard name is also listed. This is because a malicious domain name probably has or may potentially have malicious subdomains.

In the above example, the relative domain names `malicious1.org` and `malicious2.org` will match only the real domain names `malicious1.org` and `malicious2.org`, respectively. The relative domain names `*.malicious1.org` and `*.malicious2.org` will match any `subdomain.of.malicious1.org` or `subdomain.of.malicious2.org`, respectively.

This example forces an NXDOMAIN condition as its policy action, but other policy actions are also possible.
6.9.7 Performance and Scalability When Using Multiple RPZs

Since version 9.10, BIND can be configured to have different response policies depending on the identity of the querying client and the nature of the query. To configure BIND response policy, the information is placed into a zone file whose only purpose is conveying the policy information to BIND. A zone file containing response policy information is called a Response Policy Zone, or RPZ, and the mechanism in BIND that uses the information in those zones is called DNS RPZ.

It is possible to use as many as 64 separate RPZ files in a single instance of BIND, and BIND is not significantly slowed by such heavy use of RPZ.

(Note: by default, BIND 9.11 only supports up to 32 RPZ files, but this can be increased to 64 at compile time. All other supported versions of BIND support 64 by default.)

Each one of the policy zone files can specify policy for as many different domains as necessary. The limit of 64 is on the number of independently-specified policy collections and not the number of zones for which they specify policy.

Policy information from all of the policy zones together are stored in a special data structure allowing simultaneous lookups across all policy zones to be performed very rapidly. Looking up a policy rule is proportional to the logarithm of the number of rules in the largest single policy zone.

6.9.8 Practical Tips for DNS Firewalls and DNS RPZ

Administrators who subscribe to an externally published DNS policy zone and who have a large number of internal recursive name servers should create an internal name server called a “distribution master” (DM). The DM is a secondary (stealth secondary) name server from the publisher’s point of view; that is, the DM is fetching zone content from the external server. The DM is also a primary name server from the internal recursive name servers’ point of view: they fetch zone content from the DM. In this configuration the DM is acting as a gateway between the external publisher and the internal subscribers.

The primary server must know the unicast listener address of every subscribing recursive server, and must enumerate all of these addresses as destinations for real time zone change notification (as described in RFC 1996). So if an enterprise-wide RPZ is called “rpz.example.com” and if the unicast listener addresses of four of the subscribing recursive name servers are 192.0.200.1, 192.0.201.1, 192.0.202.1, and 192.0.203.1, the primary server’s configuration looks like this:

```plaintext
zone "rpz.example.com" {
    type primary;
    file "primary/rpz.example.com";
    notify explicit;
    also-notify { 192.0.200.1;
                   192.0.201.1;
                   192.0.202.1;
                   192.0.203.1; }
    allow-transfer { 192.0.200.1;
                     192.0.201.1;
                     192.0.202.1;
                     192.0.203.1; }
    allow-query { localhost; }
};
```

Each recursive DNS server that subscribes to the policy zone must be configured as a secondary server for the zone, and must also be configured to use the policy zone for local response policy. To subscribe a recursive name server to a response policy zone where the unicast listener address of the primary server is 192.0.220.2, the server’s configuration should look like this:

```plaintext
options {
    // ...
}
```
Note that queries are restricted to “localhost,” since query access is never used by DNS RPZ itself, but may be useful to DNS operators for use in debugging. Transfers should be disallowed to prevent policy information leaks.

If an organization's business continuity depends on full connectivity with another company whose ISP also serves some criminal or abusive customers, it’s possible that one or more external RPZ providers — that is, security feed vendors — may eventually add some RPZ rules that could hurt a company’s connectivity to its business partner. Users can protect themselves from this risk by using an internal RPZ in addition to any external RPZs, and by putting into their internal RPZ some “pass-through” rules to prevent any policy action from affecting a DNS response that involves a business partner.

A recursive DNS server can be connected to more than one RPZ, and these are searched in order. Therefore, to protect a network from dangerous policies which may someday appear in external RPZ zones, administrators should list the internal RPZ zones first.

Within an internal RPZ, there need to be rules describing the network assets of business partners whose communications need to be protected. Although it is not generally possible to know what domain names they use, administrators will be aware of what address space they have and perhaps what name server names they use.

Here, we know that answers in address block 10.0.0.0/8 indicate a business partner, as well as answers involving any name server whose address is in the 128.45.0.0/16 address block, and answers involving the name servers whose names are ns.partner1.com or ns.partner2.com.

The above example demonstrates that when matching by answer IP address (the .rpz-ip owner), or by name server IP address (the .rpz-nsip owner) or by name server domain name (the .rpz-nsdname owner), the special RPZ marker (.rpz-ip, .rpz-nsip, or .rpz-nsdname) does not appear as part of the CNAME target name.

By triggering these rules using the known network assets of a partner, and using the “pass-through” policy action, no later RPZ processing (which in the above example refers to the “rpz.security-vendor-1.com” and “rpz.security-vendor-2.com”
policy zones) will have any effect on DNS responses for partner assets.

### 6.9.9 Creating a Simple Walled Garden Triggered by IP Address

It may be the case that the only thing known about an attacker is the IP address block they are using for their “phishing” web servers. If the domain names and name servers they use are unknown, but it is known that every one of their “phishing” web servers is within a small block of IP addresses, a response can be triggered on all answers that would include records in this address range, using RPZ rules that look like the following example:

```
$ORIGIN rpz.example.com.
22.0.212.94.109.rpz-ip CNAME drop.garden.example.com.
*.212.94.109.in-addr.arpa CNAME .
*.213.94.109.in-addr.arpa CNAME .
*.214.94.109.in-addr.arpa CNAME .
*.215.94.109.in-addr.arpa CNAME .
```

Here, if a truthful answer would include an A (address) RR (resource record) whose value were within the 109.94.212.0/22 address block, then a synthetic answer is sent instead of the truthful answer. Assuming the query is for www.malicious.net, the synthetic answer is:

```
www.malicious.net. CNAME drop.garden.example.com.
drop.garden.example.com. A 192.168.7.89
```

This assumes that `drop.garden.example.com` has been created as real DNS content, outside of the RPZ:

```
$ORIGIN example.com.
drop.garden A 192.168.7.89
```

In this example, there is no “*” in the CNAME target name, so the original query name will not be present in the walled garden web server’s log file. This is an undesirable loss of information, and is shown here for example purposes only.

The above example RPZ rules would also affect address-to-name (also known as “reverse DNS”) lookups for the unwanted addresses. If a mail or web server receives a connection from an address in the example’s 109.94.212.0/22 address block, it will perform a PTR record lookup to find the domain name associated with that IP address.

This kind of address-to-name translation is usually used for diagnostic or logging purposes, but it is also common for email servers to reject any email from IP addresses which have no address-to-name translation. Most mail from such IP addresses is spam, so the lack of a PTR record here has some predictive value. By using the “force name-does-not-exist” policy trigger on all lookups in the PTR name space associated with an address block, DNS administrators can give their servers a hint that these IP addresses are probably sending junk.

### 6.9.10 A Known Inconsistency in DNS RPZ’s NSDNAME and NSIP Rules

Response Policy Zones define several possible triggers for each rule, and among these two are known to produce inconsistent results. This is not a bug; rather, it relates to inconsistencies in the DNS delegation model.
DNS Delegation

In DNS authority data, an NS RRset that is not at the apex of a DNS zone creates a sub-zone. That sub-zone’s data is separate from the current (or “parent”) zone, and it can have different authoritative name servers than the current zone. In this way, the root zone leads to COM, NET, ORG, and so on, each of which have their own name servers and their own way of managing their authoritative data. Similarly, ORG has delegations to ISC.ORG and to millions of other “dot-ORG” zones, each of which can have its own set of authoritative name servers. In the parlance of the protocol, these NS RRsets below the apex of a zone are called “delegation points.” An NS RRset at a delegation point contains a list of authoritative servers to which the parent zone is delegating authority for all names at or below the delegation point.

At the apex of every zone there is also an NS RRset. Ideally, this so-called “apex NS RRset” should be identical to the “delegation point NS RRset” in the parent zone, but this ideal is not always achieved. In the real DNS, it’s almost always easier for a zone administrator to update one of these NS RRsets than the other, so that one will be correct and the other out of date. This inconsistency is so common that it’s been necessarily rendered harmless: domains that are inconsistent in this way are less reliable and perhaps slower, but they still function as long as there is some overlap between each of the NS RRsets and the truth. (“Truth” in this case refers to the actual set of name servers that are authoritative for the zone.)

A Quick Review of DNS Iteration

In DNS recursive name servers, an incoming query that cannot be answered from the local cache is sent to the closest known delegation point for the query name. For example, if a server is looking up XYZZY.ISC.ORG and it the name servers for ISC.ORG, then it sends the query to those servers directly; however, if it has never heard of ISC.ORG before, it must first send the query to the name servers for ORG (or perhaps even to the root zone that is the parent of ORG).

When it asks one of the parent name servers, that server will not have an answer, so it sends a “referral” consisting only of the “delegation point NS RRset.” Once the server receives this referral, it “iterates” by sending the same query again, but this time to name servers for a more specific part of the query name. Eventually this iteration terminates, usually by getting an answer or a “name error” (NXDOMAIN) from the query name’s authoritative server, or by encountering some type of server failure.

When an authoritative server for the query name sends an answer, it has the option of including a copy of the zone’s apex NS RRset. If this occurs, the recursive name server caches this NS RRset, replacing the delegation point NS RRset that it had received during the iteration process. In the parlance of the DNS, the delegation point NS RRset is “glue,” meaning non-authoritative data, or more of a hint than a real truth. On the other hand, the apex NS RRset is authoritative data, coming as it does from the zone itself, and it is considered more credible than the “glue.” For this reason, it’s a little bit more important that the apex NS RRset be correct than that the delegation point NS RRset be correct, since the former will quickly replace the latter, and will be used more often for a longer total period of time.

Importantly, the authoritative name server need not include its apex NS RRset in any answers, and recursive name servers do not ordinarily query directly for this RRset. Therefore it is possible for the apex NS RRset to be completely wrong without any operational ill-effects, since the wrong data need not be exposed. Of course, if a query comes in for this NS RRset, most recursive name servers will forward the query to the zone’s authority servers, since it’s bad form to return “glue” data when asked a specific question. In these corner cases, bad apex NS RRset data can cause a zone to become unreachable unpredictably, according to what other queries the recursive name server has processed.

There is another kind of “glue,” for name servers whose names are below delegation points. If ORG delegates ISC.ORG to NS-EXT.ISC.ORG, the ORG server needs to know an address for NS-EXT.ISC.ORG and return this address as part of the delegation response. However, the name-to-address binding for this name server is only authoritative inside the ISC.ORG zone; therefore, the A or AAAA RRset given out with the delegation is non-authoritative “glue,” which is replaced by an authoritative RRset if one is seen. As with apex NS RRsets, the real A or AAAA RRset is not automatically queried for by the recursive name server, but is queried for if an incoming query asks for this RRset.
Enter RPZ

RPZ has two trigger types that are intended to allow policy zone authors to target entire groups of domains based on those domains all being served by the same DNS servers: NSDNAME and NSIP. The NSDNAME and NSIP rules are matched against the name and IP address (respectively) of the nameservers of the zone the answer is in, and all of its parent zones. In its default configuration, BIND actively fetches any missing NS RRsets and address records. If, in the process of attempting to resolve the names of all of these delegated server names, BIND receives a SERVFAIL response for any of the queries, then it aborts the policy rule evaluation and returns SERVFAIL for the query. This is technically neither a match nor a non-match of the rule.

Every “.” in a fully qualified domain name (FQDN) represents a potential delegation point. When BIND goes searching for parent zone NS RRsets (and, in the case of NSIP, their accompanying address records), it has to check every possible delegation point. This can become a problem for some specialized pseudo-domains, such as some domain name and network reputation systems, that have many “.” characters in the names. It is further complicated if that system also has non-compliant DNS servers that silently drop queries for NS and SOA records. This forces BIND to wait for those queries to time out before it can finish evaluating the policy rule, even if this takes longer than a reasonable client typically waits for an answer (delays of over 60 seconds have been observed).

While both of these cases do involve configurations and/or servers that are technically “broken,” they may still “work” outside of RPZ NSIP and NSDNAME rules because of redundancy and iteration optimizations.

There are two RPZ options, nsip-wait-recurse and nsdname-wait-recurse, that alter BIND’s behavior by allowing it to use only those records that already exist in the cache when evaluating NSIP and NSDNAME rules, respectively.

Therefore NSDNAME and NSIP rules are unreliable. The rules may be matched against either the apex NS RRset or the “glue” NS RRset, each with their associated addresses (that also might or might not be “glue”). It’s in the administrator’s interests to discover both the delegation name server names and addresses, and the apex name server names and authoritative address records, to ensure correct use of NS and NSIP triggers in RPZ. Even then, there may be collateral damage to completely unrelated domains that otherwise “work,” just by having NSIP and NSDNAME rules.

6.9.11 Example: Using RPZ to Disable Mozilla DoH-by-Default

Mozilla announced in September 2019 that they would enable DNS-over-HTTPS (DoH) for all US-based users of the Firefox browser, sending all their DNS queries to predefined DoH providers (Cloudflare’s 1.1.1.1 service in particular). This is a concern for some network administrators who do not want their users’ DNS queries to be rerouted unexpectedly. However, Mozilla provides a mechanism to disable the DoH-by-default setting: if the Mozilla-owned domain use-application-dns.net returns an NXDOMAIN response code, Firefox will not use DoH.

To accomplish this using RPZ:

1. Create a policy zone file called mozilla.rpz.db configured so that NXDOMAIN will be returned for any query to use-application-dns.net:

   ```
   $TTL 604800
   $ORIGIN mozilla.rpz.
   @ IN SOA localhost. root.localhost. 1 604800 86400 2419200 604800
   @ IN NS localhost.
   use-application-dns.net CNAME .
   ```

2. Add the zone into the BIND configuration (usually named.conf):

   ```
   zone mozilla.rpz {
   type primary;
   file "/<PATH_TO>/mozilla.rpz.db";
   allow-query { localhost; };
   }
   ```
3. Enable use of the Response Policy Zone for all incoming queries by adding the `response-policy` directive into the `options {}` section:

```plaintext
options {
    response-policy { zone mozilla.rpz; } break-dnssec yes;
};
```

4. Reload the configuration and test whether the Response Policy Zone that was just added is in effect:

```plaintext
# rndc reload
# dig IN A use-application-dns.net @<IP_ADDRESS_OF_YOUR_RESOLVER>
# dig IN AAAA use-application-dns.net @<IP_ADDRESS_OF_YOUR_RESOLVER>
```

The response should return `NXDOMAIN` instead of the list of IP addresses, and the BIND 9 log should contain lines like this:

```
09-Sep-2019 18:50:49.439 client @0x7faf8e004a00 ::1#54175 (use-application-dns.net):-
    rpz QNAME NXDOMAIN rewrite use-application-dns.net/AAAA/IN via use-application-dns.
    net.mozilla.rpz
09-Sep-2019 18:50:49.439 client @0x7faf8e007800 127.0.0.1#62915 (use-application-dns.
    net): rpz QNAME NXDOMAIN rewrite use-application-dns.net/AAAA/IN via use-
    application-dns.net.mozilla.rpz
```

Note that this is the simplest possible configuration; specific configurations may be different, especially for administrators who are already using other response policy zones, or whose servers are configured with multiple views.
CHAPTER SEVEN

SECURITY CONFIGURATIONS

7.1 Security Assumptions

BIND 9’s design assumes that access to the objects listed below is limited only to trusted parties. An incorrect deployment, which does not follow rules set by this section, cannot be the basis for CVE assignment or special security-sensitive handling of issues.

Unauthorized access can potentially disclose sensitive data, slow down server operation, etc. Unauthorized, unexpected, or incorrect writes to listed objects can potently cause crashes, incorrect data handling, or corruption.

- All files stored on disk - including zone files, configuration files, key files, temporary files, etc.
- Clients communicating via controls socket using configured keys
- Access to statistics-channels from untrusted clients
- Sockets used for update-policy type external

Certain aspects of the DNS protocol are left unspecified, such as the handling of responses from DNS servers which do not fully conform to the DNS protocol. For such a situation, BIND implements its own safety checks and limits which are subject to change as the protocol and deployment evolve.

7.1.1 Authoritative Servers

By default, zones use intentionally lenient limits (unlimited size, long transfer timeouts, etc.). These defaults can be misused by the source of data (zonetransfers or UPDATES) to exhaust resources on the receiving side.

The impact of malicious zone changes can be limited, to an extent, using configuration options listed in sections Server Resource Limits and Zone Transfers. Limits should also be applied to zones where malicious clients may potentially be authorized to use Dynamic Update.

7.1.2 DNS Resolvers

By definition, DNS resolvers act as traffic amplifiers; during normal operation, a DNS resolver can legitimately generate more outgoing traffic (counted in packets or bytes) than the incoming client traffic that triggered it. The DNS protocol specification does not currently specify limits for this amplification, but BIND implements its own limits to balance interoperability and safety. As a general rule, if a traffic amplification factor for any given scenario is lower than 100 packets, ISC does not handle the given scenario as a security issue. These limits are subject to change as DNS deployment evolves.

All DNS answers received by the DNS resolver are treated as untrusted input and are subject to safety and correctness checks. However, protocol non-conformity might cause unexpected behavior. If such unexpected behavior is limited to DNS domains hosted on non-conformant servers, it is not deemed a security issue in BIND.
7.2 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:

```c
// Set up an ACL named "bogusnets" that blocks
// RFC1918 space and some reserved space, which is
// commonly used in spoofing attacks.
ac1 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.
ac1 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:

```plaintext
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:

```plaintext
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.

### 7.3 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:

```plaintext
/usr/local/sbin/named -u 202 -t /var/named
```
7.3.1 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.

7.3.2 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 userid 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.

7.4 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.

7.5 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.

### 7.5.1 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.

### 7.5.2 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 "DAopyf1mhCfVZw7pgmNPBoLUq8wEUT7UoPoLENP2HY=";
};
```

(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.

### 7.5.3 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`:
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.

### 7.5.4 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.

### 7.5.5 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).

### 7.6 SIG(0)

BIND partially supports DNSSEC SIG(0) transaction signatures as specified in [RFC 2535](http://example.com/rfc2535) and [RFC 2931](http://example.com/rfc2931). 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`. 
The operational functionality of BIND 9 is defined using the file `named.conf`, which is typically located in `/etc` or `/usr/local/etc/namedb`, depending on the operating system or distribution. A further file `rndc.conf` will be present if `rndc` is being run from a remote host, but is not required if `rndc` is being run from `localhost` (the same system as BIND 9 is running on).

8.1 Configuration File (named.conf)

The file `named.conf` may contain three types of entities:

Comment

Multiple comment formats are supported.

Block

*Blocks* are containers for *statements* which either have common functionality - for example, the definition of a cryptographic key in a *key* block - or which define the scope of the statement - for example, a statement which appears in a *zone* block has scope only for that zone.

Blocks are organized hierarchically within `named.conf` and may have a number of different properties:

- Certain blocks cannot be nested inside other blocks and thus may be regarded as the *topmost-level* blocks: for example, the *options* block and the *logging* block.
- Certain blocks can appear multiple times, in which case they have an associated name to disambiguate them: for example, the *zone* block (`zone example.com { ... };`) or the *key* block (`key mykey { ... };`).
- Certain blocks may be “nested” within other blocks. For example, the *zone* block may appear within a *view* block.

The description of each block in this manual lists its permissible locations.

Statement

- Statements define and control specific BIND behaviors.
- Statements may have a single parameter (a *Value*) or multiple parameters (*Argument/Value* pairs). For example, the *recursion* statement takes a single value parameter - in this case, the string *yes* or *no* (`recursion yes;`) - while the *port* statement takes a numeric value defining the DNS port number (`port 53;`). More complex statements take one or more argument/value pairs. The *also-notify* statement may take a number of such argument/value pairs, such as `also-notify port 5353;`, where *port* is the argument and 5353 is the corresponding value.
- Statements can appear in a single *block* - for example, an *algorithm* statement can appear only in a *key* block - or in multiple blocks - for example, an *also-notify* statement can appear in an *options* block where it has global (server-wide) scope, in a *zone* block where it has scope only for the specific zone (and
overrides any global statement), or even in a *view* block where it has scope for only that view (and overrides any global statement).

The file `named.conf` may further contain one or more instances of the *include Directive*. This directive is provided for administrative convenience in assembling a complete `named.conf` file and plays no subsequent role in BIND 9 operational characteristics or functionality.

**Note:** Over a period of many years the BIND ARM acquired a bewildering array of terminology. Many of the terms used described similar concepts and served only to add a layer of complexity, possibly confusion, and perhaps mystique to BIND 9 configuration. The ARM now uses only the terms **Block**, **Statement**, **Argument**, **Value**, and **Directive** to describe all entities used in BIND 9 configuration.

### 8.1.1 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 can be inserted 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/pound 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.

## 8.1.2 Configuration Layout Styles

BIND is very picky about opening and closing brackets/braces, semicolons, and all the other separators defined in the formal syntaxes in later sections. There are many layout styles that can assist in minimizing errors, as shown in the following examples:

```plaintext
// dense single-line style
zone "example.com" in{type secondary; file "secondary.example.com"; primaries {10.0.0.1;};};

// single-statement-per-line style
zone "example.com" in{
    type secondary;
    file "secondary.example.com";
    primaries {10.0.0.1;};
};

// spot the difference
zone "example.com" in{
    type secondary;
    file "sec.secondary.com";
    primaries {10.0.0.1;};
};
```

## 8.1.3 include Directive

```plaintext
include filename;
```
**include Directive Definition and Usage**

The include directive inserts the specified file (or files if a valid glob expression is detected) at the point where the include directive is encountered. The include directive 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.

### 8.1.4 Address Match Lists

**Syntax**

An address match list is a list of semicolon-separated `address_match_element`s.

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

Each element is then defined as:

```
address_match_element
```

```
[ ! ] { <ip_address> | <netprefix> | key <server_key> | <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:

- `ip_address`: an IP address (IPv4 or IPv6)
- `netprefix`: an IP prefix (in / notation)
- `server_key`: a key ID, as defined by the `key` statement
- `acl_name`: 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`, and `blackhole` 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, 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.

8.1.5 Glossary of Terms Used

Following is a list of terms 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
See Address Match Lists.

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

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

duration
A duration in BIND 9 can be written in three ways: as a single number representing seconds, as a string of numbers with TTL-style time-unit suffixes, or in ISO 6801 duration format.

Allowed TTL time-unit suffixes are: “W” (week), “D” (day), “H” (hour), “M” (minute), and “S” (second). Examples: “1W” (1 week), “3d12h” (3 days, 12 hours).

ISO 8601 duration format consists of the letter “P”, followed by an optional series of numbers with unit suffixes “Y” (year), “M” (month), “W” (week), and “D” (day); this may optionally be followed by the letter “T”, and another series of numbers with unit suffixes “H” (hour), “M” (minute), and “S” (second). Examples: “P3M10D” (3 months, 10 days), “P2WT12H” (2 weeks, 12 hours), “pt15m” (15 minutes). For more information on ISO 8601 duration format, see RFC 3339, appendix A.

Both TTL-style and ISO 8601 duration formats are case-insensitive.

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.

integer
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.

ip_address
An ipv4_address or ipv6_address.

ipv4_address
An IPv4 address with exactly four integer elements valued 0 through 255 and separated by dots (.), such as 192.168.1.1 (a “dotted-decimal” notation with all four elements present).

ipv6_address
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.

**netprefix**

An IP network specified as an *ip_address*, followed by a slash (/) and then the number of bits in the netmask. Trailing zeros in an *ip_address* 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 an IPv6-scoped address, the scope may be omitted. In that case, the prefix matches packets from any scope.

**percentage**

An integer value followed by % to represent percent.

**port**

An IP port *integer*. It 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.

**portrange**

A list of a port or a port range. A port range is specified in the form of range followed by two ports, *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. The asterisk (*) character is not allowed as a valid port or as a port range boundary.

**remote-servers**

A named list of one or more *ip_address* es with optional *tls_id*, *server_key*, and/or *port*. A remote-servers list may include other remote-servers lists. See *primary* block.

**server_key**

A *domain_name* representing the name of a shared key, to be used for transaction security. Keys are defined using key blocks.

**size**

A 64-bit unsigned integer. 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 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.

Some statements also accept the keywords unlimited or default: 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.

**tls_id**

A named TLS configuration object which defines a TLS key and certificate. See *tls* block.
8.2 Blocks

A BIND 9 configuration consists of blocks, statements, and comments.
The following blocks 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` for details.

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

- **:any:key-store**
  Describes a DNSSEC key store. See `key-store Grammar` for details.

- **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.

- **tls**
  Specifies configuration information for a TLS connection, including a `key-file`, `cert-file`, `ca-file`, `dhparam-file`, `remote-hostname`, `ciphers`, `protocols`, `prefer-server-ciphers`, and `session-tickets`.

- **http**
  Specifies configuration information for an HTTP connection, including `endpoints`, `listener-clients`, and `streams-per-connection`.

- **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.

8.2.1 acl Block Grammar

acl
Grammar: acl <string> { <address_match_element>; ... }; // may occur multiple times
Blocks: topmost
Tags: server

Assigns a symbolic name to an address match list.

8.2.2 acl Block 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.
8.2.3 controls Block Grammar

controls Grammar:

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

Blocks: topmost
Tags: server

Specifies control channels to be used to manage the name server.

8.2.4 controls Block 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.

**unix**

**Grammar:** unix quoted_string> perm <integer> owner <integer> group <integer> [ keys { <string>; ... } ] [ read-only <boolean> ]; // may occur multiple times

**Blocks:** controls
Tags: obsolete

Specifies a Unix domain socket as a control channel.

This option has been removed and using it will cause a fatal error.

**inet**

**Grammar controls:** inet ( <ipv4_address> | <ipv6_address> | * ) [ port ( <integer> | * ) ] allow { <address_match_element>; ... } [ keys { <string>; ... } ] [ read-only <boolean> ]; // may occur multiple times

**Grammar statistics-channels:** inet ( <ipv4_address> | <ipv6_address> | * ) [ port ( <integer> | * ) ] [ allow { <address_match_element>; ... } ]; // may occur multiple times

Blocks: controls, statistics-channels
Tags: server

Specifies a TCP socket as a control channel.

An `inet` control channel is a TCP socket listening at the specified `port` on the specified `ip_address`, which can be an IPv4 or IPv6 address. An `ip_address` 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_address` 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 `port`.

The ability to issue commands over the control channel is restricted by the `allow` and `keys` clauses.

**allow**

Connections to the control channel are permitted based on the `address_match_list`. This is for simple IP address-based filtering only; any `server_key` elements of the `address_match_list` are ignored.

**keys**

The primary authorization mechanism of the command channel is the list of `server_key`s. Each listed `key` is authorized to execute commands over the control channel. See *Administrative Tools* for information about configuring keys in `rndc`.

**read-only**

If the `read-only` argument is on, 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 `/etc/rndc.key`. To create an `rndc.key` file, run `rndc-confgen -a`.

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

### 8.2.5 key Block Grammar

**key**

Grammar:

```
key <string> { 
    algorithm <string>; 
    secret <string>; 
}; // may occur multiple times
```

**Blocks:** topmost, view

**Tags:** security

 Defines a shared secret key for use with TSIG or the command channel.

### 8.2.6 key Block Definition and Usage

The `key` statement defines a shared secret key for use with TSIG (see *TSIG*) or the command channel (see `controls`). 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 must be defined at the top level.

The `server_key`, 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.
**algorithm**

*Grammar:* algorithm <string>;

*Blocks:* key, view.key

*Tags:* security

Defines the algorithm to be used in a key clause.

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`.

**secret**

*Grammar:* secret <string>;

*Blocks:* key, view.key

*Tags:* security

Defines a Base64-encoded string to be used as the secret by the algorithm.

The `secret_string` is the secret to be used by the algorithm, and is treated as a Base64-encoded string.

### 8.2.7 key-store Block Grammar

**key-store**

*Grammar:*

```plaintext
key-store <string> { 
    directory <string>;
    pkcs11-uri <quoted_string>;
} // may occur multiple times
```

*Blocks:* topmost

*Tags:* dnssec

Configures a DNSSEC key store.

### 8.2.8 key-store Block Definition and Usage

The `key-store` statement defines how DNSSEC keys should be stored.

There is one built-in key store named `key-directory`. Configuring keys to use `key-store "key-directory"` is identical to using `key-directory`.

The following options can be specified in a `key-store` statement:
pkcs11-uri

Grammar: pkcs11-uri <quoted_string>;

Blocks: key-store

Tags: pkcs11, dnssec

The uri is a string that specifies a PKCS#11 URI Scheme (defined in RFC 7512). When set, named will try to create keys inside the corresponding PKCS#11 token. This requires BIND to be built with OpenSSL 3, and have a PKCS#11 provider configured.

8.2.9 logging Block Grammar

logging

Grammar:

logging {
    category <string> { <string>; ... }; // may occur multiple times
    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> ];
    }; // may occur multiple times
}

Blocks: topmost

Tags: logging

Configures logging options for the name server.

8.2.10 logging Block 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:
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

**channel**

**Grammar:**

```plaintext
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> ];
}; // may occur multiple times
```

**Blocks:** logging

**Tags:** logging

Defines a stream of data that can be independently logged.

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).

**null**

**Grammar:** null;

**Blocks:** logging.channel

**Tags:** logging

Causes all messages sent to the logging channel to be discarded.

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

**file**

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:

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

**syslog**

Grammar: syslog [ <syslog_facility> ];

Blocks: logging.channel

Tags: logging

Directs the logging channel to the system log.

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.

**severity**

Grammar: severity <log_severity>;

Blocks: logging.channel

Tags: logging

Defines the priority level of log messages.

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.

**stderr**

**Grammar:** stderr;

**Blocks:** logging.channel

**Tags:** logging

Directs the logging channel output to the server's standard error stream.

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:

```plaintext
cchannel 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**

**Grammar:** print-time ( iso8601 | iso8601-utc | local | <boolean> );

**Blocks:** logging.channel

**Tags:** logging

Specifies the time format for log messages.

**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.

**print-category**

**Grammar:** print-category <boolean>;

**Blocks:** logging.channel

**Tags:** logging

Includes the category in log messages.

If **print-category** is requested, then the category of the message is logged as well.
print-severity

Grammar: print-severity <boolean>

Blocks: logging.channel

Tags: logging

Includes the severity in log messages.

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

buffered

Grammar: buffered <boolean>

Blocks: logging.channel

Tags: logging

Controls flushing of log messages.

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 category.

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:

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

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

```plaintext
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:

```plaintext
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:

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

**category**

- **Grammar:** category <string> { <string>; ... }; // may occur multiple times
- **Blocks:** logging
Tags: logging

Specifies the type of data logged to a particular channel.

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.

cfg
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.

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. Falseclassifying 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].

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.

**rpz-passthru**
Information about RPZ PASSTHRU policy activity. This category allows pre-approved policy activity to be logged into a dedicated channel.

**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.

**sslkeylog**
TLS pre-master secrets (for debugging purposes).

**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:

```plaintext
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.

adterr
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.

8.2.11 parental-agents Block Grammar

Defines a list of delegation agents to be used by primary and secondary zones.
8.2.12 parental-agents Block Definition and Usage

parental-agents lists allow for a common set of parental agents to be easily used by multiple primary and secondary zones. A “parental agent” is a trusted DNS server that is queried to check if DS records for a given zones are up-to-date.

8.2.13 primaries Block Grammar

primaries

Grammar zone (mirror, redirect, secondary, stub):

primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };

Grammar topmost:

primaries <string> [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... }; // may occur multiple times

Blocks: topmost, zone (mirror, redirect, secondary, stub)

Tags: zone

Defines one or more primary servers for a zone.

8.2.14 primaries Block 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.)

To force the zone transfer requests to be sent over TLS, use tls keyword, e.g. primaries { 192.0.2.1 tls
tls-configuration-name; }; where tls-configuration-name refers to a previously defined tls statement.

Warning: Please note that TLS connections to primaries are not authenticated unless remote-hostname or ca-file are specified within the tls statement in use (see information on Strict TLS and Mutual TLS for more details). Not authenticated mode (Opportunistic TLS) provides protection from passive observers but does not protect from man-in-the-middle attacks on zone transfers.

8.2.15 options Block Grammar

options

Grammar:

options {
    allow-new-zones <boolean>;
    allow-notify { <address_match_element>; ... };
    allow-proxy { <address_match_element>; ... }; // experimental
    allow-proxy-on { <address_match_element>; ... }; // experimental
}

(continues on next page)
allow-query { <address_match_element>; ... }; allow-query-cache { <address_match_element>; ... }; allow-query-on { <address_match_element>; ... }; allow-recursion { <address_match_element>; ... }; allow-recursion-on { <address_match_element>; ... }; allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_element>; ... }; allow-update { <address_match_element>; ... }; allow-update-forwarding { <address_match_element>; ... }; also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <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-svcb <boolean>; check-wildcard <boolean>; clients-per-query <integer>; cookie-algorithm ( siphash24 ); cookie-secret <string>; // may occur multiple times deny-answer-addresses { <address_match_element>; ... } [ except-from { <string>; ... } ]; deny-answer-aliases { <string>; ... } [ except-from { <string>; ... } ]; dialup ( notify | notify-passive | passive | refresh | <boolean> ); // deprecated directory <quoted_string>; disable-algorithms <string> { <string>; ... }; // may occur multiple times disable-ds-digests <string> { <string>; ... }; // may occur multiple times disable-empty-zone <string>; // may occur multiple times dns64 <netprefix> { break-dnssec <boolean>; clients { <address_match_element>; ... }; exclude { <address_match_element>; ... }; mapped { <address_match_element>; ... }; recursive-only <boolean>;

8.2. Blocks
suffix <ipv6_address>;
}; // may occur multiple times
dns64-contact <string>;
dns64-server <string>;
dnskey-contact <string>;
dnskey-validity <integer>; // obsolete
dnsrps-enable <boolean>;
dnsrps-library <quoted_string>;
dnsrps-options { <unspecified-text> }

dnssec-accept-expired <boolean>;
dnssec-dnskey-kskonly <boolean>; // obsolete
dnssec-loadkeys-interval <integer>;
dnssec-must-be-secure <boolean>; // may occur multiple times,

->deprecated
dnssec-policy <string>;
dnssec-update-mode { maintain | no-resign }; // obsolete
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 };
dual-stack-servers [ port <integer> ] { { <quoted_string> [ port <integer> ... 

] | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <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 ) ];
flush-zones-on-shutdown <boolean>;
forward { first | only };
forwarders [ port <integer> ] [ tls <string> ] { { ( <ipv4_address> | <ipv6_ 

address> ) [ port <integer> ] | tls <string> } ... };
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>;
geoi-directory { <quoted_string> | none };
heartbeat-interval <integer>; // deprecated
hostname { <quoted_string> | none };
http-listener-clients <integer>;
http-port <integer>;
http-streams-per-connection <integer>;
https-port <integer>;
interface-interval <duration>;
ipv4only-contact <string>;
ipv4only-enable <boolean>;
ipv4only-server <string>;
ixfr-from-differences { primary | master | secondary | slave | <boolean>...
keep-response-order { <address_match_element>; ... }; // obsolete
key-directory <quoted_string>;
lame-ttl <duration>;
listen-on [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http <string> ] { <address_match_element>; ... }; // may occur multiple times
listen-on-v6 [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http <string> ] { <address_match_element>; ... }; // may occur multiple times
lmdb-mapsize <sizeval>;
managed-keys-directory <quoted_string>;
masterfile-format ( 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-records-per-type <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-types-per-name <integer>;
max-udp-size <integer>;
max-validation-failures-per-fetch <integer>; // experimental
max-validations-per-fetch <integer>; // experimental
memstatistics <boolean>;
memstatistics-file <quoted_string>;
message-compression <boolean>;
im-cache-ttl <duration>;
im-ncache-ttl <duration>;
im-refresh-time <integer>;
im-retry-time <integer>;
minimal-any <boolean>;
minimal-responses ( no-auth | no-auth-recursive | <boolean> );
multi-master <boolean>;
n 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> | * );
notify-source-v6 ( <ipv6_address> | * );
notify-to-soa <boolean>;
nsec3-test-zone <boolean>; // test only
nta-lifetime <duration>;

8.2. Blocks

(continues from previous page)
nta-recheck <duration>

nxdomain-redirect <string>

parental-source ( <ipv4_address> | * )

parental-source-v6 ( <ipv6_address> | * )

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> | * )

query-source-v6 [ address ] ( <ipv6_address> | * )

querylog <boolean>

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>

resolver-query-timeout <integer>

resolver-use-dns64 <boolean>

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 | passthru | tcp-only

  <quoted_string> ) ] [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [...

  nsdname-enable <boolean> ] [ ede <string> ]; ... } [ add-soa <boolean> ] [...

  break-dnssec <boolean> ] [ max-policy-ttl <duration> ] [ min-update-interval [...

  <duration> ] [ min-ns-dots <integer> ] [ nsip-wait-recurse <boolean> ] [...

  nsname-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-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>]; // obsolete
SIG0-checks-quota <integer>; // experimental
SIG0-checks-quota-exempt {<address_match_element>; ...}; // experimental
sortlist {<address_match_element>; ...}; // deprecated
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>
tcp-receive-buffer <integer>
tcp-send-buffer <integer>
tkey-domain quoted_string
tkey-gssapi-credential quoted_string
tkey-gssapi-keytab quoted_string
tls-port integer
transfer-format (many-answers | one-answer)
transfer-message-size <integer>
transfer-source (<ipv4_address> | * )
transfer-source-v6 (<ipv6_address> | * )
transfers-in <integer>
transfers-out <integer>
transfers-per-ns <integer>
trust-anchor-telemetry <boolean>
try-tcp-refresh <boolean>
udp-receive-buffer <integer>
udp-send-buffer <integer>
update-check-ksk <boolean>; // obsolete
update-quota <integer>
use-v4-udp-ports {<portrange>; ...}; // deprecated
use-v6-udp-ports {<portrange>; ...}; // deprecated
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>)

Blocks: topmost
8.2.16 options Block 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`

**Grammar:** `attach-cache <string>;`

**Blocks:** options, view

**Tags:** view

Allows multiple views to share a single cache database.

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:

```bash
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`, `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**

**Grammar key-store:** directory <string>;

**Grammar options:** directory <quoted_string>;

**Blocks:** key-store, options

**Tags:** server

Sets the server’s working 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**

**Grammar:**

dnstap { ( all | auth | client | forwarder | resolver | update ) [ ( query | response ) ]}; ... 

**Blocks:** options, view

**Tags:** logging

Enables logging of dnstap messages.

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:

```plaintext
dnstap {
auth;
client response;
```

(continues on next page)
resolver query;
};

**Note:** In the default configuration, the dnstap output for recursive resolver traffic does not include the IP addresses used by server-side sockets. This is caused by the fact that unless the *query source address* is explicitly set, these sockets are bound to wildcard IP addresses and determining the specific IP address used by each of them requires issuing a system call (i.e. incurring a performance penalty).

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 [https://dnstap.info](https://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**

Grammar: `fstrm-set-buffer-hint <integer>;

Blocks: options

Tags: logging

Sets the number of accumulated bytes in the output buffer before forcing a buffer flush.

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**

Grammar: `fstrm-set-flush-timeout <integer>;

Blocks: options

Tags: logging

Sets the number of seconds that unflushed data remains in the output buffer.

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**

Grammar: `fstrm-set-output-notify-threshold <integer>;

Blocks: options

Tags: logging

Sets the number of outstanding queue entries allowed on an input queue before waking the I/O thread.

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**

**Grammar:** `fstrm-set-output-queue-model ( mpsc | spsc );`

**Blocks:** options  
**Tags:** logging

Sets the queuing semantics to use for queue objects.

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**

**Grammar:** `fstrm-set-input-queue-size <integer> ;`

**Blocks:** options  
**Tags:** logging

Sets the number of queue entries to allocate for each input queue.

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**

**Grammar:** `fstrm-set-output-queue-size <integer> ;`

**Blocks:** options  
**Tags:** logging

Sets the number of queue entries allocated for each output queue.

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**

**Grammar:** `fstrm-set-reopen-interval <duration> ;`

**Blocks:** options  
**Tags:** logging

Sets the number of seconds to wait between attempts to reopen a closed output stream.

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**

**Grammar:** `dnstap-output ( file | unix ) <quoted_string> [ size ( unlimited | <size> ) ] [ versions ( unlimited | <integer> ) ] [ suffix ( increment | timestamp ) ] ;`

**Blocks:** options

---

8.2. Blocks
Configuresthepathtowhichthe `dnstap` frame stream is sent.

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**

```
Grammar: dnstap-identity ( <quoted_string> | none | hostname );
```

**Blocks**: options

**Tags**: logging

Specifies an identity string to send in `dnstap` messages.

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**

```
Grammar: dnstap-version ( <quoted_string> | none );
```

**Blocks**: options

**Tags**: logging

Specifies a version string to send in `dnstap` messages.

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**

```
Grammar: geoip-directory ( <quoted_string> | none );
```

**Blocks**: options

**Tags**: server

Specifies the directory containing GeoIP database files.

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`. See acl for details about geoip ACLs.

**key-directory**

Grammar: key-directory <quoted_string>;

Blocks: options, view, zone (primary, secondary)

Tags: dnssec

Indicates the directory where public and private DNSSEC key files are found.

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 `rndc.key` or `session.key`.)

**lmdb-mapsize**

Grammar: lmdb-mapsize <sizeval>;

Blocks: options, view

Tags: server

Sets a maximum size for the memory map of the new-zone database in LMDB database format.

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**

Grammar: managed-keys-directory <quoted_string>;

Blocks: options

Tags: dnssec

Specifies the directory in which to store the files that track managed DNSSEC keys.

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 filename, 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

Grammar: max-ixfr-ratio ( unlimited | <percentage> );
Blocks: options, view, zone (mirror, primary, secondary)
Tags: transfer

Sets the maximum size for IXFR responses to zone transfer requests.
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 100%.

new-zones-directory

Grammar: new-zones-directory <quoted_string>;
Blocks: options, view
Tags: zone

Specifies the directory where configuration parameters are stored for zones added by rndc addzone.
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

Grammar: qname-minimization ( strict | relaxed | disabled | off );
Blocks: options, view
Tags: query

Controls QNAME minimization behavior in the BIND 9 resolver.
When this is 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.
In relaxed mode named makes NS queries for <domain> as it walks down the tree.
disabled disables QNAME minimization completely. off is a synonym for disabled.
The current default is relaxed, but it may be changed to strict in a future release.

tkey-gssapi-keytab

Grammar: tkey-gssapi-keytab <quoted_string>;
Blocks: options
Tags: security
Sets the KRB5 keytab file to use for GSS-TSIG updates.

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

Grammar: tkey-gssapi-credential <quoted_string>;

Blocks: options

Tags: security

Sets the security credential for authentication keys requested by the GSS-TSIG protocol.

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

Grammar: tkey-domain <quoted_string>;

Blocks: options

Tags: security

Sets the domain appended to the names of all shared keys generated with TKEY.

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.

dump-file

Grammar: dump-file <quoted_string>;

Blocks: options

Tags: logging

Indicates the pathname of the file where the server dumps the database after rndc dumpdb.

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

Grammar: memstatistics-file <quoted_string>;

Blocks: options

Tags: logging
Sets the pathname of the file where the server writes memory usage statistics on exit.

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

**pid-file**

**Grammar:** pid-file ( <quoted_string> | none );

**Blocks:** options

**Tags:** server

Specifies the pathname of the file where the server writes its process ID.

This is the pathname of the file the server writes its process ID in. If not specified, the default is /run/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**

**Grammar:** recursing-file <quoted_string>;

**Blocks:** options

**Tags:** server

Specifies the pathname of the file where the server dumps queries that are currently recursing via *rndc recursing*.

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**

**Grammar:** statistics-file <quoted_string>;

**Blocks:** options

**Tags:** server, logging

Specifies the pathname of the file where the server appends statistics, when using *rndc stats*.

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**

**Grammar:** bindkeys-file <quoted_string>; // test only

**Blocks:** options

**Tags:** dnssec

Specifies the pathname of a file to override the built-in trusted keys provided by *named*.

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. This is intended for server testing.
secroots-file
Grammar:  secroots-file <quoted_string>;
Blocks:  options
Tags:  dnssec

Specifies the pathname of the file where the server dumps security roots, when using `rndc secroots`.
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
Grammar:  session-keyfile ( <quoted_string> | none );
Blocks:  options
Tags:  security

Specifies the pathname of the file where a TSIG session key is written, when generated by `named` for use by `nsupdate -l`.
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 `/run/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
Grammar:  session-keyname <string>;
Blocks:  options
Tags:  security

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

session-keyalg
Grammar:  session-keyalg <string>;
Blocks:  options
Tags:  security

Specifies the algorithm to use for the TSIG session key.
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
Grammar:  port <integer>;
Blocks:  options
Tags:  query, server

Specifies the UDP/TCP port number the server uses to receive and send DNS protocol traffic.
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.

**tls-port**

Grammar: `tls-port <integer>;

Blocks: options

Tags: query, server

Specifies the TCP port number the server uses to receive and send DNS-over-TLS protocol traffic. This is the TCP port number the server uses to receive and send DNS-over-TLS protocol traffic. The default is 853.

**https-port**

Grammar: `https-port <integer>;

Blocks: options

Tags: query, server

Specifies the TCP port number the server uses to receive and send DNS-over-HTTPS protocol traffic. This is the TCP port number the server uses to receive and send DNS-over-HTTPS protocol traffic. The default is 443.

**http-port**

Grammar: `http-port <integer>;

Blocks: options

Tags: query, server

Specifies the TCP port number the server uses to receive and send unencrypted DNS traffic via HTTP. This is the TCP port number the server uses to receive and send unencrypted DNS traffic via HTTP (a configuration that may be useful when encryption is handled by third-party software or by a reverse proxy).

**http-listener-clients**

Grammar: `http-listener-clients <integer>;

Blocks: options

Tags: server

Limits the number of active concurrent connections on a per-listener basis. This sets a hard limit on the number of active concurrent connections on a per-listener basis. The default value is 300; setting it to 0 removes the quota.

**http-streams-per-connection**

Grammar: `http-streams-per-connection <integer>;

Blocks: options

Tags: server
Limits the number of active concurrent HTTP/2 streams on a per-connection basis.

This sets a hard limit on the number of active concurrent HTTP/2 streams on a per-connection basis. The default value is 100; setting it to 0 removes the limit. Once the limit is exceeded, the server finishes the HTTP session.

**preferred-glue**

*Grammar:* `preferred-glue <string>;

*Blocks:* options, view

*Tags:* query

Controls the order of glue records in an A or AAAA response.

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.

**disable-algorithms**

*Grammar:* `disable-algorithms <string> { <string>; ... }; // may occur multiple times`

*Blocks:* options, view

*Tags:* dnssec

Disables DNSSEC algorithms from a specified zone.

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**

*Grammar:* `disable-ds-digests <string> { <string>; ... }; // may occur multiple times`

*Blocks:* options, view

*Tags:* zone, dnssec

Disables DS digest types from a specified zone.

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

**Warning:** This option is deprecated and will be removed in a future version of BIND.

**Grammar:**
dnssec-must-be-secure <string> <boolean>; // may occur multiple times, deprecated

**Blocks:** options, view

**Tags:** deprecated

Defines hierarchies that must or may not be secure (signed and validated).

This option is deprecated and will be removed in a future release.

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**

**Grammar:**

```
dns64 <netprefix> {  
  break-dnssec <boolean>;  
  clients { <address_match_element>; ... };  
  exclude { <address_match_element>; ... };  
  mapped { <address_match_element>; ... };  
  recursive-only <boolean>;  
  suffix <ipv6_address>;  
}; // may occur multiple times
```

**Blocks:** options, view

**Tags:** query

Instructs *named* to return mapped IPv4 addresses to AAAA queries when there are no AAAA records.

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**

**Grammar:**
dns64-server <string>;

**Blocks:** options, view

**Tags:** server

Specifies the name of the server for *dns64* zones.
**dns64-contact**

**Grammar:** `dns64-contact <string>`;

**Blocks:** options, view

**Tags:** server

Specifies the name of the contact for `dns64` zones.

`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.

`dns64` will also cause IPV4ONLY.ARPA to be created if not explicitly disabled using `ipv4only-enable`.

**clients**

**Grammar:** `clients { <address_match_element>; ... }`;

**Blocks:** options.dns64, view.dns64

**Tags:** query

Specifies an access control list (ACL) of clients that are affected by a given `dns64` directive.

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

**mapped**

**Grammar:** `mapped { <address_match_element>; ... }`;

**Blocks:** options.dns64, view.dns64

**Tags:** query

Specifies an access control list (ACL) of IPv4 addresses that are to be mapped to the corresponding A RRset in `dns64`.

Each `dns64` block 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`;

**exclude**

**Grammar:** `exclude { <address_match_element>; ... }`;

**Blocks:** options.dns64, view.dns64

**Tags:** query

Allows a list of IPv6 addresses to be ignored if they appear in a domain name’s AAAA records in `dns64`.

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 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`.

---

**8.2. Blocks**
suffix
Grammar: suffix <ipv6_address>;
Blocks: options.dns64, view.dns64
Tags: query

Defines trailing bits for mapped IPv4 address bits in dns64.
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.

recursive-only
Grammar: recursive-only <boolean>;
Blocks: options.dns64, view.dns64
Tags: query

Toggles whether dns64 synthesis occurs only for recursive queries.
If recursive-only is set to yes, the DNS64 synthesis only happens for recursive queries. The default is no.

break-dnssec
Grammar: break-dnssec <boolean>;
Blocks: options.dns64, view.dns64
Tags: query

Enables dns64 synthesis even if the validated result would cause a DNSSEC validation failure.
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 ::;
};

resolver-use-dns64
Grammar: resolver-use-dns64 <boolean>;
Blocks: options, view
Tags: server

Specifies whether to apply DNS64 mappings when sending queries.
If resolver-use-dns64 is set to yes, then the IPv4-to-IPv6 address transformations specified by the dns64 option will be applied to IPv4 server addresses to which recursive queries are sent. This allows a server to perform
lookups via a NAT64 connection; queries that would have been sent via IPv4 are instead sent to mapped IPv6 addresses. The default is no.

```
ipv4only-enable
Grammar: ipv4only-enable <boolean>;
Blocks: options, view
Tags: query
```

Enables automatic IPv4 zones if a `dns64` block is configured.

This enables or disables automatic zones `ipv4only.arpa`, `170.0.0.192.in-addr.arpa`, and `171.0.0.192.in-addr.arpa`.

By default these zones are loaded if `dns64` is configured.

```
ipv4only-server
Grammar: ipv4only-server <string>;
Blocks: options, view
Tags: query, server
```

Specifies the name of the server for the IPV4ONLY.ARPA zone created by `dns64`.

```
ipv4only-contact
Grammar: ipv4only-contact <string>;
Blocks: options, view
Tags: server
```

Specifies the contact for the IPV4ONLY.ARPA zone created by `dns64`.

`ipv4only-server` and `ipv4only-contact` can be used to specify the name of the server and contact for the IPV4ONLY.ARPA zone created by `dns64`.

```
dnssec-loadkeys-interval
Grammar: dnssec-loadkeys-interval <integer>;
Blocks: options, view, zone (primary, secondary)
Tags: dnssec
```

Sets the frequency of automatic checks of the DNSSEC key repository.

When a zone is configured with `dnssec-policy`; its key repository must be checked periodically to see if the next step of a key rollover is due. The `dnssec-loadkeys-interval` option sets the default interval of key repository checks, in minutes, in case the next key event cannot be calculated (for example because a DS record needs to be published).

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` block. The default is `none`.
dnssec-update-mode

Grammar: dnssec-update-mode ( maintain | no-resign ); // obsolete

Blocks: options, view, zone (primary, secondary)

Tags: obsolete

This option no longer has any effect.

nta-lifetime

Grammar: nta-lifetime <duration>;

Blocks: options, view

Tags: dnssec

Specifies the lifetime, in seconds, for negative trust anchors added via `rndc nta`.

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

Grammar: nta-recheck <duration>;

Blocks: options, view

Tags: dnssec

Specifies the time interval for checking whether negative trust anchors added via `rndc nta` are still necessary.

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

Warning: This option is deprecated and will be removed in a future version of BIND.
**Grammar dnssec-policy:** max-zone-ttl <duration>;

**Grammar options, view, zone (primary, redirect):** max-zone-ttl ( unlimited | <duration>);

// deprecated

**Blocks:** dnssec-policy, options, view, zone (primary, redirect)

**Tags:** deprecated

Specifies a maximum permissible time-to-live (TTL) value, in seconds.

This should now be configured as part of *dnssec-policy*. Use of this option in *options*, *view* and *zone* blocks is a fatal error if *dnssec-policy* has also been configured for the same zone. In zones without *dnssec-policy*, this option is deprecated, and will be rendered non-operational in a future release.

*max-zone-ttl* specifies a maximum permissible TTL value in seconds. For convenience, TTL-style time-unit suffixes may be used to specify the maximum value. When a zone file is loaded, any record encountered with a TTL higher than *max-zone-ttl* causes the zone to be rejected.

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.

When used in *options*, *view* and *zone* blocks, setting *max-zone-ttl* to zero is equivalent to “unlimited”.

**stale-answer-ttl**

**Grammar:** stale-answer-ttl <duration>;

**Blocks:** options, view

**Tags:** query

Specifies the time to live (TTL) to be returned on stale answers, in seconds.

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**

**Grammar:** serial-update-method ( date | increment | unixtime );

**Blocks:** options, view, zone (primary)

**Tags:** zone

Specifies the update method to be used for the zone serial number in the SOA record.

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**

**Grammar:**
```
zone-statistics ( full | terse | none | <boolean> );
```

**Blocks:**
- options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)

**Tags:**
- logging, zone

Controls the level of statistics gathered for all zones.

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), and also information about the currently ongoing incoming zone transfers.

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**

**Grammar:**
```
automatic-interface-scan <boolean>;
```

**Blocks:**
- options

**Tags:**
- server

Controls the automatic rescanning of network interfaces when addresses are added or removed.

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**

**Grammar:**
```
allow-new-zones <boolean>;
```

**Blocks:**
- options, view

**Tags:**
- server, zone

Controls the ability to add zones at runtime via `rndc addzone`.

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**

Grammar: `auth-nxdomain <boolean>;`

Blocks: options, view

Tags: query

Controls whether BIND, acting as a resolver, provides authoritative NXDOMAIN (domain does not exist) answers. 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**

Grammar: `memstatistics <boolean>;`

Blocks: options

Tags: server, logging

Controls whether memory statistics are written to the file specified by `memstatistics-file` at exit. 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**

**Warning:** This option is deprecated and will be removed in a future version of BIND.

Grammar: `dialup ( notify | notify-passive | passive | refresh | <boolean> );` // deprecated

Blocks: options, view, zone (primary, secondary, stub)

Tags: deprecated

Concentrates zone maintenance so that all transfers take place once every `heartbeat-interval`, ideally during a single call.

This option is deprecated and will be removed in a future release.

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 (provided 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.

<table>
<thead>
<tr>
<th>dialup mode</th>
<th>normal refresh</th>
<th>heart-beat refresh</th>
<th>heart-beat notify</th>
</tr>
</thead>
<tbody>
<tr>
<td>no (default)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>notify</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>refresh</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>passive</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>notify-passive</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

**flush-zones-on-shutdown**

Grammar: `flush-zones-on-shutdown <boolean>`;

Blocks: options

Tags: zone

Controls whether pending zone writes are flushed when the name server exits.

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`.

**root-key-sentinel**

Grammar: `root-key-sentinel <boolean>`;

Blocks: options, view

Tags: server

Controls whether BIND 9 responds to root key sentinel probes.

If `yes`, respond to root key sentinel probes as described in RFC 8509. The default is `yes`.

**reuseport**

Grammar: `reuseport <boolean>`;

Blocks: options

Tags: server

Enables kernel load-balancing of sockets.
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

Grammar: message-compression <boolean>;

Blocks: options, view

Tags: query

Controls whether DNS name compression is used in responses to regular queries.

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

Grammar: minimal-responses ( no-auth | no-auth-recursive | <boolean> );

Blocks: options, view

Tags: query

Controls whether the server only adds records to the authority and additional data sections when they are required (e.g. delegations, negative responses). This improves server performance.

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, 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.

Responses to DNSKEY, DS, CDNSKEY, and CDS requests will never have optional additional records added. Responses to NS requests will always have additional section processing.

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**.

**minimal-any**

  **Grammar**: minimal-any <boolean>;
  
  **Blocks**: options, view
  
  **Tags**: query

 Controls whether the server replies with only one of the RRsets for a query name, when generating a positive response to a query of type ANY over UDP.

 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**

  **Grammar**: notify ( explicit | master-only | primary-only | <boolean> );
  
  **Blocks**: options, view, zone (mirror, primary, secondary)
  
  **Tags**: transfer

 Controls whether NOTIFY messages are sent on zone changes.

 If set to **yes** (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes; see [using 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**

  **Grammar**: notify-to-soa <boolean>;
  
  **Blocks**: options, view, zone (primary, secondary)
  
  **Tags**: transfer

 Controls whether the name servers in the NS RRset are checked against the SOA MNAME.

 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**

**Grammar:** recursion <boolean>;

**Blocks:** options, view

**Tags:** query

Defines whether recursion and caching are allowed.

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**

**Grammar:** request-nsid <boolean>;

**Blocks:** options, server, view, view.server

**Tags:** query

Controls whether an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution.

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.

**require-cookie**

**Grammar:** require-cookie <boolean>;

**Blocks:** server, view.server

**Tags:** query

Controls whether responses without a server cookie are accepted.

The require-cookie clause can be used to indicate that the remote server is known to support DNS COOKIE. Setting this option to yes causes named to always retry a request over TCP when it receives a UDP response without a DNS COOKIE from the remote server, even if UDP responses with DNS COOKIE have not been sent by this server before. This prevents spoofed answers from being accepted without a retry over TCP when named has not yet determined whether the remote server supports DNS COOKIE. Setting this option to no (the default) causes named to rely on autodetection of DNS COOKIE support to determine when to retry a request over TCP.

---

**Note:** If a UDP response is signed using TSIG, named accepts it even if require-cookie is set to yes and the response does not contain a DNS COOKIE.

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.
**require-server-cookie**

**Grammar:** require-server-cookie <boolean>;

**Blocks:** options, view

**Tags:** query

Controls whether a valid server cookie is required before sending a full response to a UDP request.

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**

**Grammar:** answer-cookie <boolean>;

**Blocks:** options

**Tags:** query

Controls whether COOKIE EDNS replies are sent in response to client queries.

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**

**Grammar:** send-cookie <boolean>;

**Blocks:** options, server, view, view.server

**Tags:** query

Controls whether a COOKIE EDNS option is sent along with a query.

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 *named* server may determine that COOKIE is not supported by the remote server and not add a COOKIE EDNS option to requests.
The default is **yes**.

### stale-answer-enable

**Grammar:** stale-answer-enable <boolean>;

**Blocks:** options, view

**Tags:** query, server

Enables the returning of “stale” cached answers when the name servers for a zone are not answering.

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

**Grammar:** stale-answer-client-timeout ( disabled | off | <integer> );

**Blocks:** options, view

**Tags:** query, server

Defines the amount of time (in milliseconds) that `named` waits before attempting to answer a query with a stale RRset from cache.

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 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.

When this option is enabled, the only supported value in the current version of BIND 9 is **0**. Non-zero values generate a warning message, and are treated as **0**.

### stale-cache-enable

**Grammar:** stale-cache-enable <boolean>;

**Blocks:** options, view

**Tags:** query, server

Enables the retention of “stale” cached answers.

If **yes**, enable the retaining of “stale” cached answers. Default **no**.

### stale-refresh-time

**Grammar:** stale-refresh-time <duration>;

**Blocks:** options, view
Tags: query, server

Sets the time window for the return of “stale” cached answers before the next attempt to contact, if the name servers for a given zone are not responding.

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 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`

Grammar: `nocookie-udp-size <integer>;;`

Blocks: options, view

Tags: query

Sets the maximum size of UDP responses that are sent to queries without a valid server COOKIE.

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 1232.

`cookie-algorithm`

Grammar: `cookie-algorithm ( siphash24 );`

Blocks: options

Tags: server

Sets the algorithm to be used when generating a server cookie.

This sets the algorithm to be used when generating the server cookie. The default is “siphash24”, which is the only supported option, as the previously supported “aes” option has been removed.

`cookie-secret`

Grammar: `cookie-secret <string>; // may occur multiple times`

Blocks: options

Tags: server

Specifies a shared secret used for generating and verifying EDNS COOKIE options within an anycast cluster.

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.

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

Grammar: response-padding { <address_match_element>; ... } block-size <integer>;

Blocks: options, view

Tags: query

Adds an EDNS Padding option to encrypted messages, to reduce the chance of guessing the contents based on size.

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

Grammar: trust-anchor-telemetry <boolean>;

Blocks: options, view

Tags: dnssec

Instructs named to send specially formed queries once per day to domains for which trust anchors have been configured.

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.

provide-ixfr

Grammar: provide-ixfr <boolean>;

Blocks: options, server, view, view.server

Tags: transfer

Controls whether a primary responds to an incremental zone request (IXFR) or only responds with a full zone transfer (AXFR).
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.

**request-ixfr**

**Grammar:** request-ixfr <boolean>;

**Blocks:** options, server, view, zone (mirror, secondary), view.server

**Tags:** transfer

Controls whether a secondary requests an incremental zone transfer (IXFR) or a full zone transfer (AXFR).

The `request-ixfr` statement determines whether the local server, acting as a secondary, requests incremental zone transfers from the given remote server, a primary.

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` statements 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.

It may also be set in the `zone` block; if set there, it overrides the global or view setting for that zone. It may also be set in the `server` block.

**request-expire**

**Grammar:** request-expire <boolean>;

**Blocks:** options, server, view, zone (mirror, secondary), view.server

**Tags:** query, transfer

Specifies whether the local server requests the EDNS EXPIRE value, when acting as a secondary.

The `request-expire` statement 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`.

**match-mapped-addresses**

**Grammar:** match-mapped-addresses <boolean>;

**Blocks:** options

**Tags:** server

Allows IPv4-mapped IPv6 addresses to match address-match list entries for corresponding IPv4 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

Grammar zone (mirror, primary, secondary): ixfr-from-differences <boolean>;

Grammar options, view: ixfr-from-differences ( primary | master | secondary | slave | <boolean> );

Blocks: options, view, zone (mirror, primary, secondary)

Tags: transfer

Controls how IXFR transfers are calculated.

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

Grammar: multi-master <boolean>;

Blocks: options, view, zone (mirror, secondary, stub)

Tags: transfer

Controls whether serial number mismatch errors are logged.

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.

dnssec-validation

Grammar: dnssec-validation ( yes | no | auto );

Blocks: options, view

Tags: dnssec

Enables DNSSEC validation in named.

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. This trust anchor is provided as part of BIND and is kept up-to-date using Dynamic Trust Anchor Management key management. Adding an explicit static key using the trust-anchors statement with a static-key anchor...
type (or using the deprecated `trusted-keys` statement) for the root zone is not supported with the `auto` setting, and is treated as a configuration error.

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 `trust-anchors` is not configured, it is a configuration error. If `trust-anchors` does not include a valid root key, then validation does not take place for names which are not covered by any of the configured trust anchors.

If set to `no`, DNSSEC validation is disabled. (Note: the resolver will still set the DO bit in outgoing queries indicating that it can accept DNSSEC responses, even if `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 compiled into `named` and is current as of the release date. If the root key changes, a running BIND server will detect this and roll smoothly to the new key, but newly-installed servers will be unable to start validation, so BIND must be upgraded to a newer version.

`validate-except`

**Grammar:**

```
validate-except { <string>; ... };
```

**Blocks:** options, view

**Tags:** dnssec

Specifies a list of domain names at and beneath which DNSSEC validation should not be performed.

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`

**Grammar:**

```
dnssec-accept-expired <boolean>;
```

**Blocks:** options, view

**Tags:** dnssec

Instructs BIND 9 to accept expired DNSSEC signatures when validating.

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

`querylog`

**Grammar:**

```
querylog <boolean>;
```

**Blocks:** options

**Tags:** server, logging

Specifies whether query logging should be active when `named` first starts.

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**

Grammar zone (hint, mirror, primary, secondary, stub): `check-names ( fail | warn | ignore )`;

Grammar options, view: `check-names ( primary | master | secondary | slave | response ) ( fail | warn | ignore )`; // may occur multiple times

Blocks: options, view, zone (hint, mirror, primary, secondary, stub)

Tags: query, server

Restricts the character set and syntax of certain domain names in primary files and/or DNS responses received from the network.

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 type primary zones the default is fail. For type 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**

Grammar: `check-dup-records ( fail | warn | ignore )`;

Blocks: options, view, zone (primary)

Tags: query, dnssec

Checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS.

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**

Grammar: `check-mx ( fail | warn | ignore )`;

Blocks: options, view, zone (primary)

Tags: zone

Checks whether an MX record appears to refer to an IP address.

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**

Grammar: `check-wildcard <boolean>`;

Blocks: options, view, zone (primary)
Tags: zone

Checks for non-terminal wildcards.

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**

**Grammar:** check-integrity <boolean>;

**Blocks:** options, view, zone (primary)

**Tags:** zone

Performs post-load zone integrity checks on primary zones.

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). DS records not at delegations are rejected. 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**

**Grammar:** check-mx-cname ( fail | warn | ignore );

**Blocks:** options, view, zone (primary)

**Tags:** zone

Sets the response to MX records that refer to CNAMEs.

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

**check-srv-cname**

**Grammar:** check-srv-cname ( fail | warn | ignore );

**Blocks:** options, view, zone (primary)

**Tags:** zone

Sets the response to SRV records that refer to CNAMEs.

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

**check-sibling**

**Grammar:** check-sibling <boolean>;

**Blocks:** options, view, zone (primary)
Tags: zone

Specifies whether to check for sibling glue when performing integrity checks.
When performing integrity checks, also check that sibling glue exists. The default is yes.

**check-spf**

Grammar: `check-spf ( warn | ignore );`

Blocks: options, view, zone (primary)

Tags: zone

Specifies whether to check for a TXT Sender Policy Framework record, if an SPF record is present.
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`.

**check-svcb**

Grammar: `check-svcb <boolean>;`

Blocks: options, view, zone (primary)

Tags: zone

Specifies whether to perform additional checks on SVCB records.
If `yes`, checks that SVCB records that start with a `_dns` label prefixed by an optional `_<port>` label (e.g. `_443._dns.ns1.example`), have an `alpn` parameter and that the `dohpath` parameter exists when the `alpn` indicates that it should be present. The default is `yes`.

**zero-no-soa-ttl**

Grammar: `zero-no-soa-ttl <boolean>;`

Blocks: options, view, zone (mirror, primary, secondary)

Tags: query, server, zone

Specifies whether to set the time to live (TTL) of the SOA record to zero, when returning authoritative negative responses to SOA queries.
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**

Grammar: `zero-no-soa-ttl-cache <boolean>;`

Blocks: options, view

Tags: query, server, zone

Sets the time to live (TTL) to zero when caching a negative response to an SOA query.
If `yes`, when caching a negative response to an SOA query set the TTL to zero. The default is `no`.  

8.2. Blocks 147
update-check-ksk
   Grammar: update-check-ksk <boolean>; // obsolete
   Blocks: options, view, zone (primary, secondary)
   Tags: obsolete
   This option no longer has any effect.

dnssec-dnskey-kskonly
   Grammar: dnssec-dnskey-kskonly <boolean>; // obsolete
   Blocks: options, view, zone (primary, secondary)
   Tags: obsolete
   This option no longer has any effect.

try-tcp-refresh
   Grammar: try-tcp-refresh <boolean>;
   Blocks: options, view, zone (mirror, secondary)
   Tags: transfer

   Specifies that BIND 9 should attempt to refresh a zone using TCP if UDP queries fail.
   If yes, try to refresh the zone using TCP if UDP queries fail. The default is yes.

dnssec-secure-to-insecure
   Grammar: dnssec-secure-to-insecure <boolean>; // obsolete
   Blocks: options, view, zone (primary)
   Tags: obsolete
   This option no longer has any effect.

synth-from-dnssec
   Grammar: synth-from-dnssec <boolean>;
   Blocks: options, view
   Tags: dnssec

   Enables support for RFC 8198, Aggressive Use of DNSSEC-Validated Cache.
   This option enables support for RFC 8198, Aggressive Use of DNSSEC-Validated Cache. It allows the resolver
to send a smaller number of queries when resolving queries for DNSSEC-signed domains by synthesizing answers
from cached NSEC and other RRsets that have been proved to be correct using DNSSEC. The default is yes.

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.

```plaintext
forward
  Grammar: forward ( first | only );
  Blocks: options, view, zone (forward, primary, secondary, static-stub, stub)
  Tags: query

Allows or disallows fallback to recursion if forwarding has failed; it is always used in conjunction with the forwarders statement.

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
  Grammar: forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
  Blocks: options, view, zone (forward, primary, secondary, static-stub, stub)
  Tags: query

Defines one or more hosts to which queries are forwarded.

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 a TLS transport. A default port number and a TLS transport can be set for the entire list.

If a TLS configuration is specified, named will use DNS-over-TLS (DoT) connections when connecting to the specified IP address(es), using the TLS configuration referenced by the tls statement.

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.

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
  Grammar: dual-stack-servers [ port <integer> ] { ( <quoted_string> [ port <integer> ] | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ); ... };
  Blocks: options, view
  Tags: server
Specifies hostnames or addresses of machines with access to both IPv4 and IPv6 transports.

This specifies hostnames 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**

**Grammar:** allow-notify { <address_match_element>; ... };

**Blocks:** options, view, zone (mirror, secondary)

**Tags:** transfer

Defines an `address_match_list` that is allowed to send NOTIFY messages for the zone, in addition to addresses defined in the `primary` option for the zone.

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 `primary` for the zone. `allow-notify` can be used to expand the list of permitted hosts, not to reduce it.

**allow-proxy**

**Warning:** This option is experimental and subject to change.

**Grammar:** allow-proxy { <address_match_element>; ... }; // experimental

**Blocks:** options, view

**Tags:** server

Defines an `address_match_list` for the client addresses allowed to send PROXYv2 headers.

The default `address_match_list` is `none`, which means that no client is allowed to do that by default for security reasons, as the PROXYv2 protocol provides an easy way to spoof both source and destination addresses.

This `address_match_list` is primarily meant to have addresses and subnets of the proxies that are allowed to send PROXYv2 headers to BIND. In most cases, we do not recommend setting this `address_match_list` to be very allowing, in particular, to set it to `any`, especially in the cases when PROXYv2 headers can be accepted on publically available networking interfaces.

The specified option is the only option that matches against real peer addresses when PROXYv2 headers are used. Most of the options that work with peer addresses, use the ones extracted from PROXYv2 headers.

Also, see: `allow-proxy-on`
**allow-proxy-on**

**Warning:** This option is experimental and subject to change.

**Grammar:** allow-proxy-on { <address_match_element>; ... }; // experimental

**Blocks:** options, view

**Tags:** server

Defines an *address_match_list* for the interface addresses allowed to accept PROXYv2 headers. The option is mostly intended for multi-homed configurations.

The default *address_match_list* is *any*, which means that accepting PROXYv2 is allowed on any interface.

The option is useful in cases when you need to have precise control over which interfaces PROXYv2 is allowed, as it is the only one that matches against real interface addresses when PROXYv2 headers are used. Most of the options that work with interface addresses will use the ones extracted from PROXYv2 headers.

You may want to set *allow-proxy* first.

**allow-query**

**Grammar:** allow-query { <address_match_element>; ... };

**Blocks:** options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)

**Tags:** query

Specifies which hosts (an IP address list) are allowed to send queries to this resolver.

*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**

**Grammar:** allow-query-on { <address_match_element>; ... };

**Blocks:** options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)

**Tags:** query

Specifies which local addresses (an IP address list) are allowed to send queries to this resolver. Used in multi-homed configurations.

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.
allow-query-cache

Grammar: allow-query-cache { <address_match_element>; ... };

Blocks: options, view

Tags: query

Specifies which hosts (an IP address list) can access this server's cache and thus effectively controls recursion.

Defines an address_match_list of IP address(es) which are allowed to issue queries that access the local cache. Without access to the local cache recursive queries are effectively useless so, in effect, this statement (or its default) controls recursive behavior. This statement's default setting depends on:

1. If recursion no; present, defaults to allow-query-cache {none;};. No local cache access permitted.

2. If recursion yes; (default) then, if allow-recursion present, defaults to the value of allow-recursion. Local cache access permitted to the same address_match_list as allow-recursion.

3. If recursion yes; (default) then, if allow-recursion is not present, defaults to allow-query-cache {localnets; localhost;};. Local cache access permitted to address_match_list localnets and localhost IP addresses only.

allow-query-cache-on

Grammar: allow-query-cache-on { <address_match_element>; ... };

Blocks: options, view

Tags: query

Specifies which hosts (an IP address list) can access this server's cache. Used on servers with multiple interfaces.

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

Grammar: allow-recursion { <address_match_element>; ... };

Blocks: options, view

Tags: query

Defines an address_match_list of clients that are allowed to perform recursive queries.

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
Grammar: allow-recursion-on { <address_match_element>; ... };
Blocks: options, view
Tags: query, server

Specifies which local addresses can accept recursive queries. 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
Grammar: allow-update { <address_match_element>; ... };
Blocks: options, view, zone (primary)
Tags: transfer

Defines an address_match_list of hosts that are allowed to submit dynamic updates for primary zones. A simple access control list. 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.

Updates are written to the zone’s filename that is set in file.

allow-update-forwarding
Grammar: allow-update-forwarding { <address_match_element>; ... };
Blocks: options, view, zone (mirror, secondary)
Tags: transfer

Defines an address_match_list of hosts that are allowed to submit dynamic updates to a secondary server for transmission to a primary. 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

Grammar: allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_element>; ... };

Blocks: options, view, zone (mirror, primary, secondary)

Tags: transfer

Defines an address_match_list of hosts that are allowed to transfer the zone information from this server. 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.

The transport level limitations can also be specified. In particular, zone transfers can be restricted to a specific port and/or DNS transport protocol by using the options port and transport. Either option can be specified; if both are used, both constraints must be satisfied in order for the transfer to be allowed. Zone transfers are currently only possible via the TCP and TLS transports.

For example: allow-transfer port 853 transport tls { any; }; allows outgoing zone transfers to any host using the TLS transport over port 853.

If allow-transfer is not specified, then the default is none; outgoing zone transfers are disabled.

Warning: Please note that incoming TLS connections are not authenticated at the TLS level by default. Please use TSIG to authenticate requestors or consider implementing Mutual TLS authentication.

blackhole

Grammar: blackhole { <address_match_element>; ... };

Blocks: options

Tags: query

Defines an address_match_list of hosts to ignore. The server will neither respond to queries from nor send queries to these addresses.

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.

no-case-compress

Grammar: no-case-compress { <address_match_element>; ... };

Blocks: options, view

Tags: server

Specifies a list of addresses that require case-insensitive compression in responses.

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-sensitive 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

**Grammar:** resolver-query-timeout <integer>;

**Blocks:** options, view

**Tags:** query

Specifies the length of time, in milliseconds, that a resolver attempts to resolve a recursive query before failing.

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, ports, and protocols that the server can use to answer queries may be specified using the listen-on and listen-on-v6 options.

**listen-on**

**Grammar:** listen-on [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http <string> ] { <address_match_element>; ... }; // may occur multiple times

**Blocks:** options

**Tags:** server

Specifies the IPv4 addresses on which a server listens for DNS queries.

**listen-on-v6**

**Grammar:** listen-on-v6 [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http <string> ] { <address_match_element>; ... }; // may occur multiple times

**Blocks:** options

**Tags:** server
Specifies the IPv6 addresses on which a server listens for DNS queries.

The `listen-on` and `listen-on-v6` statements can each take an optional port, PROXYv2 support switch, TLS configuration identifier, and/or HTTP configuration identifier, in addition to an `address_match_list`.

The `address_match_list` in `listen-on` specifies the IPv4 addresses on which the server will listen. (IPv6 addresses are ignored, with a logged warning.) The server listens on all interfaces allowed by the address match list. If no `listen-on` is specified, the default is to listen for standard DNS queries on port 53 of all IPv4 interfaces.

`listen-on-v6` takes an `address_match_list` of IPv6 addresses. The server listens on all interfaces allowed by the address match list. If no `listen-on-v6` is specified, the default is to listen for standard DNS queries on port 53 of all IPv4 interfaces.

When specified, the PROXYv2 support switch `proxy` allows enabling the PROXYv2 protocol support. The PROXYv2 protocol provides the means for passing connection information, such as a client’s source and destination addresses and ports, across multiple layers of NAT or TCP/UDP proxies to back-end servers. The addresses passed to by the PROXYv2 protocol are then used instead of the peer and interface addresses provided by the operating system.

The `proxy` switch can have the following values:

- `plain` - accept plain PROXYv2 headers. It is the only valid option for transports that do not employ encryption. In the case of transports that employ encryption, it instructs BIND that PROXYv2 headers are sent without encryption before the TLS handshake. In that case, only PROXYv2 headers are not encrypted.

- `encrypted` - accept encrypted PROXYv2 headers. In the case of transports that employ encryption, it instructs BIND that PROXYv2 headers are sent encrypted immediately after the TLS handshake. The option is valid only for the transports that employ encryption.

You must consult your proxying front-end software documentation to decide which value you need to use. If in doubt, use `plain` for encrypted transports, especially for DNS-over-HTTPS (DoH), but DNS-specific software is likely to need `encrypted`.

It should be noted that when PROXYv2 is enabled on a listener, it loses the ability to accept regular DNS queries without associated PROXYv2 headers.

In some cases, PROXYv2 headers might not contain usable source and destination addresses. In particular, that happens when the headers use `LOCAL` command or the headers that use unspecified or unsupported by BIND address types. If otherwise correct, such headers are accepted by BIND and the real endpoint addresses are used in these cases.

The PROXYv2 protocol is designed to be extensible and can carry additional information in the form of type-length-values (TLVs). Many of the types are defined in the protocol specification, and for some of these, we do a reasonable amount of validation in order to detect and reject ill-formed or hand-crafted headers. Apart from that, this additional data, while accepted, is not currently used by BIND for anything else.

By default, no client is allowed to send queries that contain PROXYv2 protocol headers, even when support for the protocol is enabled in a `listen-on` statement. If you are interested in enabling the PROXYv2 protocol support, you may also want to take a look at `allow-proxy` and `allow-proxy-on` options to adjust the corresponding ACLs.

If a TLS configuration is specified, `named` will listen for DNS-over-TLS (DoT) connections, using the key and certificate specified in the referenced `tls` statement. If the name `ephemeral` is used, an ephemeral key and certificate created for the currently running `named` process will be used.

If an HTTP configuration is specified, `named` will listen for DNS-over-HTTPS (DoH) connections using the HTTP endpoint specified in the referenced `http` statement. If the name `default` is used, then `named` will listen for connections at the default endpoint, `/dns-query`.
Use of an http specification requires tls to be specified as well. If an unencrypted connection is desired (for example, on load-sharing servers behind a reverse proxy), tls none may be used.

If a port number is not specified, the default is 53 for standard DNS, 853 for DNS over TLS, 443 for DNS over HTTPS, and 80 for DNS over HTTP (unencrypted). These defaults may be overridden using the port, tls-port, https-port, and http-port options.

Multiple listen-on statements are allowed. For example:

```plaintext
listen-on { 5.6.7.8; }
listen-on port 1234 { !1.2.3.4; 1.2/16; }
listen-on port 8853 tls ephemeral { 4.3.2.1; }
listen-on port 8453 tls ephemeral http myserver { 8.7.6.5; }
listen-on port 5300 proxy plain { !1.2.3.4; 1.2/16; }
listen-on port 8953 proxy encrypted tls ephemeral { 4.3.2.1; }
listen-on port 8553 proxy plain tls ephemeral http myserver { 8.7.6.5; }
```

The first two lines instruct the name server to listen for standard DNS queries on port 53 of 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. The third line instructs the server to listen for DNS-over-TLS connections on port 8853 of the IP address 4.3.2.1 using the ephemeral key and certificate. The fourth line enables DNS-over-HTTPS connections on port 8453 of address 8.7.6.5, using the ephemeral key and certificate, and the HTTP endpoint or endpoints configured in an http statement with the name myserver.

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

```plaintext
listen-on-v6 { any; }
listen-on-v6 port 1234 { !2001:db8::/32; any; }
listen-on-v6 port 8853 tls example-tls { 2001:db8::100; }
listen-on-v6 port 8453 tls example-tls http default { 2001:db8::100; }
listen-on-v6 port 8000 tls none http myserver { 2001:db8::100; }
listen-on-v6 port 53000 proxy plain { !2001:db8::/32; any; }
listen-on-v6 port 8953 proxy encrypted tls example-tls { 2001:db8::100; }
listen-on-v6 port 8553 proxy plain tls example-tls http default { 2001:db8::100; }
```

The first two lines instruct the name server to listen for standard DNS queries on port 53 of any IPv6 addresses, and on port 1234 of IPv6 addresses that are not in the prefix 2001:db8::/32. The third line instructs the server to listen for DNS-over-TLS connections on port 8853 of the address 2001:db8::100, using a TLS key and certificate specified in the a tls statement with the name example-tls. The fourth instructs the server to listen for DNS-over-HTTPS connections, again using example-tls, on the default HTTP endpoint. The fifth line, in which the tls parameter is set to none, instructs the server to listen for unencrypted DNS queries over HTTP at the endpoint specified in myserver.

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

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

### Query Address

**query-source**

**Grammar:** query-source [ address ] ( <ipv4_address> | * );

**Blocks:** options, server, view, view.server

**Tags:** query

Controls the IPv4 address from which queries are issued.
query-source-v6

Grammar: query-source-v6 [ address ] ( <ipv6_address> | * );

Blocks: options, server, view, view.server

Tags: query

Controls the IPv6 address from which queries are issued.

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.

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

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

Note: port configuration is deprecated. A warning will be logged when this parameter is used.

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.

use-v4-udp-ports

Warning: This option is deprecated and will be removed in a future version of BIND.

Grammar: use-v4-udp-ports { <portrange>; ... }; // deprecated

Blocks: options

Tags: deprecated

Specifies a list of ports that are valid sources for UDP/IPv4 messages.

use-v6-udp-ports

Warning: This option is deprecated and will be removed in a future version of BIND.

Grammar: use-v6-udp-ports { <portrange>; ... }; // deprecated

Blocks: options

Tags: deprecated

Specifies a list of ports that are valid sources for UDP/IPv6 messages.

These statements, which are deprecated and will be removed in a future release, specify a list of IPv4 and IPv6 UDP ports that are used as source ports for UDP messages.
If `port` is * or is omitted, a random port number from a pre-configured range is selected and used for each query. The port range(s) are specified in the `use-v4-udp-ports` (for IPv4) and `use-v6-udp-ports` (for IPv6) options.

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; }
```

### avoid-v4-udp-ports

**Warning:** This option is deprecated and will be removed in a future version of BIND.

**Grammar:** `avoid-v4-udp-ports { <portrange>; ... };` // deprecated

**Blocks:** options

**Tags:** deprecated

Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv4 messages.

### avoid-v6-udp-ports

**Warning:** This option is deprecated and will be removed in a future version of BIND.

**Grammar:** `avoid-v6-udp-ports { <portrange>; ... };` // deprecated

**Blocks:** options

**Tags:** deprecated

Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv6 messages.

These statements, which are deprecated and will be removed in a future release, specific ranges of port numbers to exclude from those specified in the `avoid-v4-udp-ports` and `avoid-v6-udp-ports` options, respectively.

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

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

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, or 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 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.

**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.

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

**Warning:** The configured `port` must not be the 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**

**Grammar:**
```
also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };
```

**Blocks:** options, view, zone (mirror, primary, secondary)

**Tags:** transfer

Defines one or more hosts that are sent NOTIFY messages when zone changes occur.

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
Grammar: max-transfer-time-in <integer>;
Blocks: options, view, zone (mirror, secondary, stub)
Tags: transfer

Specifies the number of minutes after which inbound zone transfers are terminated. 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
Grammar: max-transfer-idle-in <integer>;
Blocks: options, view, zone (mirror, secondary, stub)
Tags: transfer

Specifies the number of minutes after which inbound zone transfers making no progress are terminated. 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).

Note: The inbound zone transfers are also affected by tcp-idle-timeout. The max-transfer-idle-in will close the inbound zone transfer if there was no complete AXFR or no complete IXFR chunk. The tcp-idle-timeout will close the connection if there’s no progress on the TCP level.

max-transfer-time-out
Grammar: max-transfer-time-out <integer>;
Blocks: options, view, zone (mirror, primary, secondary)
Tags: transfer

Specifies the number of minutes after which outbound zone transfers are terminated. 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
Grammar: max-transfer-idle-out <integer>;
Blocks: options, view, zone (mirror, primary, secondary)
Tags: transfer
Specifies the number of minutes after which outbound zone transfers making no progress are terminated.

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**

---

**Grammar:** notify-rate <integer>;

**Blocks:** options

**Tags:** zone, transfer

Specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations.

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**

---

**Grammar:** startup-notify-rate <integer>;

**Blocks:** options

**Tags:** zone, transfer

Specifies the rate at which NOTIFY requests are sent when the name server is first starting, or when new zones have been added.

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**

---

**Grammar:** serial-query-rate <integer>;

**Blocks:** options

**Tags:** transfer

Defines an upper limit on the number of queries per second issued by the server, when querying the SOA RRs used for zone transfers.

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**

---

**Grammar:** transfer-format ( many-answers | one-answer );

**Blocks:** options, server, view, view.server

**Tags:** transfer

Controls whether multiple records can be packed into a message during zone transfers.
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 block.

**transfer-message-size**

**Grammar:** transfer-message-size <integer>;

**Blocks:** options

**Tags:** transfer

Limits the uncompressed size of DNS messages used in zone transfers over TCP.

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**

**Grammar:** transfers-in <integer>;

**Blocks:** options

**Tags:** transfer

Limits the number of concurrent inbound zone transfers.

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**

**Grammar:** transfers-out <integer>;

**Blocks:** options

**Tags:** transfer

Limits the number of concurrent outbound zone transfers.

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**

**Grammar:** transfers-per-ns <integer>;

**Blocks:** options

**Tags:** transfer
Limits the number of concurrent inbound zone transfers from a remote server.

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**

Grammar: transfer-source ( <ipv4_address> | * );

Blocks: options, server, view, zone (mirror, secondary, stub), view.server

Tags: transfer

Defines which local IPv4 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.

**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:** port configuration is deprecated. A warning will be logged when this parameter is used.

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

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

**transfer-source-v6**

Grammar: transfer-source-v6 ( <ipv6_address> | * );

Blocks: options, server, view, zone (mirror, secondary, stub), view.server

Tags: transfer

Defines which local IPv6 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.

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

**notify-source**

Grammar: notify-source ( <ipv4_address> | * );

Blocks: options, server, view, zone (mirror, primary, secondary), view.server

Tags: transfer
Defines the IPv4 address (and optional port) to be used for outgoing NOTIFY messages.

`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:** port configuration is deprecated. A warning will be logged when this parameter is used.

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

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

**notify-source-v6**

**Grammar:**

```
notify-source-v6 ( <ipv6_address> | * );
```

**Blocks:** options, server, view, zone (mirror, primary, secondary), view.server

**Tags:** transfer

Defines the IPv6 address (and optional port) to be used for outgoing NOTIFY messages.

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

**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**

**Grammar:**

```
max-journal-size ( default | unlimited | <sizeval> );
```

**Blocks:** options, view, zone (mirror, primary, secondary)

**Tags:** transfer

Controls the size of journal files.

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**

**Grammar:**

```
max-records <integer>;
```

**Blocks:** options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)
Sets the maximum number of records permitted in a zone.
This sets the maximum number of records permitted in a zone. The default is zero, which means the maximum is unlimited.

**max-records-per-type**

**Grammar:** `max-records-per-type <integer>;`

**Blocks:** options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)

**Tags:** server

Sets the maximum number of records that can be stored in an RRset
This sets the maximum number of resource records that can be stored in an RRset in a database. When configured in `options` or `view`, it controls the cache database; it also sets the default value for zone databases, which can be overridden by setting it at the `zone` level.

If set to a positive value, any attempt to cache or to add to a zone an RRset with more than the specified number of records will result in a failure. If set to 0, there is no cap on RRset size. The default is 100.

**max-types-per-name**

**Grammar:** `max-types-per-name <integer>;`

**Blocks:** options, view, zone (mirror, primary, redirect, secondary, static-stub, stub)

**Tags:** server

Sets the maximum number of RR types that can be stored for an owner name
This sets the maximum number of resource record types that can be stored for a single owner name in a database. When configured in `options` or `view`, it controls the cache database and sets the default value for zone databases, which can be overridden by setting it at the `zone` level.

An RR type and its corresponding signature are counted as two types. So, for example, a signed node containing A and AAAA records has four types: A, RRSIG(A), AAAA, and RRSIG(AAAA).

The behavior is slightly different for zone and cache databases:
In a zone, if `max-types-per-name` is set to a positive number, any attempt to add a new resource record set to a name that already has the specified number of types will fail.
In a cache, if `max-types-per-name` is set to a positive number, an attempt to add a new resource record set to a name that already has the specified number of types will temporarily succeed so that the query can be answered. However, the newly added RRset will immediately be purged.

Certain high-priority types, including SOA, CNAME, DNSKEY, and their corresponding signatures, are always cached. If `max-types-per-name` is set to a very low value, then it may be ignored to allow high-priority types to be cached.

When `max-types-per-name` is set to 0, there is no cap on the number of RR types. The default is 100.

**recursive-clients**

**Grammar:** `recursive-clients <integer>;`

**Blocks:** options

**Tags:** query
Specifies the maximum number of concurrent recursive queries the server can perform.

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**

**Grammar:** `tcp-clients <integer>;

**Blocks:** options

**Tags:** server

Specifies the maximum number of simultaneous client TCP connections accepted by the server.

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

**clients-per-query**

**Grammar:** `clients-per-query <integer>;

**Blocks:** options, view

**Tags:** server

Sets the initial minimum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.

This sets the initial value (minimum) number of simultaneous recursive 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 value is 10.

The chosen value should reflect how many queries come in for a given name in the time it takes to resolve that name.

**max-clients-per-query**

**Grammar:** `max-clients-per-query <integer>;

**Blocks:** options, view

**Tags:** server

Sets the maximum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.

This sets the maximum number of simultaneous recursive clients for any given query (<qname,qtype,qclass>) that the server accepts before dropping additional clients.
If the number of queries exceeds `clients-per-query`, `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, up to a limit of `max-clients-per-query`. The estimate is then lowered after 20 minutes if it has remained unchanged.

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

`max-validations-per-fetch`  

**Warning:** This option is experimental and subject to change.

**Grammar:** `max-validations-per-fetch <integer>; // experimental`

**Blocks:** options, view

**Tags:** server

Set the maximum number of DNSSEC validations that can happen in a single fetch. This is an experimental setting to set the maximum number of DNSSEC validations that can happen in a single resolver fetch. The default is 16.

`max-validation-failures-per-fetch`  

**Warning:** This option is experimental and subject to change.

**Grammar:** `max-validation-failures-per-fetch <integer>; // experimental`

**Blocks:** options, view

**Tags:** server

Set the maximum number of DNSSEC validation failures that can happen in a single fetch. This is an experimental setting to set the maximum number of DNSSEC validation failures that can happen in a single resolver fetch. The default is 1.

`fetches-per-zone`  

**Grammar:** `fetches-per-zone <integer> [ ( drop | fail ) ];`

**Blocks:** options, view

**Tags:** query, server

Sets the maximum number of simultaneous iterative queries allowed to any one domain before the server blocks new queries for data in or beneath that 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.)

$fetches-per-server$

Grammar: $fetches-per-server$ <integer> [ ( drop | fail ) ];

Blocks: options, view

Tags: query, server

Sets the maximum number of simultaneous iterative queries allowed to be sent by a server to an upstream name server before the server blocks additional queries.

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.

$fetch-quota-params$

Grammar: $fetch-quota-params$ <integer> <fixedpoint> <fixedpoint> <fixedpoint>;

Blocks: options, view

Tags: query, server

Sets the parameters for dynamic resizing of the $fetches-per-server$ quota in response to detected congestion.

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.

max-cache-size

Grammar: max-cache-size ( default | unlimited | <sizeval> | <percentage> );

Blocks: options, view
Tags: server

Sets the maximum amount of memory to use for an individual cache database and its associated metadata.

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.

Note: max-cache-size does not work reliably for the maximum amount of memory of 100 MB or lower.

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

Grammar: tcp-listen-queue <integer>;

Blocks: options
**Tags:** server

Sets the listen-queue depth.

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**

**Grammar:** tcp-initial-timeout <integer>;

**Blocks:** options

**Tags:** query, server

Sets the amount of time (in milliseconds) that the server waits on a new TCP connection for the first message from the client.

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**

**Grammar:** tcp-idle-timeout <integer>;

**Blocks:** options

**Tags:** query

Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is not in use.

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**

**Grammar:** tcp-keepalive-timeout <integer>;

**Blocks:** options

**Tags:** query

Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is in use.

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**

*Grammar:* `tcp-advertised-timeout <integer>;`

*Blocks:* options

*Tags:* query

Sets the timeout value (in milliseconds) that the server sends in responses containing the EDNS TCP keepalive option.

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`.

**update-quota**

*Grammar:* `update-quota <integer>;`

*Blocks:* options

*Tags:* server

Specifies the maximum number of concurrent DNS UPDATE messages that can be processed by the server.

This is the maximum number of simultaneous DNS UPDATE messages that the server will accept for updating local authoritative zones or forwarding to a primary server. The default is 100.

**sig0checks-quota**

*Warning:* This option is experimental and subject to change.

*Grammar:* `sig0checks-quota <integer>; // experimental`

*Blocks:* options

*Tags:* server

Specifies the maximum number of concurrent SIG(0) signature checks that can be processed by the server.

This is the maximum number of simultaneous SIG(0)-signed messages that the server accepts. If the quota is reached, then `named` answers with a status code of REFUSED. The value of 0 disables the quota. The default is 1.

**sig0checks-quota-exempt**

*Warning:* This option is experimental and subject to change.

*Grammar:* `sig0checks-quota-exempt { <address_match_element>; ... }; // experimental`
Blocks: options
Tags: server

Exempts specific clients or client groups from SIG(0) signature checking quota.

DNS clients can be exempted from SIG(0) signature checking quota with the `sig0checks-quota-exempt` clause using their IP and/or Network addresses. The default value is an empty list.

Example:

```
sig0checks-quota-exempt {
    10.0.0.0/8;
    2001:db8::100;
};
```

### Periodic Task Intervals

**heartbeat-interval**

```
Warning: This option is deprecated and will be removed in a future version of BIND.
```

Grammar: `heartbeat-interval <integer>;` // deprecated
Blocks: options
Tags: deprecated

Sets the interval at which the server performs zone maintenance tasks for all zones marked as `dialup`.

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.

This option is deprecated and will be removed in a future release.

**interface-interval**

Grammar: `interface-interval <duration>;`
Blocks: options
Tags: server

Sets the interval at which the server scans the network interface list.

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.

**sortlist**

---

**Warning:** This option is deprecated and will be removed in a future version of BIND.

**Grammar:**

```
sortlist { <address_match_element>; ... }; // deprecated
```

**Blocks:** options, view

**Tags:** query, deprecated

Controls the ordering of RRs returned to the client, based on the client's IP address.

This option is deprecated and will be removed in a future release.

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
    { 192.168.2/24;
        192.168.2/24;
            { 192.168.3/24; }; }; }
    // IF on class C 192.168.4/24 THEN use .4
    { 192.168.4/24;
        192.168.4/24;
            { 192.168.5/24; }; }; }
    // IF on class C 192.168.5/24 THEN use .5
    { 192.168.5/24;
        192.168.5/24;
            { 192.168.6/24; }; }; }
    // IF on class C 192.168.6/24 THEN use .6
    { 192.168.6/24;
        192.168.6/24;
            { 192.168.7/24; }; }; }
    // IF on class C 192.168.7/24 THEN use .7
    { 192.168.7/24;
        192.168.7/24;
            { 192.168.8/24; }; }; }
    // IF on class C 192.168.8/24 THEN use .8
    { 192.168.8/24;
        192.168.8/24;
            { 192.168.9/24; }; }; }
}
```

(continues on next page)
The following example illustrates 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.

**rrset-order**

**Grammar:**

```
rrset-order { [ class <string> ] [ type <string> ] [ name <quoted_string> ] <string> <string>; ... };
```

**Blocks:** options, view

**Tags:** query

Defines the order in which equal RRs (RRsets) are returned.

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

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.
This value is deprecated and will be removed in a future release.

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:

```plaintext
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:
<table>
<thead>
<tr>
<th>QNAME</th>
<th>QTYPE</th>
<th>RRset Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo.isc.org</td>
<td>A</td>
<td>random</td>
</tr>
<tr>
<td>foo.isc.org</td>
<td>AAA</td>
<td>cyclic</td>
</tr>
<tr>
<td>foo.isc.org</td>
<td>TXT</td>
<td>none</td>
</tr>
<tr>
<td>sub.foo.isc.org</td>
<td>all</td>
<td>none</td>
</tr>
<tr>
<td>bar.isc.org</td>
<td>all</td>
<td>fixed</td>
</tr>
<tr>
<td>sub.bar.isc.org</td>
<td>all</td>
<td>random</td>
</tr>
<tr>
<td>baz.isc.org</td>
<td>all</td>
<td>none</td>
</tr>
<tr>
<td>sub.baz.isc.org</td>
<td>all</td>
<td>cyclic</td>
</tr>
</tbody>
</table>

### Tuning

#### lame-ttl

**Grammar:** lame-ttl <duration>;

**Blocks:** options, view

**Tags:** server

Sets the resolver's lame cache.

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

#### servfail-ttl

**Grammar:** servfail-ttl <duration>;

**Blocks:** options, view

**Tags:** server

Sets the length of time (in seconds) that a SERVFAIL response is cached.

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

**Grammar:** min-ncache-ttl <duration>;

**Blocks:** options, view

**Tags:** server

Specifies the minimum retention time (in seconds) for storage of negative answers in the server’s cache.

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

Grammar: min-cache-ttl <duration>;
Blocks: options, view
Tags: server

Specifies the minimum time (in seconds) that the server caches ordinary (positive) answers.
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

Grammar: max-ncache-ttl <duration>;
Blocks: options, view
Tags: server

Specifies the maximum retention time (in seconds) for storage of negative answers in the server’s cache.
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

Grammar: max-cache-ttl <duration>;
Blocks: options, view
Tags: server

Specifies the maximum time (in seconds) that the server caches ordinary (positive) answers.
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

Grammar: max-stale-ttl <duration>;
Blocks: options, view
Tags: server

Specifies the maximum time that the server retains records past their normal expiry, to return them as stale records.
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.

**sig-validity-interval**

Grammar: `sig-validity-interval <integer> [, <integer>]`; // obsolete

Blocks: options, view, zone (primary, secondary)

Tags: obsolete

This option no longer has any effect.

**dnskey-sig-validity**

Grammar: `dnskey-sig-validity <integer>`; // obsolete

Blocks: options, view, zone (primary, secondary)

Tags: obsolete

This option no longer has any effect.

**sig-signing-nodes**

Grammar: `sig-signing-nodes <integer>`;

Blocks: options, view, zone (primary, secondary)

Tags: dnssec

Specifies the maximum number of nodes to be examined in each quantum, when signing a zone with a new DNSKEY.

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**

Grammar: `sig-signing-signatures <integer>`;

Blocks: options, view, zone (primary, secondary)

Tags: dnssec

Specifies the threshold for the number of signatures that terminates processing a quantum, when signing a zone with a new DNSKEY.

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**

Grammar: `sig-signing-type <integer>`;

Blocks: options, view, zone (primary, secondary)

Tags: dnssec
Specifies a private RDATA type to use when generating signing-state records.
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 \texttt{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 \texttt{rndc signing -clear keyid/algorithm zone}. To clear all of the completed signing-state records for a zone, use \texttt{rndc signing -clear all zone}.

\textbf{min-refresh-time}

\textbf{Grammar:} min-refresh-time <integer>;
\textbf{Blocks:} options, view, zone (mirror, secondary, stub)
\textbf{Tags:} transfer

Limits the zone refresh interval to no more often than the specified value, in seconds.
This option controls the server's behavior on refreshing a zone (querying for SOA changes). Usually, the SOA refresh values for the zone are used; however, these values are set by the primary, giving secondary server administrators little control over their contents.
This option allows the administrator to set a minimum refresh time in seconds per-zone, per-view, or globally. This option is valid for secondary and stub zones, and clamps the SOA refresh time to the specified value.
The default is 300 seconds.

\textbf{max-refresh-time}

\textbf{Grammar:} max-refresh-time <integer>;
\textbf{Blocks:} options, view, zone (mirror, secondary, stub)
\textbf{Tags:} transfer

Limits the zone refresh interval to no less often than the specified value, in seconds.
This option controls the server's behavior on refreshing a zone (querying for SOA changes). Usually, the SOA refresh values for the zone are used; however, these values are set by the primary, giving secondary server administrators little control over their contents.
This option allows the administrator to set a maximum refresh time in seconds per-zone, per-view, or globally. This option is valid for secondary and stub zones, and clamps the SOA refresh time to the specified value.
The default is 2419200 seconds (4 weeks).

\textbf{min-retry-time}

\textbf{Grammar:} min-retry-time <integer>;
\textbf{Blocks:} options, view, zone (mirror, secondary, stub)
\textbf{Tags:} transfer

Limits the zone refresh retry interval to no more often than the specified value, in seconds.
This option controls the server's behavior on retrying failed zone transfers. Usually, the SOA retry values for the zone are used; however, these values are set by the primary, giving secondary server administrators little control over their contents.
This option allows the administrator to set a minimum retry time in seconds per-zone, per-view, or globally. This option is valid for secondary and stub zones, and clamps the SOA retry time to the specified value.

The default is 500 seconds.

**max-retry-time**

**Grammar:** max-retry-time <integer>;

**Blocks:** options, view, zone (mirror, secondary, stub)

**Tags:** transfer

Limits the zone refresh retry interval to no less often than the specified value, in seconds.

This option controls the server's behavior on retrying failed zone transfers. Usually, the SOA retry values for the zone are used; however, these values are set by the primary, giving secondary server administrators little control over their contents.

This option allows the administrator to set a maximum retry time in seconds per-zone, per-view, or globally. This option is valid for secondary and stub zones, and clamps the SOA retry time to the specified value.

The default is 1209600 seconds (2 weeks).

**edns-udp-size**

**Grammar:** edns-udp-size <integer>;

**Blocks:** options, server, view, view.server

**Tags:** query

Sets the maximum advertised EDNS UDP buffer size to control the size of packets received from authoritative servers in response to recursive queries.

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 1232. Query timeouts observed for any given server affect the buffer size advertised in queries sent to that server. Depending on observed packet dropping patterns, the query is retried over TCP. Per-server EDNS statistics are only retained in memory for the lifetime of a given server's ADB entry.

According to the measurements done by multiple parties the default value should not be causing the fragmentation 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, i.e. configures a static value for a given `server` block.

**max-udp-size**

**Grammar:** max-udp-size <integer>;

**Blocks:** options, server, view, view.server
Tags: query

Sets the maximum EDNS UDP message size sent by named.

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

Grammar: masterfile-format ( raw | text );

Blocks: options, view, zone (mirror, primary, redirect, secondary, stub)

Tags: server, zone

Specifies the file format of zone files.

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. Zone files in raw format should be generated with the same check level as that specified in the named configuration file.

When configured in options, this statement sets the masterfile-format for all zones, but it can be over-ridden 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

Grammar: masterfile-style ( full | relative );

Blocks: options, view, zone (mirror, primary, redirect, secondary, stub)

Tags: server

Specifies the format of zone files during a dump, when the masterfile-format is text.

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**

**Grammar:** `max-recursion-depth <integer>;`

**Blocks:** options, view

**Tags:** server

Sets the maximum number of levels of recursion permitted at any one time while servicing a recursive query.

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**

**Grammar:** `max-recursion-queries <integer>;`

**Blocks:** options, view

**Tags:** query, server

Sets the maximum number of iterative queries while servicing a recursive query.

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**

**Grammar:** `notify-delay <integer>;`

**Blocks:** options, view, zone (mirror, primary, secondary)

**Tags:** zone, transfer

Sets the delay (in seconds) between sending sets of NOTIFY messages for a zone.

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**

**Grammar:** `max-rsa-exponent-size <integer>;`

**Blocks:** options

**Tags:** query, dnssec

Sets the maximum RSA exponent size (in bits) when validating.

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**

**Grammar:** `prefetch <integer> [ <integer> ];`

**Blocks:** options, view
Specifies the “trigger” time-to-live (TTL) value at which prefetch of the current query takes place.

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 or equal 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**

**Grammar:** v6-bias <integer>;

**Blocks:** options, view

**Tags:** query, server

Indicates the number of milliseconds of preference to give to IPv6 name servers.

When determining the next name server to try, this indicates by how many milliseconds to prefer IPv6 name servers. The default is 50 milliseconds.

**tcp-receive-buffer**

**Grammar:** tcp-receive-buffer <integer>;

**Blocks:** options

**Tags:** server

Sets the operating system’s receive buffer size for TCP sockets.

**udp-receive-buffer**

**Grammar:** udp-receive-buffer <integer>;

**Blocks:** options

**Tags:** server

Sets the operating system’s receive buffer size for UDP sockets.

These options control the operating system’s receive buffer sizes (SO_RCVBUF) for TCP and UDP sockets, respectively. Buffering at the operating system level can prevent packet drops during brief load spikes, but if the buffer size is set too high, a running server could get clogged with outstanding queries that have already timed out. The default is 0, which means the operating system’s default value should be used. The minimum configurable value is 4096; any nonzero value lower than that is silently raised. The maximum value is determined by the kernel, and values exceeding the maximum are silently reduced.
tcp-send-buffer
Grammar: tcp-send-buffer <integer>;
Blocks: options
Tags: server

Sets the operating system’s send buffer size for TCP sockets.

udp-send-buffer
Grammar: udp-send-buffer <integer>;
Blocks: options
Tags: server

Sets the operating system’s send buffer size for UDP sockets.

These options control the operating system’s send buffer sizes (SO_SNDBUF) for TCP and UDP sockets, respectively. Buffering at the operating system level can prevent packet drops during brief load spikes, but if the buffer size is set too high, a running server could get clogged with outstanding queries that have already timed out. The default is 0, which means the operating system’s default value should be used. The minimum configurable value is 4096; any nonzero value lower than that is silently raised. The maximum value is determined by the kernel, and values exceeding the maximum are silently reduced.

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) 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
Grammar: version { <quoted_string> | none };
Blocks: options
Tags: server

Specifies the version number of the server to return in response to a version.bind query.

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
Grammar: hostname { <quoted_string> | none };
Blocks: options
Tags: server
server-id

**Grammar:** server-id ( <quoted_string> | none | hostname );

**Blocks:** options

**Tags:** server

Specifies the ID of the server to return in response to a **ID.SERVER** query.

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
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

**Grammar:** empty-server <string>;

**Blocks:** options, view

**Tags:** server, zone

Specifies the server name in the returned SOA record for empty zones.

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

**Grammar:** empty-contact <string>;

**Blocks:** options, view

**Tags:** server, zone
Specifies the contact name in the returned SOA record for empty zones.
This specifies the contact name that appears in the returned SOA record for empty zones. If none is specified, "." is used.

**empty-zones-enable**

Grammar: empty-zones-enable <boolean>;

Blocks: options, view

Tags: server, zone

Enables or disables all empty zones.
This enables or disables all empty zones. By default, they are enabled.

**disable-empty-zone**

Grammar: disable-empty-zone <string>; // may occur multiple times

Blocks: options, view

Tags: server, zone

Disables individual empty zones.
This disables individual empty zones. By default, none are disabled. This option can be specified multiple times.

**Content Filtering**

**deny-answer-addresses**

Grammar: deny-answer-addresses  {  <address_match_element>;  ...  }  [  except-from  {  <string>;  ...  }  ];

Blocks: options, view

Tags: query

Rejects A or AAAA records if the corresponding IPv4 or IPv6 addresses match a given address_match_list.
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.

In the address_match_list of the deny-answer-addresses option, only ip_address and netprefix are meaningful; any server_key is silently ignored.

**deny-answer-aliases**

Grammar: deny-answer-aliases  {  <string>;  ...  }  [  except-from  {  <string>;  ...  }  ];

Blocks: options, view

Tags: query

Rejects CNAME or DNAME records if the “alias” name matches a given list of domain_name elements.
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 list of domain_name elements of the deny-answer-aliases option, where “match” means the alias name is a subdomain of one of the listed domain names. If the optional list is specified in the except-from argument, 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, returned by an “example.com” server is accepted.

If a response message is rejected due to the filtering, the entire message is discarded without being cached, and a SERVFAIL 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:

```plaintext
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:

```plaintext
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:

```plaintext
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**

**Grammar:**

```plaintext
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> ] [ ede <string> ]; ...
... } [ add-soa <boolean> ] [ break-dnssec <boolean> ] [ max-policy-ttl <duration> ] [ min-update-interval <duration> ] [ min-ns-dots <integer> ] [ nsip-wait-recurse <boolean> ] [ nsdname-wait-recurse <boolean> ] [ qname-wait-recurse <boolean> ] [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [ nsdname-enable <boolean> ] [ dnsrps-enable <boolean> ] [ dnsrps-options { <unspecified-text> } ];
```

**Blocks:** options, view

**Tags:** query, server, zone, security

Specifies response policy zones for the view or among global options.

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; }`.

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. To speed up processing at the cost of precision, the nsdname-wait-recursively option can be used; when set to no, RPZ-NSDNAME rules are only applied when authoritative servers for the query name have already been looked up and cached. If authoritative servers for the query name are not in the cache, the RPZ-NSDNAME rule is ignored, but the authoritative servers for the query name are looked up in the background and the rule is applied to subsequent queries. The default is yes, meaning RPZ-NSDNAME rules are always applied, even if authoritative servers for the query name need to be looked up first.

**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-recursively 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.

**dnhrsps-enable**

Grammar: `dnhrsps-enable <boolean>;`

Blocks: options, view

Tags: server, security

Turns on the DNS Response Policy Service (DNSRPS) interface.

The `dnhrsps-enable yes` option turns on the DNS Response Policy Service (DNSRPS) interface, if it has been compiled in named using `configure --enable-dnhrsps`.

**dnhrsps-library**

Grammar: `dnhrsps-library <quoted_string>;`

Blocks: options

Tags: server, security

Turns on the DNS Response Policy Service (DNSRPS) interface.

This option specifies the path to the DNSRPS provider library. Typically this library is detected when building with `configure --enable-dnhrsps` and does not need to be specified in `named.conf`; the option exists to override the default library for testing purposes.

**dnhrsps-options**

Grammar: `dnhrsps-options { <unspecified-text> };`

Blocks: options, view

Tags: server, security

Provides additional RPZ configuration settings, which are passed to the DNS Response Policy Service (DNSRPS) provider library.

The block provides additional RPZ configuration settings, which are passed through to the DNSRPS provider library. Multiple DNSRPS settings in an `dnhrsps-options` string should be separated with semi-colons (;). The DNSRPS provider library is passed a configuration string consisting of the `dnhrsps-options` text, concatenated with settings derived from the `response-policy` statement.

Note: the `dnhrsps-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 `dnzrpzd-conf`, `dnzrpzd-sock`, and `dnzrpzd-args` (for details of these options, see the librpz documentation). Other RPZ configuration settings could be included in `dnhrsps-options` as well, but if `named` were switched back to traditional RPZ by setting `dnhrsps-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:

```plaintext
response-policy { zone "badlist"; };
```

and this zone statement:

```plaintext
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

; 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
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.
```

Response policy zones can be configured to set an Extended DNS Error (EDE) code on the responses which have been modified by the response policy:

```plaintext
response-policy { zone "badlist" ede filtered; };
```

The following settings are supported for the ede option:

```
none
```

No Extended DNS Error code is set (default).
forged
Extended DNS Error code 4 - Forged Answer.

blocked
Extended DNS Error code 15 - Blocked.

censored
Extended DNS Error code 16 - Censored.

filtered
Extended DNS Error code 17 - Filtered.

prohibited
Extended DNS Error code 18 - Prohibited.

See RFC 8914 for more information about the Extended DNS Error codes.

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

rate-limit

Grammar:

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>;
}

(continues on next page)
Controls excessive UDP responses, to prevent BIND 9 from being used to amplify reflection denial-of-service (DoS) attacks.

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.

```
window <integer>;
```

**Blocks:** options, view

**Tags:** query

Specifies the length of time during which responses are tracked.

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.

```
ipv4-prefix-length <integer>;
```

**Grammar:** ipv4-prefix-length <integer>;

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** server

Specifies the prefix lengths of IPv4 address blocks.

```
ipv6-prefix-length <integer>;
```

**Grammar:** ipv6-prefix-length <integer>;

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** server
Specifies the prefix lengths of IPv6 address blocks.

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 \texttt{ipv4-prefix-length} (default 24) and \texttt{ipv6-prefix-length} (default 56).

\textbf{responses-per-second}

\textbf{Grammar:} responses-per-second <integer>;

\textbf{Blocks:} options.rate-limit, view.rate-limit

\textbf{Tags:} query

Limits the number of non-empty responses for a valid domain name and record type.

All non-empty responses for a valid domain name (qname) and record type (qtype) are identical and have a limit specified with \texttt{responses-per-second} (default 0 or no limit). All valid wildcard domain names are interpreted as the zone's origin name concatenated to the "*" name.

\textbf{nodata-per-second}

\textbf{Grammar:} nodata-per-second <integer>;

\textbf{Blocks:} options.rate-limit, view.rate-limit

\textbf{Tags:} query

Limits the number of empty (NODATA) responses for a valid domain name.

All empty (NODATA) responses for a valid domain, regardless of query type, are identical. Responses in the NODATA class are limited by \texttt{nodata-per-second} (default \texttt{responses-per-second}).

\textbf{nxdomains-per-second}

\textbf{Grammar:} nxdomains-per-second <integer>;

\textbf{Blocks:} options.rate-limit, view.rate-limit

\textbf{Tags:} query

Limits the number of undefined subdomains for a valid domain name.

Requests for any and all undefined subdomains of a given valid domain result in NXDOMAIN errors, and are identical regardless of query type. They are limited by \texttt{nxdomains-per-second} (default \texttt{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.

\textbf{referrals-per-second}

\textbf{Grammar:} referrals-per-second <integer>;

\textbf{Blocks:} options.rate-limit, view.rate-limit

\textbf{Tags:} query

Limits the number of referrals or delegations to a server for a given domain.

Referrals or delegations to the server of a given domain are identical and are limited by \texttt{referrals-per-second} (default \texttt{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.

**errors-per-second**

**Grammar:** errors-per-second <integer>;

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** server

Limits the number of errors for a valid domain name and record type.

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.

**slip**

**Grammar:** slip <integer>;

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** query

Sets the number of “slipped” responses to minimize the use of forged source addresses for an attack.

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.)

**qps-scale**

**Grammar:** qps-scale <integer>;

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** query
Tightens defenses during DNS attacks by scaling back the ratio of the current query-per-second rate.

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.

**exempt-clients**

**Grammar:**

```
exempt-clients { <address_match_element>; ... }
```

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** query

Exempts specific clients or client groups from rate limiting.

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.

**all-per-second**

**Grammar:**

```
all-per-second <integer>
```

**Blocks:** options.rate-limit, view.rate-limit

**Tags:** query

Limits UDP responses of all kinds.

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.

**max-table-size**

**Grammar:**

```
max-table-size <integer>
```

**Blocks:** options.rate-limit, view.rate-limit

8.2. Blocks
Tags: server

Sets the maximum size of the table used to track requests and rate-limit responses.

**min-table-size**

*Grammar:* `min-table-size <integer>`;

*Blocks:* options.rate-limit, view.rate-limit

*Tags:* query

Sets the minimum size of the table used to track requests and rate-limit responses.

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.

**log-only**

*Grammar:* `log-only <boolean>`;

*Blocks:* options.rate-limit, view.rate-limit

*Tags:* query, logging

Tests rate-limiting parameters without actually dropping any requests.

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
- Redirect namespace

With either method, when `named` gets an NXDOMAIN response it examines a separate namespace to see if the NXDOMAIN 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.

**nxdomain-redirect**

*Grammar:* `nxdomain-redirect <string>`;

*Blocks:* options, view

*Tags:* query
Appends the specified suffix to the original query name, when replacing an NXDOMAIN with a redirect namespace.

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.

### 8.2.17 server Block Grammar

**server**

**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> | * };  
notify-source-v6 { <ipv6_address> | * };  
padding <integer>;  
provide-ixfr <boolean>;  
query-source [ address ] { <ipv4_address> | * };  
query-source-v6 [ address ] { <ipv6_address> | * };  
request-expire <boolean>;  
request-ixfr <boolean>;  
request-nsid <boolean>;  
require-cookie <boolean>;  
send-cookie <boolean>;  
tcp-keepalive <boolean>;  
tcp-only <boolean>;  
transfer-format ( many-answers | one-answer );  
transfer-source { <ipv4_address> | * };  
transfer-source-v6 { <ipv6_address> | * };  
transfers <integer>;  
}; // may occur multiple times
```

**Blocks:** topmost, view

**Tags:** server

Defines characteristics to be associated with a remote name server.
8.2.18 server Block 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.

bogus

Grammar: `bogus <boolean>`;
Blocks: `server, view.server`
Tags: `server`

Allows a remote server to be ignored.
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`.

edns

Grammar: `edns <boolean>`;
Blocks: `server, view.server`
Tags: `server`

Controls the use of the EDNS0 (RFC 2671) feature.
The `edns` clause determines whether the local server attempts to use EDNS when communicating with the remote server. The default is `yes`.

edns-version

Grammar: `edns-version <integer>`;
Blocks: `server, view.server`
Tags: `server`

Sets the maximum EDNS VERSION that is sent to the server(s) by the resolver.
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.

padding

Grammar: `padding <integer>`;
Blocks: `server, view.server`
Tags: `server`

Adds EDNS Padding options to outgoing messages to increase the packet size.
The 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.

**tcp-only**

*Grammar:* tcp-only <boolean>;

*Blocks:* server, view.server

*Tags:* server

Sets the transport protocol to TCP.

The 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.

**tcp-keepalive**

*Grammar:* tcp-keepalive <boolean>;

*Blocks:* server, view.server

*Tags:* server

Adds EDNS TCP keepalive to messages sent over TCP.

The option adds EDNS TCP keepalive to messages sent over TCP. Note that currently idle timeouts in responses are ignored.

**transfers**

*Grammar:* transfers <integer>;

*Blocks:* server, view.server

*Tags:* server

Limits the number of concurrent inbound zone transfers from a server.

*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.

**keys**

*Blocks:* dnssec-policy, server, view.server

*Tags:* server, security

Specifies one or more *server_keys* to be used with a remote server.

**Warning:** Not to be confused with *keys* in *dnssec-policy* specification. Although statements with the same name exist in both contexts, they refer to fundamentally incompatible concepts.

In the context of a *server* block, the option identifies a *server_key* 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.

It is possible to override the following values defined in `view` and `options` blocks:

- `edns-udp-size`
- `max-udp-size`
- `notify-source-v6`
- `notify-source`
- `provide-ixfr`
- `query-source-v6`
- `query-source`
- `request-expire`
- `request-ixfr`
- `request-nsid`
- `require-cookie`
- `send-cookie`
- `transfer-format`
- `transfer-source-v6`
- `transfer-source`

### 8.2.19 statistics–channels Block Grammar

**statistics–channels**

**Grammar:**

```
statistics–channels (   inet ( <ipv4_address> | <ipv6_address> | * ) [ port ( <integer> | * ) ] [–→allow { <address_match_element>; ... } ]; // may occur multiple times ]; // may occur multiple times
```

**Blocks:** topmost

**Tags:** logging

Specifies the communication channels to be used by system administrators to access statistics information on the name server.
8.2.20 statistics-channels Block 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 **port** on the specified **ip_address**, which can be an IPv4 or IPv6 address. An **ip_address** 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_address** of ::.

If no port is specified, port 80 is used for HTTP channels. The asterisk (*) cannot be used for **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, the source of connection requests must be restricted appropriately so that only trusted parties can access the statistics channel.

Gathering data exposed by the statistics channel locks various subsystems in **named**, which could slow down query processing if statistics data is requested too often.

An issue in the statistics channel would be considered a security issue only if it could be exploited by unprivileged users circumventing the access control list. In other words, any issue in the statistics channel that could be used to access information unavailable otherwise, or to crash **named**, is not considered a security issue if it can be avoided through the use of a secure configuration.

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.


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/xfrins (incoming zone transfer statistics), and http://127.0.0.1:8888/json/v1/net (network status and socket statistics), and http://127.0.0.1:8888/json/v1/mem (memory manager statistics), and http://127.0.0.1:8888/json/v1/traffic (traffic sizes).
8.2.21 tls Block Grammar

tls

Grammar:

```
tls <string> {  
    ca-file <quoted_string>;  
    cert-file <quoted_string>;  
    cipher-suites <string>;  
    ciphers <string>;  
    dhparam-file <quoted_string>;  
    key-file <quoted_string>;  
    prefer-server-ciphers <boolean>;  
    protocols { <string>; ... };  
    remote-hostname <quoted_string>;  
    session-tickets <boolean>;  
} // may occur multiple times
```

Blocks: topmost
Tags: security

Configures a TLS connection.

8.2.22 tls Block Definition and Usage

The `tls` statement is used to configure a TLS connection; this configuration can then be referenced by a `listen-on` or `listen-on-v6` statement to cause `named` to listen for incoming requests via TLS, or in the `primaries` statement for a zone of `type secondary` to cause zone transfer requests to be sent via TLS.

`tls` can only be set at the top level of `named.conf`.

The following options can be specified in a `tls` statement:

**key-file**

Grammar: `key-file <quoted_string> ;`

Blocks: tls
Tags: server, security

Specifies the path to a file containing the private TLS key for a connection.

Path to a file containing the private TLS key to be used for the connection.

**cert-file**

Grammar: `cert-file <quoted_string> ;`

Blocks: tls
Tags: server, security

Specifies the path to a file containing the TLS certificate for a connection.
Path to a file containing the TLS certificate to be used for the connection.

**ca-file**

**Grammar:** ca-file <quoted_string>;

**Blocks:** tls

**Tags:** server, security

Specifies the path to a file containing TLS certificates for trusted CA authorities, used to verify remote peer certificates.

Path to a file containing trusted CA authorities' TLS certificates used to verify remote peer certificates. Specifying this option enables remote peer certificates' verification. For incoming connections, specifying this option makes BIND require a valid TLS certificate from a client. In the case of outgoing connections, if `remote-hostname` is not specified, then the remote server IP address is used instead.

**dhparam-file**

**Grammar:** dhparam-file <quoted_string>;

**Blocks:** tls

**Tags:** server, security

Specifies the path to a file containing Diffie-Hellman parameters, for enabling cipher suites.

Path to a file containing Diffie-Hellman parameters, which is needed to enable the cipher suites depending on the Diffie-Hellman ephemeral key exchange (DHE). Having these parameters specified is essential for enabling perfect forward secrecy capable ciphers in TLSv1.2.

**remote-hostname**

**Grammar:** remote-hostname <quoted_string>;

**Blocks:** tls

**Tags:** security

Specifies the expected hostname in the TLS certificate of the remote server.

The expected hostname in the TLS certificate of the remote server. This option enables a remote server certificate verification. If `ca-file` is not specified, then the platform-specific certificates store is used for verification. This option is used when connecting to a remote peer only and, thus, is ignored when `tls` statements are referenced by `listen-on` or `listen-on-v6` statements.

**protocols**

**Grammar:** protocols { <string>; ... };

**Blocks:** tls

**Tags:** security

Specifies the allowed versions of the TLS protocol.
Allowed versions of the TLS protocol. TLS version 1.2 and higher are supported, depending on the cryptographic library in use. Multiple versions might be specified (e.g. protocols { TLSv1.2; TLSv1.3; });.

cipher-suites
   Grammar: cipher-suites <string>;
   Blocks: tls
   Tags: security

   Specifies a list of allowed cipher suites in the order of preference for TLSv1.3 only.

   Cipher suites list which defines allowed cipher suites, such as
   TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256.
   The string must be formed according to the rules specified in the OpenSSL documentation (see
   https://www.openssl.org/docs/man1.1.1/man1/ciphers.html, section “TLS v1.3 cipher suites” for
   details).

ciphers
   Grammar: ciphers <string>;
   Blocks: tls
   Tags: security

   Specifies a list of allowed ciphers in the order of preference for TLSv1.2 only.

   Cipher list which defines allowed ciphers, such as
   HIGH:!aNULL:!MD5:!SHA1:!SHA256:!SHA384. The string must be formed according to the rules specified in the OpenSSL documentation
   (see https://www.openssl.org/docs/man1.1.1/man1/ciphers.html for details).

prefer-server-ciphers
   Grammar: prefer-server-ciphers <boolean>;
   Blocks: tls
   Tags: server, security

   Specifies that server ciphers should be preferred over client ones.

   Specifies that server ciphers should be preferred over client ones.

session-tickets
   Grammar: session-tickets <boolean>;
   Blocks: tls
   Tags: security

   Enables or disables session resumption through TLS session tickets.
Enables or disables session resumption through TLS session tickets, as defined in RFC 5077. Disabling the stateless session tickets might be required in the cases when forward secrecy is needed, or the TLS certificate and key pair is planned to be used across multiple BIND instances.

**Warning:** TLS configuration is subject to change and incompatible changes might be introduced in the future. Users of TLS are encouraged to carefully read release notes when upgrading.

The options described above are used to control different aspects of TLS functioning. Thus, most of them have no well-defined default values, as these depend on the cryptographic library version in use and system-wide cryptographic policy. On the other hand, by specifying the needed options one could have a uniform configuration deployable across a range of platforms.

An example of privacy-oriented, perfect forward secrecy enabled configuration can be found below. It can be used as a starting point.

```plaintext
tls local-tls {
    key-file "/path/to/key.pem";
    cert-file "/path/to/fullchain_cert.pem";
    dhparam-file "/path/to/dhparam.pem";
    ciphers "HIGH:!kRSA:!aNULL:!eNULL:!RC4:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!SHA1:!SHA256:!SHA384";
    prefer-server-ciphers yes;
    session-tickets no;
};
```

A Diffie-Hellman parameters file can be generated using e.g. OpenSSL, like follows:

```plaintext
openssl dhparam -out /path/to/dhparam.pem <3072_or_4096>
```

Ensure that it gets generated on a machine with enough entropy from external sources (e.g. the computer you work on should be fine, the remote virtual machine or server might be not). These files do not contain any sensitive data and can be shared if required.

There are two built-in TLS connection configurations: ephemeral, uses a temporary key and certificate created for the current named session only, and none, which can be used when setting up an HTTP listener with no encryption.

The main motivation behind having the ephemeral configuration is to aid in testing, as trusted certificate authorities do not issue the certificates associated with this configuration. Thus, these certificates will never be trusted by any clients that verify TLS certificates. They provide encryption of the traffic but no authentication of the transmission channel. That might be enough in the case of deployment in a controlled environment.

It should be noted that on reconfiguration, the ephemeral TLS key and the certificate are recreated, and all TLS certificates and keys, as well as associated data, are reloaded from the disk. In that case, listening sockets associated with TLS remain intact.

Please keep in mind that doing reconfiguration can cause a short interruption in BIND’s ability to process inbound client packets. The length of interruption is environment and configuration-specific. A good example of when reconfiguration is necessary is when TLS keys and certificates are updated on the disk.

BIND supports the following TLS authentication mechanisms described in the RFC 9103, Section 9.3: Opportunistic TLS, Strict TLS, and Mutual TLS.

Opportunistic TLS provides encryption for data but does not provide any authentication for the channel. This mode is the default one and it is used whenever remote-hostname and ca-file options are not set in tls statements in use. RFC 9103 allows optional fallback to clear-text DNS in the cases when TLS is not available. Still, BIND intentionally does not support that in order to protect from unexpected data leaks due to misconfiguration. Both BIND and its complementary
tools either successfully establish a secure channel via TLS when instructed to do so or fail to establish a connection otherwise.

Strict TLS provides server authentication via a pre-configured hostname for outgoing connections. This mechanism offers both channel confidentiality and channel authentication (of the server). In order to achieve Strict TLS, one needs to use `remote-hostname` and, optionally, `ca-file` options in the `tls` statements used for establishing outgoing connections (e.g. the ones used to download zone from primaries via TLS). Providing any of the mentioned options will enable server authentication. If `remote-hostname` is provided but `ca-file` is missing, then the platform-specific certificate authority certificates are used for authentication. The set roughly corresponds to the one used by WEB-browsers to authenticate HTTPS hosts. On the other hand, if `ca-file` is provided but `remote-hostname` is missing, then the remote side’s IP address is used instead.

Mutual TLS is an extension to Strict TLS that provides channel confidentiality and mutual channel authentication. It builds up upon the clients offering client certificates when establishing connections and them doing the server authentication as in the case of Strict TLS. The server verifies the provided client certificates and accepts the TLS connection in case of successful verification or rejects it otherwise. In order to instruct the server to require and verify client TLS certificates, one needs to specify the `ca-file` option in `tls` configurations used to configure server listeners. The provided file must contain certificate authority certificates used to issue client certificates. In most cases, one should build one’s own TLS certificate authority specifically to issue client certificates and include the certificate authority certificate into the file.

For authenticating zone transfers over TLS, Mutual TLS might be considered a standalone solution, while Strict TLS paired with TSIG-based authentication and, optionally, IP-based access lists, might be considered acceptable for most practical purposes. Mutual TLS has the advantage of not requiring TSIG and thus, not having security issues related to shared cryptographic secrets.

### 8.2.23 http Block Grammar

**http**

Grammar:

```
http <string> {  
  endpoints { <quoted_string>; ... };  
  listener-clients <integer>;  
  streams-per-connection <integer>;  
}; // may occur multiple times
```

- **Blocks**: topmost
- **Tags**: query, server

Configures HTTP endpoints on which to listen for DNS-over-HTTPS (DoH) queries.

### 8.2.24 http Block Definition and Usage

The `http` statement is used to configure HTTP endpoints on which to listen for DNS-over-HTTPS (DoH) queries. This configuration can then be referenced by a `listen-on` or `listen-on-v6` statement to cause `named` to listen for incoming requests over HTTPS.

**http** can only be set at the top level of `named.conf`.

The following options can be specified in an `http` statement:
endpoints
Grammar: endpoints { <quoted_string>; ... }; 
Blocks: http 
Tags: query, server

Specifies a list of HTTP query paths on which to listen.

A list of HTTP query paths on which to listen. This is the portion of an RFC 3986-compliant URI following the hostname; it must be an absolute path, beginning with “/”. The default value is "/dns-query", if omitted.

listener-clients
Grammar: listener-clients <integer>; 
Blocks: http 
Tags: query, server

Specifies a per-listener quota for active connections.

The option specifies a per-listener quota for active connections.

streams-per-connection
Grammar: streams-per-connection <integer>; 
Blocks: http 
Tags: query, server

Specifies the maximum number of concurrent HTTP/2 streams over an HTTP/2 connection.

The option specifies the hard limit on the number of concurrent HTTP/2 streams over an HTTP/2 connection.

Any of the options above could be omitted. In such a case, a global value specified in the options statement is used (see http-listener-clients, http-streams-per-connection.

For example, the following configuration enables DNS-over-HTTPS queries on all local addresses:

```
http local {
    endpoints { "/dns-query"; }
};

options {
    listen-on tls ephemeral http local { any; }
    listen-on-v6 tls ephemeral http local { any; }
};
```
8.2.25 trust-anchors Block Grammar

**trust-anchors**

**Grammar:** trust-anchors { <string> ( static-key | initial-key | static-ds | initial-ds ) <integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times

**Blocks:** topmost, view

**Tags:** dnssec

Defines DNSSEC trust anchors.

8.2.26 trust-anchors Block 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 sets up an initial-key for the root
zone. This initializing key is built into named, and is current as of the release date. When the root zone key changes, a
running server will detect the change and roll to the new key, but newly-installed servers being run for the first time will need to be from a recent enough version of BIND to have been built with the current key.

8.2.27 dnssec-policy Block Grammar

dnssec-policy

Grammar options, view, zone (primary, secondary): dnssec-policy <string>;

Grammar topmost:

dnssec-policy <string> {
    cdnskey <boolean>;
    cds-digest-types { <string>; ... };
    dnskey-ttl <duration>;
    inline-signing <boolean>;
    keys { ( csk | ksk | zsk ) [ key-directory | key-store <string> ]...
    ←lifetime <duration_or_unlimited> algorithm <string> [ <integer> ]; ... };
    max-zone-ttl <duration>;
    nsec3param [ iterations <integer> ] [ optout <boolean> ] [ salt-length
    ←<integer> ];
    parent-ds-ttl <duration>;
}

(continues on next page)
parent-propagation-delay <duration>;
publish-safety <duration>;
purge-keys <duration>;
retire-safety <duration>;
signatures-jitter <duration>;
signatures-refresh <duration>;
signatures-validity <duration>;
signatures-validity-dnskey <duration>;
zone-propagation-delay <duration>;
}; // may occur multiple times

Blocks: topmost, options, view, zone (primary, secondary)

Tags: dnssec

Defines a key and signing policy (KASP) for zones.

8.2.28 dnssec-policy Block 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. Multiple key and signing policies can be configured with unique policy names.

A policy for a zone is selected using a dnssec-policy statement in the zone block, specifying the name of the policy that should be used.

There are three built-in policies:

- default, which uses the default policy,
- insecure, to be used when you want to gracefully unsigned your zone,
- none, which means no DNSSEC policy (the same as not selecting dnssec-policy at all; the zone is not signed.)

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.

The dnssec-policy statement requires dynamic DNS to be set up, or inline-signing to be enabled (which is the default for DNSSEC zones).

If inline-signing is enabled, this means that a signed version of the zone is maintained separately and is written out to a different file on disk (the zone’s filename plus a .signed extension).

If inline-signing is disabled, the zone needs to be configured with an update-policy or allow-update. In such case, the DNSSEC records are written to the filename set in the original zone’s file.

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.
Policy `default` causes the zone to be signed with a single combined-signing key (CSK) using algorithm ECD-SAP256SHA256; 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:

**cdnskey**

*Grammar:* `cdnskey <boolean>;`

*Blocks:* `dnssec-policy`

*Tags:* `dnssec`

Specifies whether a CDNSKEY record should be published during KSK rollover.

When set to the default value of `yes`, a CDNSKEY record is published during KSK rollovers when the DS of the successor key may be submitted to the parent.

**cds-digest-types**

*Grammar:* `cds-digest-types { <string>; ... };`

*Blocks:* `dnssec-policy`

*Tags:* `dnssec`

Specifies the digest types to use for CDS resource records.

This indicates the digest types to use when generating CDS resource records. The default is SHA-256 only.

**dnskey-ttl**

*Grammar:* `dnskey-ttl <duration>;`

*Blocks:* `dnssec-policy`

*Tags:* `dnssec`

Specifies the time to live (TTL) for DNSKEY resource records.

This indicates the TTL to use when generating DNSKEY resource records. The default is 1 hour (3600 seconds).
**inline-signing**

**tags**

dnssec

**short**

Specifies whether BIND 9 maintains a separate signed version of a zone.

If `yes`, BIND 9 maintains a separate signed version of the zone. An unsigned zone is transferred in or loaded from disk and the signed version of the zone is served with, possibly, a different serial number. The signed version of the zone is stored in a file that is the zone’s filename (set in `file`) with a `.signed` extension.

This behavior is enabled by default.

---

**Note:** When changing the `key-directory` or the `key-store`, BIND will be unable to find existing key files. Make sure you copy key files to the new directory before changing the path used in the configuration file. This is also true when changing to a built-in policy, for example to `insecure`. In this specific case you should move the existing key files to the zone’s `key-directory` from the new configuration.

---

**purge-keys**

**Grammar:** `purge-keys <duration>;

**Blocks:** dnssec-policy

**Tags:** dnssec

Specifies the amount of time after which DNSSEC keys that have been deleted from the zone can be removed from disk.

This is the amount of time after which 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**

**Grammar:** `publish-safety <duration>;

**Blocks:** dnssec-policy

**Tags:** dnssec

Increases the amount of time between when keys are published and when they become active, to allow for unforeseen events.

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**

**Grammar:** `retire-safety <duration>;

**Blocks:** dnssec-policy

**Tags:** dnssec
Increases the amount of time a key remains published after it is no longer active, to allow for unforeseen events.

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-jitter**

Grammar: signatures-jitter <duration>;

Blocks: dnssec-policy

Tags: dnssec

Specifies a range for signatures expirations.

To prevent all signatures from expiring at the same moment, BIND 9 may vary the validity interval of individual signatures. The validity of a newly generated signatures is in range between signatures-validity (maximum) and signatures-validity minus signatures-jitter (minimum). The default jitter is 12 hours and the configured value must be lower than signatures-validity and signatures-validity-dnskey.

**signatures-refresh**

Grammar: signatures-refresh <duration>;

Blocks: dnssec-policy

Tags: dnssec

Specifies how frequently an RRSIG record is refreshed.

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. The signatures-refresh value must be less than 90% of the minimum value of signatures-validity and signatures-validity-dnskey.

**signatures-validity**

Grammar: signatures-validity <duration>;

Blocks: dnssec-policy

Tags: dnssec

Indicates the validity period of an RRSIG record.

This indicates the validity period of an RRSIG record (subject to inception offset and jitter). The default is P2W (2 weeks).
**signatures-validity-dnskey**

*Grammar:* signatures-validity-dnskey <duration>;

*Blocks:* dnssec-policy

*Tags:* dnssec

Indicates the validity period of DNSKEY records.

This is similar to *signatures-validity*, but for DNSKEY records. The default is P2W (2 weeks).

**max-zone-ttl**

*tags*

zone, query

*short*

Specifies a maximum permissible time-to-live (TTL) value, in seconds.

This specifies the maximum permissible TTL value for the zone. When a zone file is loaded, any record encountered with a TTL higher than *max-zone-ttl* causes the zone to be rejected.

This ensures that when rolling to a new DNSKEY, the old key will 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 a known and predictable value.

The default value PT24H (24 hours). A value of zero is treated as if the default value were in use.

**nsec3param**

*Grammar:* nsec3param [ iterations <integer> ] [ optout <boolean> ] [ salt-length <integer> ];

*Blocks:* dnssec-policy

*Tags:* dnssec

Specifies the use of NSEC3 instead of NSEC, and sets NSEC3 parameters.

Use NSEC3 instead of NSEC, and optionally set the NSEC3 parameters.

Here is an example of an nsec3 configuration:

```
nsec3param iterations 0 optout no salt-length 0;
```

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 the specific salt string is not specified by the user; *named* creates a salt of the indicated length.

**Warning:** Do not use extra *iterations*, *salt*, and *opt-out* unless their implications are fully understood. A higher number of iterations causes interoperability problems and opens servers to CPU-exhausting DoS attacks. See RFC 9276.

**zone-propagation-delay**

*Grammar:* zone-propagation-delay <duration>;

*Blocks:* dnssec-policy
Sets the propagation delay from the time a zone is first updated to when the new version of the zone is served by all secondary servers.

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 \texttt{PT5M} (5 minutes).

\textbf{parent-ds-ttl}

\textbf{Grammar:} parent-ds-ttl <duration>;

\textbf{Blocks:} dnssec-policy

\textbf{Tags:} dnssec

Sets the time to live (TTL) of the DS RRset used by the parent zone.

This is the TTL of the DS RRset that the parent zone uses. The default is \texttt{P1D} (1 day).

\textbf{parent-propagation-delay}

\textbf{Grammar:} parent-propagation-delay <duration>;

\textbf{Blocks:} dnssec-policy

\textbf{Tags:} zone, dnssec

Sets the propagation delay from the time the parent zone is updated to when the new version is served by all of the parent zone’s name servers.

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 \texttt{PT1H} (1 hour).

\textbf{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.

\textbf{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 \textit{parental-agents}:

\textbf{checkds}

\textbf{Grammar:} checkds ( explicit | <boolean> );

\textbf{Blocks:} zone (primary, secondary)

\textbf{Tags:} dnssec

Controls whether DS queries are sent to parental agents.
If set to "yes", DS queries are sent when a KSK rollover is in progress. The queries are sent to the servers listed in the parent zone’s NS records. This is the default if there are no parental-agents configured for the zone.

If set to "explicit", DS queries are sent only to servers explicitly listed using parental-agents. This is the default if there are parental agents configured.

If set to "no", no DS queries are sent. Users should manually run `rndc dnssec -checkds` with the appropriate parameters to signal that specific DS records are published and/or withdrawn.

**parental-source**

**Grammar:** `parental-source ( <ipv4_address> | * );`

**Blocks:** options, view, zone (primary, secondary)

**Tags:** dnssec

Specifies which local IPv4 source address is used to send parental DS queries.

`parental-source` determines which local source address, and optionally UDP port, is used to send parental DS queries. 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:** port configuration is deprecated. A warning will be logged when this parameter is used.

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

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

**parental-source-v6**

**Grammar:** `parental-source-v6 ( <ipv6_address> | * );`

**Blocks:** options, view, zone (primary, secondary)

**Tags:** dnssec

Specifies which local IPv6 source address is used to send parental DS queries.

This option acts like `parental-source`, but applies to parental DS queries sent to IPv6 addresses.

### 8.2.29 managed-keys Block Grammar

**managed-keys**

**Warning:** This option is deprecated and will be removed in a future version of BIND.
Blocks: topmost, view
Tags: deprecated

8.2.30 managed-keys Block Definition and Usage

The managed-keys statement has been deprecated in favor of trust-anchors with the initial-key keyword.

8.2.31 trusted-keys Block Grammar

trusted-keys

Warning: This option is deprecated and will be removed in a future version of BIND.

Grammar: trusted-keys { <string> <integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times, deprecated
Blocks: topmost, view
Tags: deprecated

8.2.32 trusted-keys Block Definition and Usage

The trusted-keys statement has been deprecated in favor of trust-anchors with the static-key keyword.

8.2.33 view Block Grammar

view

Grammar:

view <string> [ <class> ] { allow-new-zones <boolean>; allow-notify { <address_match_element>; ... }; // experimental allow-proxy { <address_match_element>; ... }; // experimental allow-proxy-on { <address_match_element>; ... }; // experimental 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 [ port <integer> ] [ transport <string> ] { <address_match_element>; ... }; allow-update { <address_match_element>; ... }; allow-update-forwarding { <address_match_element>; ... }; also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] [ ipv4_address> ] [ key <string> ] [ tls <string> ] [ ... ]; attach-cache <string>; auth-nxdomain <boolean>; (continues on next page)
catalog-zones { zone <string> [ default-primaries [ port <integer> ] [...
source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { (...
<remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port...
<integer> ] [ key <string> ] [ tls <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 ); // may occur multiple times
    check-sibling <boolean>;
    check-spf ( warn | ignore );
    check-srv-cname ( fail | warn | ignore );
    check-svcb <boolean>;
    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> };
    // deprecated
    disable-algorithms <string> { <string>; ... }; // may occur multiple times
    disable-ds-digests <string> { <string>; ... }; // may occur multiple times
    disable-empty-zone <string>; // may occur multiple times
dlz <string> {
        database <string>;
        search <boolean>;
    } // may occur multiple times
dns64 <netprefix> {
    break-dnssec <boolean>;
    clients { <address_match_element>; ... };
    exclude { <address_match_element>; ... };
    mapped { <address_match_element>; ... };
    recursive-only <boolean>;
    suffix <ipv6_address>;
} // may occur multiple times
dns64-contact <string>;
dns64-server <string>;
dnskey-sig-validity <integer>; // obsolete
dnsrps-enable <boolean>;
dnsrps-options { <unspecified-text> };
dnssec-accept-expired <boolean>;
dnssec-dnskey-kskonly <boolean>; // obsolete
dnssec-loadkeys-interval <integer>;
dnssec-must-be-secure <string> <boolean>; // may occur multiple times,
... deprecated
dnssec-policy <string>;
dnssec-secure-to-insecure <boolean>; // obsolete
dnssec-update-mode ( maintain | no-resign ); // obsolete
dnssec-validation ( yes | no | auto );
dnstap { ( all | auth | client | forwarder | resolver | update ) [ (... query | response ) ]; ... };
    dual-stack-servers [ port <integer> ] { ( <quoted_string> [ port <integer>
... ] | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] )}; ..
...};

(continues on next page)
dyndb <string> <quoted_string> { <unspecified-text> }; // may occur
←multiple times
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> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
ipv4only-contact <string>;
ipv4only-enable <boolean>;
ipv4only-server <string>;
ixfr-from-differences ( primary | master | secondary | slave | <boolean> );
key <string> { algorithm <string>;
secret <string>;
}; // may occur multiple times
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>; ... }; // may occur
←multiple times, deprecated
masterfile-format ( 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-records-per-type <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-types-per-name <integer>;
max-udp-size <integer>;
max-validation-failures-per-fetch <integer>; // experimental
max-validations-per-fetch <integer>; // experimental
max-zone-ttl ( unlimited | <duration> ); // deprecated
message-compression <boolean>;
min-cache-ttl <duration>;
min-ncache-ttl <duration>;
(continues on next page)
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>;  
 notifies ( explicit | master-only | primary-only | <boolean> );  
notify-delay <integer>;  
notify-source { <ipv4_address> | * };  
notify-source-v6 { <ipv6_address> | * };  
notify-to-soa <boolean>;  
nsec3-test-zone <boolean>; // test only  
nta-lifetime <duration>;  
nta-recheck <duration>;  
nxdomain-redirect <string>;  
parental-source { <ipv4_address> | * };  
parental-source-v6 { <ipv6_address> | * };  
plugin ( query ) <string> [ { <unspecified-text> } ]; // may occur...  
multiple times  
preferred-glue <string>;  
prefetch <integer> [ <integer> ];  
provide-ixfr <boolean>;  
quname-minimization ( strict | relaxed | disabled | off );  
query-source [ address ] { <ipv4_address> | * };  
query-source-v6 [ address ] { <ipv6_address> | * };  
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-query-timeout <integer>;  
resolver-use-dns64 <boolean>;  
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 | passthr | tcp-only  
  quoted_string> ) ] [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [ ...  
  nsdname-enable <boolean> ] [ ede <string> ]; ... } [ add-soa <boolean> ] [ ...}
break-dnssec <boolean> [ max-policy-ttl <duration> ] [ min-update-interval <duration> ] [ min-ns-dots <integer> ] [ nsip-wait-recurse <boolean> ] [ nsip-enable <boolean> ] [ nsdname-enable <boolean> ] [ recursive-only <boolean> ] [ qname-wait-recurse <boolean> ] [ recursive-nsip-enable <boolean> ] [ nsdname-enable <boolean> ] [ dnsrps-enable <boolean> ] [ dnsrps-options { <unspecified-text> } ];
   root-key-sentinel <boolean>;
   rrsset-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> | * );
   notify-source-v6 ( <ipv6_address> | * );
   padding <integer>;
   provide-ixfr <boolean>;
   query-source [ address ] ( <ipv4_address> | * );
   query-source-v6 [ address ] ( <ipv6_address> | * );
   request-expire <boolean>;
   request-ixfr <boolean>;
   request-nsid <boolean>;
   require-cookie <boolean>;
   send-cookie <boolean>;
   tcp-keepalive <boolean>;
   tcp-only <boolean>;
   transfer-format ( many-answers | one-answer );
   transfer-source ( <ipv4_address> | * );
   transfer-source-v6 ( <ipv6_address> | * );
   transfers <integer>;
}; // may occur multiple times
servfail-ttl <duration>;
sig-signing-nodes <integer>;
sig-signing-signatures <integer>;
sig-signing-type <integer>;
sig-validity-interval <integer>; // obsolete
sortlist { <address_match_element>; ... }; // deprecated
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> | * );
transfer-source-v6 ( <ipv6_address> | * );
trust-anchor-telemetry <boolean>;
trust-anchors { <string> ( static-key | initial-key | static-ds | initial-ds ) <integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times
trusted-keys { <string> <integer> <integer> <integer> <quoted_string>; ... };
try-tcp-refresh <boolean>;

(continues on next page)
update-check-ksk <boolean>; // obsolete
v6-bias <integer>; 
validate-except { <string>; ... }; 
zero-no-soa-ttl <boolean>; 
zero-no-soa-ttl-cache <boolean>; 
zone <string> [ <class> ] { 
in-view <string>; 
}; 
zone <string> [ <class> ] { 
type forward; 
forward { first | only }; 
forwarders [ port <integer> ] [ tls <string> ] { { <ipv4_address>... … 
| <ipv6_address> } [ port <integer> ] [ tls <string> ]; ... }; 
}; 
zone <string> [ <class> ] { 
type hint; 
check-names { fail | warn | ignore }; 
file <quoted_string>; 
}; 
zone <string> [ <class> ] { 
type mirror; 
allow-notify { <address_match_element>; ... }; 
allow-query { <address_match_element>; ... }; 
allow-query-on { <address_match_element>; ... }; 
allow-transfer [ port <integer> ] [ transport <string> ] { 
|<address_match_element>; ... }; 
allow-update-forwarding { <address_match_element>; ... }; 
also-notify [ port <integer> ] [ source { ( <ipv4_address> | * ) ...  
|<source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> ...  
|<port <integer> ) | <ipv6_address> [ port <integer> ] [ key <string> ] [ tls  
|<string> ]; ... }; 
check-names { fail | warn | ignore }; 
database <string>; 
ixfr-from-differences <boolean>; 
journal <quoted_string>; 
masterfile-format { raw | text }; 
masterfile-style { full | relative }; 
max-ixfr-ratio { unlimited | <percentage> }; 
max-journal-size { default | unlimited | <sizeval> }; 
max-records <integer>; 
max-records-per-type <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-types-per-name <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> | * }; 
notify-source-v6 { <ipv6_address> | * }; 
primaries [ port <integer> ] [ source { ( <ipv4_address> | * ) ... ] [...
source-v6 { <ipv6_address> | * } { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ] ]; ...

request-expire <boolean>; request-ixfr <boolean>; transfer-source ( <ipv4_address> | * ); transfer-source-v6 ( <ipv6_address> | * ); try-tcp-refresh <boolean>; zero-no-soa-ttl <boolean>; zone-statistics ( full | terse | none | <boolean> );

zone <string> [ <class> ] { type primary; allow-query { <address_match_element>; ... }; allow-query-on { <address_match_element>; ... }; allow-transfer [ port <integer> ] [ transport <string> ] {<address_match_element>; ... }; allow-update { <address_match_element>; ... }; also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] { source-v6 ( <ipv6_address> | * ) } { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };... } 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-svcb <boolean>; check-wildcard <boolean>; checks ( explicit | <boolean> ); database <string>; dialup ( notify | notify-passive | passive | refresh | <boolean> );

dlz <string>; dnskey-sig-validity <integer>; // obsolete dnsssec-dnskey-kskonly <boolean>; // obsolete dnsssec-loadkeys-interval <integer>; dnsssec-policy <string>; dnsssec-secure-to-insecure <boolean>; // obsolete dnsssec-update-mode ( maintain | no-resign ); // obsolete file <quoted_string>; forward ( first | only ); forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... }; inline-signing <boolean>; ixfr-from-differences <boolean>; journal <quoted_string>; key-directory <quoted_string>; max-ixfr-ratio ( unlimited | <percentage> ); max-journal-size ( default | unlimited | <sizeval> ); max-records <integer>; max-records-per-type <integer>;
max-transfer-idle-out <integer>;
max-transfer-time-out <integer>;
max-types-per-name <integer>;
max-zone-ttl ( unlimited | <duration> ); // deprecated
notify ( explicit | master-only | primary-only | <boolean> );
notify-delay <integer>;
notify-source ( <ipv4_address> | * );
notify-source-v6 ( <ipv6_address> | * );
notify-to-soa <boolean>;
nsec3-test-zone <boolean>; // test only
parental-agents [ port <integer> ] [ source ( <ipv4_address> | * _
→ ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address>_.
→{ port <integer> } | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls
←<string> ] ; ... };
parental-source ( <ipv4_address> | * );
parental-source-v6 ( <ipv6_address> | * );
serial-update-method ( date | increment | unixtime );
sig-signing-nodes <integer>;
sig-signing-signatures <integer>;
sig-signing-type <integer>;
sig-validity-interval <integer> [ <integer> ]; // obsolete
update-check-ksk <boolean>; // obsolete
update-policy ( local | { ( deny | grant ) <string> ( 6to4-self |_
→external | krb5-self | krb5-selfsub | krb5-subdomain | krb5-subdomain-self-rhs_
→| ms-self | ms-selfsub | ms-subdomain | ms-subdomain-self-rhs | 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 redirect;
 allow-query { <address_match_element>; ... };
 allow-query-on { <address_match_element>; ... };
 dlz <string>;
 file <quoted_string>;
 masterfile-format ( raw | text );
 masterfile-style ( full | relative );
 max-records <integer>;
 max-records-per-type <integer>;
 max-types-per-name <integer>;
 max-zone-ttl ( unlimited | <duration> ); // deprecated
 primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ _
→source-v6 { <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port
←<integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls
←<string> ] ; ... };
 zone-statistics ( full | terse | none | <boolean> );
};
zone <string> [ <class> ] { type secondary;
 allow-query { <address_match_element>; ... };
 allow-query-on { <address_match_element>; ... };
 allow-transfer [ port <integer> ] [ transport <string> ] {_
←<address_match_element> ; ... };
 allow-update-forwarding { <address_match_element> ; ... };
 also-notify { port <integer> } [ source ( <ipv4_address> | * ) ]_
(continues on next page)
### 8.2. Blocks

```plaintext
| source-v6 ( <ipv6_address> | * ) | ( ( <remote-servers> | <ipv4_address> | port <integer> ) | <ipv6_address> | port <integer> ) | ( key <string> | [ tls <string> ] | ... ]; ... | ... };
check-names ( fail | warn | ignore );
checkds ( explicit | <boolean> );
database <string>;
dialup ( notify | notify-passive | passive | refresh | <boolean> )
}; // deprecated
dlz <string>;
dmskey-sig-validity <integer>; // obsolete
dmssec-dnskey-kskonly <boolean>; // obsolete
dmssec-loadkeys-interval <integer>;
dmssec-policy <string>;
dmssec-update-mode ( maintain | no-resign ); // obsolete
dnskey-sig-validity <integer>; // obsolete
dnssec-dnskey-kskonly <boolean>; // obsolete
dnssec-loadkeys-interval <integer>;
dnssec-policy <string>;
dnssec-update-mode ( maintain | no-resign ); // obsolete
file <quoted_string>;
forward ( first | only );
forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> | port <integer> ) [ tls <string> ]; ... }
inline-signing <boolean>;
ixfr-from-differences <boolean>;
journal <quoted_string>;
key-directory <quoted_string>;
masterfile-format ( raw | text );
masterfile-style ( full | relative );
max-ixfr-ratio ( unlimited | <percentage> );
max-journal-size ( default | unlimited | <sizeval> );
max-records <integer>;
max-records-per-type <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-types-per-name <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> | * );
notify-source-v6 ( <ipv6_address> | * );
notify-to-soa <boolean>;
nsec3-test-zone <boolean>; // test only
parental-agents [ port <integer> ] [ source ( <ipv4_address> | * ) | source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] | ( key <string> | [ tls <string> ] | ... };
parental-source ( <ipv4_address> | * );
parental-source-v6 ( <ipv6_address> | * );
primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] | ( key <string> | [ tls <string> ] | ... };
request-expire <boolean>;
request-ixfr <boolean>;
sig-signing-nodes <integer>;
```
sig-signing-signatures <integer>;
sig-signing-type <integer>;
sig-validity-interval <integer> [ <integer> ]; // obsolete
transfer-source ( <ipv4_address> | * ); // obsolete
transfer-source-v6 ( <ipv6_address> | * );
try-tcp-refresh <boolean>;
update-check-ksk <boolean>; // obsolete
zero-no-soa-ttl <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> ] [ tls <string> ] { ( <ipv4_address>... 
  transfer-source ( <ipv4_address> | * ); // obsolete
  transfer-source-v6 ( <ipv6_address> | * );
  try-tcp-refresh <boolean>;
  update-check-ksk <boolean>; // obsolete
  zero-no-soa-ttl <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> ] [ tls <string> ] { ( <ipv4_address>... 
  transfer-source ( <ipv4_address> | * ); // obsolete
  transfer-source-v6 ( <ipv6_address> | * );
  try-tcp-refresh <boolean>;
  update-check-ksk <boolean>; // obsolete
  zero-no-soa-ttl <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> ] [ tls <string> ] { ( <ipv4_address>... 
  transfer-source ( <ipv4_address> | * ); // obsolete
  transfer-source-v6 ( <ipv6_address> | * );
  try-tcp-refresh <boolean>;
  update-check-ksk <boolean>; // obsolete
  zero-no-soa-ttl <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> ] [ tls <string> ] { ( <ipv4_address>... 
  transfer-source ( <ipv4_address> | * ); // obsolete
  transfer-source-v6 ( <ipv6_address> | * );
  try-tcp-refresh <boolean>;
  update-check-ksk <boolean>; // obsolete
  zero-no-soa-ttl <boolean>;
  zone-statistics ( full | terse | none | <boolean> );
})
zone-statistics ( full | terse | none | <boolean> );
};  // may occur multiple times

Blocks: topmost
Tags: view

Allows a name server to answer a DNS query differently depending on who is asking.

8.2.34 view Block 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.

**match-clients**

Grammar: match-clients { <address_match_element>; ... };

Blocks: view
Tags: view

Specifies a view of DNS namespace for a given subset of client IP addresses.

**match-destinations**

Grammar: match-destinations { <address_match_element>; ... };

Blocks: view
Tags: view

Specifies a view of DNS namespace for a given subset of destination IP addresses.

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 the name of a TSIG key, which provides a mechanism for the client to select the view.

**match-recursive-only**

Grammar: match-recursive-only <boolean>;

Blocks: view
Tags: view

8.2. Blocks
Specifies that only recursive requests can match this view of the DNS namespace.

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:

```plaintext
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";
    }
};
```
8.2.35 zone Block Grammar

zone
  Blocks: topmost, view
  Tags: zone

Specifies the zone in a BIND 9 configuration.

8.2.36 zone Block Definition and Usage

Zone Types

type
  Blocks: zone (forward, hint, mirror, primary, redirect, secondary, static-stub, stub)
  Tags: zone

Specifies the kind of zone in a given configuration.

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, or redirect.

type primary

Grammar:

(continues on next page)
contains the main copy of the data for a zone.

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.

**type secondary**

**Grammar:**

```plaintext
zone <string> [ <class> ] {
    type secondary;
    allow-notify { <address_match_element>; ... };
    allow-query { <address_match_element>; ... };
    allow-query-on { <address_match_element>; ... };
    allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_element>; ... };
    allow-update-forwarding { <address_match_element>; ... };
    also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] [ <ipv6_address> ] [ port <integer> ] ) [ key <string> ] [ tls <string> ] [ <……
```
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 100,000 files in a single directory.)

**type mirror**

**Grammar:**

```
zone <string> [ <class> ] { 
  type mirror; 
  allow-notify { <address_match_element>; ... }; 
  allow-query { <address_match_element>; ... }; 
  allow-query-on { <address_match_element>; ... }; 
  allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_element>; ... }; 
}; 
```
(continued from previous page)

```
allow-update-forwarding { <address_match_element>; ... );
also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-
v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port
<-<integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls
<-<string> ] ]; ... );
    check-names ( fail | warn | ignore );
database <string>;
file <quoted_string>;
ixfr-from-differences <boolean>;
journal <quoted_string>;
masterfile-format ( raw | text );
masterfile-style ( full | relative );
max-ixfr-ratio ( unlimited | <percentage> );
max-journal-size ( default | unlimited | <sizeval> );
max-records <integer>;
max-records-per-type <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-types-per-name <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> | * );
notify-source-v6 ( <ipv6_address> | * );
    primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-
v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port
<-<integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls
<-<string> ] ]; ... );
    request-expire <boolean>;
request-ixfr <boolean>;
transfer-source ( <ipv4_address> | * );
transfer-source-v6 ( <ipv6_address> | * );
    try-tcp-refresh <boolean>;
zero-no-soa-ttl <boolean>;
zzone-statistics ( full | terse | none | <boolean> );
```

**Blocks:** zone, view.zone

**Tags:** zone

Contains a DNSSEC-validated duplicate of the main data for a zone.

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.

---

8.2. Blocks  239
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 for details), and a key-signing key (KSK) for the specified zone to be explicitly configured as a trust anchor (see trust-anchors).

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.

### type hint

**Grammar:**

```
zeone <string> [ <class> ] {  
    type hint;  
    check-names { fail | warn | ignore }; 
    file <quoted_string>; 
};
```

**Blocks:** zone, view.zone  
**Tags:** zone  

Contains the initial set of root name servers to be used at BIND 9 startup.

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.
Blocks: zone, view.zone

Tags: zone

Contains a duplicate of the NS records of a primary zone.

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.

type static-stub

Grammar:
Blocks: zone, view.zone

Tags: zone

Contains a duplicate of the NS records of a primary zone, but statically configured rather than transferred from a primary server.

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 anrcode of NOTAUTH.

Each static-stub zone is configured with internally generated NS and (if necessary) glue A or AAAA RRs.

**type forward**

Grammar:

```
zone <string> [ <class> ] {
  type forward;
  forward ( first | only );
  forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
};
```

Blocks: zone, view.zone

Tags: zone

Contains forwarding statements that apply to queries within a given domain.

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.

**type redirect**

**Grammar:**

```markdown
zone <string> [ <class> ] {  
  type redirect;
  allow-query { <address_match_element>; ... };  
  allow-query-on { <address_match_element>; ... };  
  dlz <string>;
  file <quoted_string>;
  masterfile-format ( raw | text );
  masterfile-style ( full | relative );
  max-records <integer>;
  max-records-per-type <integer>;
  max-types-per-name <integer>;
  max-zone-ttl ( unlimited | <duration> ); // deprecated
  primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-\~<ipv6_address> | * ] [ ( <remote-servers> | <ipv4_address> [ port \~<integer> ] | <ipv6_address> [ port <integer> ] ) ] [ key <string> ] [ tls \~<string> ] ... ;
  zone-statistics ( full | terse | none | <boolean> );
};
```

**Blocks:** zone, view.zone

**Tags:** zone

Contains information to answer queries when normal resolution would return NXDOMAIN.

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::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
* IN AAAA 2001: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.

---

8.2. Blocks 243
in-view

Grammar zone, view.zone:

```
zone <string> [ <class> ] {
  in-view <string>;
};
```

Grammar zone (in-view): in-view <string>;

Blocks: zone, zone (in-view), view.zone

Tags: zone, view

Specifies the view in which a given zone is defined.

When using multiple views, a `type primary` or `type 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`. 

244 Chapter 8. Configuration Reference
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.

dnssec-update-mode

See the description of dnssec-update-mode in options.

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

Grammar: database <string>;

Blocks: dlz, zone (mirror, primary, secondary, stub), view.dlz

Tags: zone

Specifies the type of database to be used to store zone data.

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*.

**file**

Grammar logging.channel: `file <quoted_string> [ versions ( unlimited | <integer> ) ] [ size <size> ] [ suffix ( increment | timestamp ) ];

Grammar zone (hint, mirror, primary, redirect, secondary, stub): `file <quoted_string>;

**Blocks:** zone (hint, mirror, primary, redirect, secondary, stub), logging.channel

**Tags:** zone

Specifies the zone’s filename.

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**

Grammar: `journal <quoted_string>;

**Blocks:** zone (mirror, primary, secondary)

**Tags:** zone

Allows the default journal’s filename to be overridden.

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*.

**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*.

**server-addresses**

**Grammar:**
server-addresses { ( <ipv4_address> | <ipv6_address> ); ... };

**Blocks:**
zone (static-stub)

**Tags:** query, zone

Specifies a list of IP addresses to which queries should be sent in recursive resolution for a static-stub zone.

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**

**Grammar:**
server-names { <string>; ... };

**Blocks:**
zone (static-stub)

**Tags:** zone

Specifies a list of domain names of name servers that act as authoritative servers of a static-stub zone.

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*.

**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*.

**serial-update-method**  
See the description of *serial-update-method* in *options*.

**inline-signing**  
**Grammar:** inline-signing <boolean>;

**Blocks:** dnssec-policy, zone (primary, secondary)

**Tags:** zone, dnssec

Specifies whether BIND 9 maintains a separate signed version of a zone.

The use of inline signing is determined by the *dnssec-policy* for the zone. If *inline-signing* is explicitly set to *yes* or *no* in *zone*, then it overrides any value from *dnssec-policy*.

**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. The use of this option in zone blocks is deprecated and will be rendered nonoperational in a future release.

Dynamic Update Policies

BIND 9 supports two methods of granting clients the right to perform dynamic updates to a zone:

- allow-update - a simple access control list
- update-policy - fine-grained access control

In both cases, BIND 9 writes the updates to the zone’s filename set in file.

In the case of a DNSSEC zone where inline-signing is disabled, DNSSEC records are also written to the zone’s filename.

Note: The zone file can no longer be manually updated while named is running; it is now necessary to perform rndc freeze, edit, and then perform rndc thaw. Comments and formatting in the zone file are lost when dynamic updates occur.

update-policy

Grammar: update-policy ( local | { ( deny | grant ) <string> ( 6to4-self | external | krb5-self | krb5-selfsub | krb5-subdomain | krb5-subdomain-self-rhs | ms-self | ms-selfsub | ms-subdomain | ms-subdomain-self-rhs | name | self | selfsub | selfwild | subdomain | tcp-self | wildcard | zonesub ) [ <string> ] <rrtypelist>; ... } );

Blocks: zone (primary)

Tags: transfer

Sets fine-grained rules to allow or deny dynamic updates (DDNS), based on requester identity, updated content, etc.

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 /run/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 16 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.

If the type is immediately followed by a number in parentheses, that number is the maximum number of records of that type permitted to exist in the RRset after an update has been applied. For example, PTR(1) indicates that only one PTR record is allowed. If an attempt is made to add two PTR records in an update, the second one is silently discarded. If a PTR record already exists, both new records are silently discarded.

If type ANY is specified with a limit, then that limit applies to all types that are not otherwise specified. For example, A PTR(1) ANY(2) indicates that an unlimited number of A records can exist, but only one PTR record, and no more than two of any other type.

Typical use with a rule `grant * tcp-self . PTR(1)`; in the zone 2.0.192.IN-ADDR.ARPA looks like this:

```
nsupdate -v <<EOF
local 192.0.2.1
del 1.2.0.192.IN-ADDR.ARPA PTR
add 1.2.0.192.IN-ADDR.ARPA 0 PTR EXAMPLE.COM
send
EOF
```

The `ruletype` field has 18 values: name, subdomain, zonesub, wildcard, self, selfsub, selfwild, ms-self, ms-selfsub, ms-subdomain, ms-subdomain-self-rhs, krb5-self, krb5-selfsub, krb5-subdomain, krb5-subdomain-self-rhs, 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.
**ms-subdomain-self-rhs**
This rule is similar to ms-subdomain, with an additional restriction that PTR and SRV target names must match the name of the machine identified in the principal.

**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 AAAAA 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.

**krb5-subdomain-self-rhs**
This rule is similar to krb5-subdomain, with an additional restriction that PTR and SRV target names must match the name of the machine identified in the principal.

**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.

**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.

**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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol version number</td>
<td>4 bytes, network byte order, currently 1</td>
</tr>
<tr>
<td>Request length</td>
<td>4 bytes, network byte order</td>
</tr>
<tr>
<td>Signer</td>
<td>null-terminated string</td>
</tr>
<tr>
<td>Name</td>
<td>null-terminated string</td>
</tr>
<tr>
<td>TCP source address</td>
<td>null-terminated string</td>
</tr>
<tr>
<td>Rdata type</td>
<td>null-terminated string</td>
</tr>
<tr>
<td>Key</td>
<td>null-terminated string</td>
</tr>
<tr>
<td>TKEY token length</td>
<td>4 bytes, network byte order</td>
</tr>
<tr>
<td>TKEY token</td>
<td>remainder of packet</td>
</tr>
</tbody>
</table>

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.

**Warning:** The external daemon must not delay communication. This policy is evaluated synchronously; any wait period negatively affects *named* performance.

### 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:

```bind
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.
8.3 Statements

BIND 9 supports many hundreds of statements; finding the right statement to control a specific behavior or solve a particular problem can be a daunting task. To simplify the task for users, all statements have been assigned one or more tags. Tags are designed to group together statements that have broadly similar functionality; thus, for example, all statements that control the handling of queries or of zone transfers are respectively tagged under `query` and `transfer`.

*DNSSEC Tag Statements* are those that relate to or control DNSSEC.

*Logging Tag Statements* relate to or control logging, and typically only appear in a logging block.

*Query Tag Statements* relate to or control queries.

*Security Tag Statements* relate to or control security features.

*Server Tag Statements* relate to or control server behavior, and typically only appear in a server block.

*Transfer Tag Statements* relate to or control zone transfers.

*View Tag Statements* relate to or control view selection criteria, and typically only appear in a view block.

*Zone Tag Statements* relate to or control zone behavior, and typically only appear in a zone block.

*Deprecated Tag Statements* are those that are now deprecated, but are included here for historical reference.

The following table lists all statements permissible in `named.conf`, with their associated tags; the next section groups the statements by tag. Please note that these sections are a work in progress.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Assigns a symbolic name to an address match list.</td>
<td>server</td>
</tr>
<tr>
<td>algorithm</td>
<td>Defines the algorithm to be used in a key clause.</td>
<td>security</td>
</tr>
<tr>
<td>all-per-second</td>
<td>Limits UDP responses of all kinds.</td>
<td>query</td>
</tr>
<tr>
<td>allow-new-zones</td>
<td>Controls the ability to add zones at runtime via <code>rndc addzone</code>.</td>
<td>server, zone</td>
</tr>
<tr>
<td>allow-notify</td>
<td>Defines an <code>address_match_list</code> that is allowed to send NOTIFY messages for the zone, in addition to addresses defined in the <code>primaries</code> option for the zone.</td>
<td>transfer</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-proxy</td>
<td>Defines an address_match_list for the client addresses allowed to send PROXYv2 headers.</td>
<td>server</td>
</tr>
<tr>
<td>allow-proxy-on</td>
<td>Defines an address_match_list for the interface addresses allowed to accept PROXYv2 headers.</td>
<td>server</td>
</tr>
<tr>
<td>allow-query</td>
<td>Specifies which hosts (an IP address list) are allowed to send queries to this resolver.</td>
<td>query</td>
</tr>
<tr>
<td>allow-query-cache</td>
<td>Specifies which hosts (an IP address list) can access this server’s cache and thus effectively controls recursion.</td>
<td>query</td>
</tr>
<tr>
<td>allow-query-cache-on</td>
<td>Specifies which hosts (an IP address list) can access this server’s cache.</td>
<td>query</td>
</tr>
<tr>
<td>allow-query-on</td>
<td>Specifies which local addresses (an IP address list) are allowed to send queries to this resolver.</td>
<td>query</td>
</tr>
<tr>
<td>allow-recursion</td>
<td>Defines an address_match_list of clients that are allowed to perform recursive queries.</td>
<td>query</td>
</tr>
<tr>
<td>allow-recursion-on</td>
<td>Specifies which local addresses can accept recursive queries.</td>
<td>query, server</td>
</tr>
</tbody>
</table>

8.3. Statements
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-transfer</td>
<td>Defines an address_match_list of hosts that are allowed to transfer the zone information from this server.</td>
<td>transfer</td>
</tr>
<tr>
<td>allow-update</td>
<td>Defines an address_match_list of hosts that are allowed to submit dynamic updates for primary zones.</td>
<td>transfer</td>
</tr>
<tr>
<td>allow-update-forwarding</td>
<td>Defines an address_match_list of hosts that are allowed to submit dynamic updates to a secondary server for transmission to a primary.</td>
<td>transfer</td>
</tr>
<tr>
<td>also-notify</td>
<td>Defines one or more hosts that are sent NOTIFY messages when zone changes occur.</td>
<td>transfer</td>
</tr>
<tr>
<td>answer-cookie</td>
<td>Controls whether COOKIE EDNS replies are sent in response to client queries.</td>
<td>query</td>
</tr>
<tr>
<td>attach-cache</td>
<td>Allows multiple views to share a single cache database.</td>
<td>view</td>
</tr>
<tr>
<td>auth-nxdomain</td>
<td>Controls whether BIND, acting as a resolver, provides authoritative NXDOMAIN (domain does not exist) answers.</td>
<td>query</td>
</tr>
<tr>
<td>automatic-interface-scan</td>
<td>Controls the automatic rescanning of network interfaces when addresses are added or removed.</td>
<td>server</td>
</tr>
<tr>
<td>avoid-v4-udp-ports</td>
<td>Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv4 messages.</td>
<td>deprecated</td>
</tr>
</tbody>
</table>

continues on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>avoid-v6-udp-ports</strong></td>
<td>Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv6 messages.</td>
<td>deprecated</td>
</tr>
<tr>
<td><strong>bindkeys-file</strong></td>
<td>Specifies the pathname of a file to override the built-in trusted keys provided by named.</td>
<td>dnssec</td>
</tr>
<tr>
<td><strong>blackhole</strong></td>
<td>Defines an address_match_list of hosts to ignore. The server will neither respond to queries from nor send queries to these addresses.</td>
<td>query</td>
</tr>
<tr>
<td><strong>bogus</strong></td>
<td>Allows a remote server to be ignored.</td>
<td>server</td>
</tr>
<tr>
<td><strong>break-dnssec</strong></td>
<td>Enables dns64 synthesis even if the validated result would cause a DNSSEC validation failure.</td>
<td>query</td>
</tr>
<tr>
<td><strong>buffered</strong></td>
<td>Controls flushing of log messages.</td>
<td>logging</td>
</tr>
<tr>
<td><strong>ca-file</strong></td>
<td>Specifies the path to a file containing TLS certificates for trusted CA authorities, used to verify remote peer certificates.</td>
<td>server, security</td>
</tr>
<tr>
<td><strong>catalog-zones</strong></td>
<td>Configures catalog zones in named.conf.</td>
<td>zone</td>
</tr>
<tr>
<td><strong>category</strong></td>
<td>Specifies the type of data logged to a particular channel.</td>
<td>logging</td>
</tr>
<tr>
<td><strong>cdnskey</strong></td>
<td>Specifies whether a CDNSKEY record should be published during KSK rollover.</td>
<td>dnssec</td>
</tr>
</tbody>
</table>

*continues on next page*
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cds-digest-types</code></td>
<td>Specifies the digest types to use for CDS resource records.</td>
<td><code>dnssec</code></td>
</tr>
<tr>
<td><code>cert-file</code></td>
<td>Specifies the path to a file containing the TLS certificate for a connection.</td>
<td><code>server, security</code></td>
</tr>
<tr>
<td><code>channel</code></td>
<td>Defines a stream of data that can be independently logged.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>check-dup-records</code></td>
<td>Checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS.</td>
<td><code>query, dnssec</code></td>
</tr>
<tr>
<td><code>check-integrity</code></td>
<td>Performs post-load zone integrity checks on primary zones.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>check-mx</code></td>
<td>Checks whether an MX record appears to refer to an IP address.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>check-mx-cname</code></td>
<td>Sets the response to MX records that refer to CNAMEs.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>check-names</code></td>
<td>Restricts the character set and syntax of certain domain names in primary files and/or DNS responses received from the network.</td>
<td><code>query, server</code></td>
</tr>
<tr>
<td><code>check-sibling</code></td>
<td>Specifies whether to check for sibling glue when performing integrity checks.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>check-spf</code></td>
<td>Specifies whether to check for a TXT Sender Policy Framework record, if an SPF record is present.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>check-srv-cname</code></td>
<td>Sets the response to SRV records that refer to CNAMEs.</td>
<td><code>zone</code></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check-svcb</code></td>
<td>Specifies whether to perform additional checks on SVCB records.</td>
<td>zone</td>
</tr>
<tr>
<td><code>check-wildcard</code></td>
<td>Checks for non-terminal wildcards.</td>
<td>zone</td>
</tr>
<tr>
<td><code>checkds</code></td>
<td>Controls whether DS queries are sent to parental agents.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>cipher-suites</code></td>
<td>Specifies a list of allowed cipher suites in the order of preference for TLSv1.3 only.</td>
<td>security</td>
</tr>
<tr>
<td><code>ciphers</code></td>
<td>Specifies a list of allowed ciphers in the order of preference for TLSv1.2 only.</td>
<td>security</td>
</tr>
<tr>
<td><code>clients</code></td>
<td>Specifies an access control list (ACL) of clients that are affected by a given <code>dns64</code> directive.</td>
<td>query</td>
</tr>
<tr>
<td><code>clients-per-query</code></td>
<td>Sets the initial minimum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.</td>
<td>server</td>
</tr>
<tr>
<td><code>controls</code></td>
<td>Specifies control channels to be used to manage the name server.</td>
<td>server</td>
</tr>
<tr>
<td><code>cookie-algorithm</code></td>
<td>Sets the algorithm to be used when generating a server cookie.</td>
<td>server</td>
</tr>
<tr>
<td><code>cookie-secret</code></td>
<td>Specifies a shared secret used for generating and verifying EDNS COOKIE options within an anycast cluster.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>Specifies the type of database to be used to store zone data.</td>
<td>zone</td>
</tr>
<tr>
<td><strong>deny-answer-addresses</strong></td>
<td>Rejects A or AAAA records if the corresponding IPv4 or IPv6 addresses match a given address_match_list.</td>
<td>query</td>
</tr>
<tr>
<td><strong>deny-answer-aliases</strong></td>
<td>Rejects CNAME or DNAME records if the &quot;alias&quot; name matches a given list of domain_name elements.</td>
<td>query</td>
</tr>
<tr>
<td><strong>dhparam-file</strong></td>
<td>Specifies the path to a file containing Diffie-Hellman parameters, for enabling cipher suites.</td>
<td>server, security</td>
</tr>
<tr>
<td><strong>dialup</strong></td>
<td>Concentrates zone maintenance so that all transfers take place once every heartbeat-interval, ideally during a single call.</td>
<td>deprecated</td>
</tr>
<tr>
<td><strong>directory</strong></td>
<td>Sets the server’s working directory.</td>
<td>server</td>
</tr>
<tr>
<td><strong>disable-algorithms</strong></td>
<td>Disables DNSSEC algorithms from a specified zone.</td>
<td>dnssec</td>
</tr>
<tr>
<td><strong>disable-ds-digests</strong></td>
<td>Disables DS digest types from a specified zone.</td>
<td>zone, dnssec</td>
</tr>
<tr>
<td><strong>disable-empty-zone</strong></td>
<td>Disables individual empty zones.</td>
<td>server, zone</td>
</tr>
<tr>
<td><strong>dlz</strong></td>
<td>Configures a Dynamically Loadable Zone (DLZ) database in named.conf.</td>
<td>zone</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dns64</code></td>
<td>Instructs named to return mapped IPv4 addresses to AAAA queries when there are no AAAA records.</td>
<td>query</td>
</tr>
<tr>
<td><code>dns64-contact</code></td>
<td>Specifies the name of the contact for <code>dns64</code> zones.</td>
<td>server</td>
</tr>
<tr>
<td><code>dns64-server</code></td>
<td>Specifies the name of the server for <code>dns64</code> zones.</td>
<td>server</td>
</tr>
<tr>
<td><code>dnskey-sig-validity</code></td>
<td>obsolete</td>
<td></td>
</tr>
<tr>
<td><code>dnskey-ttl</code></td>
<td>Specifies the time to live (TTL) for DNSKEY resource records.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>dnssrs-enable</code></td>
<td>Turns on the DNS Response Policy Service (DNRPS) interface.</td>
<td>server, security</td>
</tr>
<tr>
<td><code>dnssrs-library</code></td>
<td>Turns on the DNS Response Policy Service (DNRPS) interface.</td>
<td>server, security</td>
</tr>
<tr>
<td><code>dnssrs-options</code></td>
<td>Provides additional RPZ configuration settings, which are passed to the DNS Response Policy Service (DNRPS) provider library.</td>
<td>server, security</td>
</tr>
<tr>
<td><code>dnsssec-accept-expired</code></td>
<td>Instructs BIND 9 to accept expired DNSSEC signatures when validating.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>dnsssec-dnskey-kskonly</code></td>
<td>obsolete</td>
<td></td>
</tr>
<tr>
<td><code>dnsssec-loadkeys-interval</code></td>
<td>Sets the frequency of automatic checks of the DNSSEC key repository.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>dnsssec-must-be-secure</code></td>
<td>Defines hierarchies that must or may not be secure (signed and validated).</td>
<td>deprecated</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnssec-policy</code></td>
<td>Defines a key and signing policy (KASP) for zones.</td>
<td><code>dnssec</code></td>
</tr>
<tr>
<td><code>dnssec-secure-to-insecure</code></td>
<td>obsolete</td>
<td></td>
</tr>
<tr>
<td><code>dnssec-update-mode</code></td>
<td>obsolete</td>
<td></td>
</tr>
<tr>
<td><code>dnssec-validation</code></td>
<td>Enables DNSSEC validation in <code>named</code>.</td>
<td><code>dnssec</code></td>
</tr>
<tr>
<td><code>dnstap</code></td>
<td>Enables logging of <code>dnstap</code> messages.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>dnstap-identity</code></td>
<td>Specifies an <code>identity</code> string to send in <code>dnstap</code> messages.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>dnstap-output</code></td>
<td>Configures the path to which the <code>dnstap</code> frame stream is sent.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>dnstap-version</code></td>
<td>Specifies a <code>version</code> string to send in <code>dnstap</code> messages.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>dual-stack-servers</code></td>
<td>Specifies host names or addresses of machines with access to both IPv4 and IPv6 transports.</td>
<td><code>server</code></td>
</tr>
<tr>
<td><code>dump-file</code></td>
<td>Indicates the pathname of the file where the server dumps the database after <code>rndc dumpdb</code>.</td>
<td><code>logging</code></td>
</tr>
<tr>
<td><code>dyndb</code></td>
<td>Configures a DynDB database in <code>named.conf</code>.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>edns</code></td>
<td>Controls the use of the EDNS0 (RFC 2671) feature.</td>
<td><code>server</code></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>edns-udp-size</strong></td>
<td>Sets the maximum advertised EDNS UDP buffer size to control the size of packets received from authoritative servers in response to recursive queries.</td>
<td>query</td>
</tr>
<tr>
<td><strong>edns-version</strong></td>
<td>Sets the maximum EDNS VERSION that is sent to the server(s) by the resolver.</td>
<td>server</td>
</tr>
<tr>
<td><strong>empty-contact</strong></td>
<td>Specifies the contact name in the returned SOA record for empty zones.</td>
<td>server, zone</td>
</tr>
<tr>
<td><strong>empty-server</strong></td>
<td>Specifies the server name in the returned SOA record for empty zones.</td>
<td>server, zone</td>
</tr>
<tr>
<td><strong>empty-zones-enable</strong></td>
<td>Enables or disables all empty zones.</td>
<td>server, zone</td>
</tr>
<tr>
<td><strong>endpoints</strong></td>
<td>Specifies a list of HTTP query paths on which to listen.</td>
<td>query, server</td>
</tr>
<tr>
<td><strong>errors-per-second</strong></td>
<td>Limits the number of errors for a valid domain name and record type.</td>
<td>server</td>
</tr>
<tr>
<td><strong>exclude</strong></td>
<td>Allows a list of IPv6 addresses to be ignored if they appear in a domain name's AAAA records in <code>dns64</code>.</td>
<td>query</td>
</tr>
<tr>
<td><strong>exempt-clients</strong></td>
<td>Exempts specific clients or client groups from rate limiting.</td>
<td>query</td>
</tr>
<tr>
<td><strong>fetch-quota-params</strong></td>
<td>Sets the parameters for dynamic resizing of the <code>fetches-per-server</code> quota in response to detected congestion.</td>
<td>query, server</td>
</tr>
</tbody>
</table>

Table 1 – continued from previous page

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetches-per-server</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to be sent by a server to an upstream name server before the server blocks additional queries.</td>
<td>query, server</td>
</tr>
<tr>
<td>fetches-per-zone</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to any one domain before the server blocks new queries for data in or beneath that zone.</td>
<td>query, server</td>
</tr>
<tr>
<td>file</td>
<td>Specifies the zone's filename.</td>
<td>zone</td>
</tr>
<tr>
<td>flush-zones-on-shutdown</td>
<td>Controls whether pending zone writes are flushed when the name server exits.</td>
<td>zone</td>
</tr>
<tr>
<td>forward</td>
<td>Allows or disallows fallback to recursion if forwarding has failed; it is always used in conjunction with the forwarders statement.</td>
<td>query</td>
</tr>
<tr>
<td>forwarders</td>
<td>Defines one or more hosts to which queries are forwarded.</td>
<td>query</td>
</tr>
<tr>
<td>fstrm-set-buffer-hint</td>
<td>Sets the number of accumulated bytes in the output buffer before forcing a buffer flush.</td>
<td>logging</td>
</tr>
<tr>
<td>fstrm-set-flush-timeout</td>
<td>Sets the number of seconds that unflushed data remains in the output buffer.</td>
<td>logging</td>
</tr>
<tr>
<td>fstrm-set-input-queue-size</td>
<td>Sets the number of queue entries to allocate for each input queue.</td>
<td>logging</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>fstrm-set-output-notify-threshold</td>
<td>Sets the number of outstanding queue entries allowed on an input queue before waking the I/O thread.</td>
<td>logging</td>
</tr>
<tr>
<td>fstrm-set-output-queue-model</td>
<td>Sets the queuing semantics to use for queue objects.</td>
<td>logging</td>
</tr>
<tr>
<td>fstrm-set-output-queue-size</td>
<td>Sets the number of queue entries allocated for each output queue.</td>
<td>logging</td>
</tr>
<tr>
<td>fstrm-set-reopen-interval</td>
<td>Sets the number of seconds to wait between attempts to reopen a closed output stream.</td>
<td>logging</td>
</tr>
<tr>
<td>geoip-directory</td>
<td>Specifies the directory containing GeoIP database files.</td>
<td>server</td>
</tr>
<tr>
<td>heartbeat-interval</td>
<td>Sets the interval at which the server performs zone maintenance tasks for all zones marked as dialup.</td>
<td>deprecated</td>
</tr>
<tr>
<td>hostname</td>
<td>Specifies the hostname of the server to return in response to a hostname.bind query.</td>
<td>server</td>
</tr>
<tr>
<td>http</td>
<td>Configures HTTP endpoints on which to listen for DNS-over-HTTPS (DoH) queries.</td>
<td>query, server</td>
</tr>
<tr>
<td>http-listener-clients</td>
<td>Limits the number of active concurrent connections on a per-listener basis.</td>
<td>server</td>
</tr>
<tr>
<td>http-port</td>
<td>Specifies the TCP port number the server uses to receive and send unencrypted DNS traffic via HTTP.</td>
<td>query, server</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
<td>Tags</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>http-streams-per-connection</td>
<td>Limits the number of active concurrent HTTP/2 streams on a per-connection basis.</td>
<td>server</td>
</tr>
<tr>
<td>https-port</td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-HTTPS protocol traffic.</td>
<td>query, server</td>
</tr>
<tr>
<td>in-view</td>
<td>Specifies the view in which a given zone is defined.</td>
<td>zone, view</td>
</tr>
<tr>
<td>inet</td>
<td>Specifies a TCP socket as a control channel.</td>
<td>server</td>
</tr>
<tr>
<td>inline-signing</td>
<td>Specifies whether BIND 9 maintains a separate signed version of a zone.</td>
<td>zone, dnssec</td>
</tr>
<tr>
<td>interface-interval</td>
<td>Sets the interval at which the server scans the network interface list.</td>
<td>server</td>
</tr>
<tr>
<td>ipv4-prefix-length</td>
<td>Specifies the prefix lengths of IPv4 address blocks.</td>
<td>server</td>
</tr>
<tr>
<td>ipv4-only-contact</td>
<td>Specifies the contact for the IPV4ONLY.ARPA zone created by dns64.</td>
<td>server</td>
</tr>
<tr>
<td>ipv4-only-enable</td>
<td>Enables automatic IPv4 zones if a dns64 block is configured.</td>
<td>query</td>
</tr>
<tr>
<td>ipv4-only-server</td>
<td>Specifies the name of the server for the IPV4ONLY.ARPA zone created by dns64.</td>
<td>query, server</td>
</tr>
<tr>
<td>ipv6-prefix-length</td>
<td>Specifies the prefix lengths of IPv6 address blocks.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ixfr-from-differences</code></td>
<td>Controls how IXFR transfers are calculated.</td>
<td><code>transfer</code></td>
</tr>
<tr>
<td><code>journal</code></td>
<td>Allows the default journal’s filename to be overridden.</td>
<td><code>zone</code></td>
</tr>
<tr>
<td><code>key</code></td>
<td>Defines a shared secret key for use with <code>TSIG</code> or the command channel.</td>
<td><code>security</code></td>
</tr>
<tr>
<td><code>key-directory</code></td>
<td>Indicates the directory where public and private DNSSEC key files are found.</td>
<td><code>dnssec</code></td>
</tr>
<tr>
<td><code>key-file</code></td>
<td>Specifies the path to a file containing the private TLS key for a connection.</td>
<td><code>server, security</code></td>
</tr>
<tr>
<td><code>key-store</code></td>
<td>Configures a DNSSEC key store.</td>
<td><code>dnssec</code></td>
</tr>
<tr>
<td><code>keys</code></td>
<td>Specifies one or more <code>server_key</code> s to be used with a remote server.</td>
<td><code>server, security</code></td>
</tr>
<tr>
<td><code>lame-ttl</code></td>
<td>Sets the resolver's lame cache.</td>
<td><code>server</code></td>
</tr>
<tr>
<td><code>listen-on</code></td>
<td>Specifies the IPv4 addresses on which a server listens for DNS queries.</td>
<td><code>server</code></td>
</tr>
<tr>
<td><code>listen-on-v6</code></td>
<td>Specifies the IPv6 addresses on which a server listens for DNS queries.</td>
<td><code>server</code></td>
</tr>
<tr>
<td><code>listener-clients</code></td>
<td>Specifies a per-listener quota for active connections.</td>
<td><code>query, server</code></td>
</tr>
<tr>
<td><code>lmdb-mapsize</code></td>
<td>Sets a maximum size for the memory map of the new-zone database in LMDB database format.</td>
<td><code>server</code></td>
</tr>
</tbody>
</table>

*continues on next page*
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-only</td>
<td>Tests rate-limiting parameters without actually dropping any requests.</td>
<td>query, logging</td>
</tr>
<tr>
<td>logging</td>
<td>Configures logging options for the nameserver.</td>
<td>logging</td>
</tr>
<tr>
<td>managed-keys</td>
<td></td>
<td>deprecated</td>
</tr>
<tr>
<td>managed-keys-directory</td>
<td>Specifies the directory in which to store the files that track managed DNSSEC keys.</td>
<td>dnssec</td>
</tr>
<tr>
<td>mapped</td>
<td>Specifies an access control list (ACL) of IPv4 addresses that are to be mapped to the corresponding A RRset in dns64.</td>
<td>query</td>
</tr>
<tr>
<td>masterfile-format</td>
<td>Specifies the file format of zone files.</td>
<td>server, zone</td>
</tr>
<tr>
<td>masterfile-style</td>
<td>Specifies the format of zone files during a dump, when the masterfile-format is text.</td>
<td>server</td>
</tr>
<tr>
<td>match-clients</td>
<td>Specifies a view of DNS namespace for a given subset of client IP addresses.</td>
<td>view</td>
</tr>
<tr>
<td>match-destinations</td>
<td>Specifies a view of DNS namespace for a given subset of destination IP addresses.</td>
<td>view</td>
</tr>
<tr>
<td>match-mapped-addresses</td>
<td>Allows IPv4-mapped IPv6 addresses to match address-match list entries for corresponding IPv4 addresses.</td>
<td>server</td>
</tr>
<tr>
<td>match-recursive-only</td>
<td>Specifies that only recursive requests can match this view of the DNS namespace.</td>
<td>view</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-cache-size</td>
<td>Sets the maximum amount of memory to use for an individual cache database and its associated metadata.</td>
<td>server</td>
</tr>
<tr>
<td>max-cache-ttl</td>
<td>Specifies the maximum time (in seconds) that the server caches ordinary (positive) answers.</td>
<td>server</td>
</tr>
<tr>
<td>max-clients-per-query</td>
<td>Sets the maximum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.</td>
<td>server</td>
</tr>
<tr>
<td>max-ixfr-ratio</td>
<td>Sets the maximum size for IXFR responses to zone transfer requests.</td>
<td>transfer</td>
</tr>
<tr>
<td>max-journal-size</td>
<td>Controls the size of journal files.</td>
<td>transfer</td>
</tr>
<tr>
<td>max-ncache-ttl</td>
<td>Specifies the maximum retention time (in seconds) for storage of negative answers in the server’s cache.</td>
<td>server</td>
</tr>
<tr>
<td>max-records</td>
<td>Sets the maximum number of records permitted in a zone.</td>
<td>server, zone</td>
</tr>
<tr>
<td>max-records-per-type</td>
<td>Sets the maximum number of records that can be stored in an RRset</td>
<td>server</td>
</tr>
<tr>
<td>max-recursion-depth</td>
<td>Sets the maximum number of levels of recursion permitted at any one time while servicing a recursive query.</td>
<td>server</td>
</tr>
<tr>
<td>max-recursion-queries</td>
<td>Sets the maximum number of iterative queries while servicing a recursive query.</td>
<td>query, server</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
<td>Tags</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>max-refresh-time</code></td>
<td>Limits the zone refresh interval to no less often than the specified value, in seconds.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-retry-time</code></td>
<td>Limits the zone refresh retry interval to no less often than the specified value, in seconds.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-rsa-exponent-size</code></td>
<td>Sets the maximum RSA exponent size (in bits) when validating.</td>
<td>query, dnssec</td>
</tr>
<tr>
<td><code>max-stale-ttl</code></td>
<td>Specifies the maximum time that the server retains records past their normal expiry, to return them as stale records.</td>
<td>server</td>
</tr>
<tr>
<td><code>max-table-size</code></td>
<td>Sets the maximum size of the table used to track requests and rate-limit responses.</td>
<td>server</td>
</tr>
<tr>
<td><code>max-transfer-idle-in</code></td>
<td>Specifies the number of minutes after which inbound zone transfers making no progress are terminated.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-transfer-idle-out</code></td>
<td>Specifies the number of minutes after which outbound zone transfers making no progress are terminated.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-transfer-time-in</code></td>
<td>Specifies the number of minutes after which inbound zone transfers are terminated.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-transfer-time-out</code></td>
<td>Specifies the number of minutes after which outbound zone transfers are terminated.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>max-types-per-name</code></td>
<td>Sets the maximum number of RR types that can be stored for an owner name.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max-udp-size</code></td>
<td>Sets the maximum EDNS UDP message size sent by <code>named</code>.</td>
<td>query</td>
</tr>
<tr>
<td><code>max-validation-failures-per-fetch</code></td>
<td>Set the maximum number of DNSSEC validation failures that can happen in single fetch</td>
<td>server</td>
</tr>
<tr>
<td><code>max-validations-per-fetch</code></td>
<td>Set the maximum number of DNSSEC validations that can happen in single fetch</td>
<td>server</td>
</tr>
<tr>
<td><code>max-zone-ttl</code></td>
<td>Specifies a maximum permissible time-to-live (TTL) value, in seconds.</td>
<td>deprecated</td>
</tr>
<tr>
<td><code>memstatistics</code></td>
<td>Controls whether memory statistics are written to the file specified by <code>memstatistics-file</code> at exit.</td>
<td>server, logging</td>
</tr>
<tr>
<td><code>memstatistics-file</code></td>
<td>Sets the pathname of the file where the server writes memory usage statistics on exit.</td>
<td>logging</td>
</tr>
<tr>
<td><code>message-compression</code></td>
<td>Controls whether DNS name compression is used in responses to regular queries.</td>
<td>query</td>
</tr>
<tr>
<td><code>min-cache-ttl</code></td>
<td>Specifies the minimum time (in seconds) that the server caches ordinary (positive) answers.</td>
<td>server</td>
</tr>
<tr>
<td><code>min-ncache-ttl</code></td>
<td>Specifies the minimum retention time (in seconds) for storage of negative answers in the server's cache.</td>
<td>server</td>
</tr>
<tr>
<td><code>min-refresh-time</code></td>
<td>Limits the zone refresh interval to no more often than the specified value, in seconds.</td>
<td>transfer</td>
</tr>
</tbody>
</table>

continues on next page

8.3. Statements 271
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-retry-time</td>
<td>Limits the zone refresh retry interval to no more often than the specified value, in seconds.</td>
<td>transfer</td>
</tr>
<tr>
<td>min-table-size</td>
<td>Sets the minimum size of the table used to track requests and rate-limit responses.</td>
<td>query</td>
</tr>
<tr>
<td>minimal-any</td>
<td>Controls whether the server replies with only one of the RRsets for a query name, when generating a positive response to a query of type ANY over UDP.</td>
<td>query</td>
</tr>
<tr>
<td>minimal-responses</td>
<td>Controls whether the server only adds records to the authority and additional data sections when they are required (e.g. delegations, negative responses). This improves server performance.</td>
<td>query</td>
</tr>
<tr>
<td>multi-master</td>
<td>Controls whether serial number mismatch errors are logged.</td>
<td>transfer</td>
</tr>
<tr>
<td>new-zones-directory</td>
<td>Specifies the directory where configuration parameters are stored for zones added by <code>rndc addzone</code>.</td>
<td>zone</td>
</tr>
<tr>
<td>no-case-compress</td>
<td>Specifies a list of addresses that require case-insensitive compression in responses.</td>
<td>server</td>
</tr>
<tr>
<td>nocookie-udp-size</td>
<td>Sets the maximum size of UDP responses that are sent to queries without a valid server COOKIE.</td>
<td>query</td>
</tr>
<tr>
<td>nodata-per-second</td>
<td>Limits the number of empty (NO-DATA) responses for a valid domain name.</td>
<td>query</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify</td>
<td>Controls whether NOTIFY messages are sent on zone changes.</td>
<td>transfer</td>
</tr>
<tr>
<td>notify-delay</td>
<td>Sets the delay (in seconds) between sending sets of NOTIFY messages for a zone.</td>
<td>zone, transfer</td>
</tr>
<tr>
<td>notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations.</td>
<td>zone, transfer</td>
</tr>
<tr>
<td>notify-source</td>
<td>Defines the IPv4 address (and optional port) to be used for outgoing NOTIFY messages.</td>
<td>transfer</td>
</tr>
<tr>
<td>notify-source-v6</td>
<td>Defines the IPv6 address (and optional port) to be used for outgoing NOTIFY messages.</td>
<td>transfer</td>
</tr>
<tr>
<td>notify-to-soa</td>
<td>Controls whether the name servers in the NS RRset are checked against the SOA MNAME.</td>
<td>transfer</td>
</tr>
<tr>
<td>nsec3param</td>
<td>Specifies the use of NSEC3 instead of NSEC, and sets NSEC3 parameters.</td>
<td>dnssec</td>
</tr>
<tr>
<td>nta-lifetime</td>
<td>Specifies the lifetime, in seconds, for negative trust anchors added via <code>rndc nta</code>.</td>
<td>dnssec</td>
</tr>
<tr>
<td>nta-recheck</td>
<td>Specifies the time interval for checking whether negative trust anchors added via <code>rndc nta</code> are still necessary.</td>
<td>dnssec</td>
</tr>
<tr>
<td>null</td>
<td>Causes all messages sent to the logging channel to be discarded.</td>
<td>logging</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxdomain-redirect</td>
<td>Appends the specified suffix to the original query name, when replacing an NXDOMAIN with a redirect namespace.</td>
<td>query</td>
</tr>
<tr>
<td>nxdomains-per-second</td>
<td>Limits the number of undefined subdomains for a valid domain name.</td>
<td>query</td>
</tr>
<tr>
<td>options</td>
<td>Defines global options to be used by BIND 9.</td>
<td>server</td>
</tr>
<tr>
<td>padding</td>
<td>Adds EDNS Padding options to outgoing messages to increase the packet size.</td>
<td>server</td>
</tr>
<tr>
<td>parent-ds-ttl</td>
<td>Sets the time to live (TTL) of the DS RRset used by the parent zone.</td>
<td>dnssec</td>
</tr>
<tr>
<td>parent-propagation-delay</td>
<td>Sets the propagation delay from the time the parent zone is updated to when the new version is served by all of the parent zone's name servers.</td>
<td>zone, dnssec</td>
</tr>
<tr>
<td>parental-agents</td>
<td>Defines a list of delegation agents to be used by primary and secondary zones.</td>
<td>zone</td>
</tr>
<tr>
<td>parental-source</td>
<td>Specifies which local IPv4 source address is used to send parental DS queries.</td>
<td>dnssec</td>
</tr>
<tr>
<td>parental-source-v6</td>
<td>Specifies which local IPv6 source address is used to send parental DS queries.</td>
<td>dnssec</td>
</tr>
<tr>
<td>pid-file</td>
<td>Specifies the pathname of the file where the server writes its process ID.</td>
<td>server</td>
</tr>
<tr>
<td>pkcs11-uri</td>
<td></td>
<td>pkcs11, dnssec</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin</td>
<td>Configures plugins in named.conf.</td>
<td>server</td>
</tr>
<tr>
<td>port</td>
<td>Specifies the UDP/TCP port number the server uses to receive and send DNS protocol traffic.</td>
<td>query, server</td>
</tr>
<tr>
<td>prefer-server-ciphers</td>
<td>Specifies that server ciphers should be preferred over client ones.</td>
<td>server, security</td>
</tr>
<tr>
<td>preferred-glue</td>
<td>Controls the order of glue records in an A or AAAA response.</td>
<td>query</td>
</tr>
<tr>
<td>prefetch</td>
<td>Specifies the &quot;trigger&quot; time-to-live (TTL) value at which prefetch of the current query takes place.</td>
<td>query</td>
</tr>
<tr>
<td>primaries</td>
<td>Defines one or more primary servers for a zone.</td>
<td>zone</td>
</tr>
<tr>
<td>print-category</td>
<td>Includes the category in log messages.</td>
<td>logging</td>
</tr>
<tr>
<td>print-severity</td>
<td>Includes the severity in log messages.</td>
<td>logging</td>
</tr>
<tr>
<td>print-time</td>
<td>Specifies the time format for log messages.</td>
<td>logging</td>
</tr>
<tr>
<td>protocols</td>
<td>Specifies the allowed versions of the TLS protocol.</td>
<td>security</td>
</tr>
<tr>
<td>provide-ixfr</td>
<td>Controls whether a primary responds to an incremental zone request (IXFR) or only responds with a full zone transfer (AXFR).</td>
<td>transfer</td>
</tr>
</tbody>
</table>
Table 1 - continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>publish-safety</td>
<td>Increases the amount of time between when keys are published and when they become active, to allow for unforeseen events.</td>
<td>dnssec</td>
</tr>
<tr>
<td>purge-keys</td>
<td>Specifies the amount of time after which DNSSEC keys that have been deleted from the zone can be removed from disk.</td>
<td>dnssec</td>
</tr>
<tr>
<td>qname-minimization</td>
<td>Controls QNAME minimization behavior in the BIND 9 resolver.</td>
<td>query</td>
</tr>
<tr>
<td>qps-scale</td>
<td>Tightens defenses during DNS attacks by scaling back the ratio of the current query-per-second rate.</td>
<td>query</td>
</tr>
<tr>
<td>query-source</td>
<td>Controls the IPv4 address from which queries are issued.</td>
<td>query</td>
</tr>
<tr>
<td>query-source-v6</td>
<td>Controls the IPv6 address from which queries are issued.</td>
<td>query</td>
</tr>
<tr>
<td>querylog</td>
<td>Specifies whether query logging should be active when named first starts.</td>
<td>server, logging</td>
</tr>
<tr>
<td>rate-limit</td>
<td>Controls excessive UDP responses, to prevent BIND 9 from being used to amplify reflection denial-of-service (DoS) attacks.</td>
<td>query</td>
</tr>
<tr>
<td>recursing-file</td>
<td>Specifies the pathname of the file where the server dumps queries that are currently recursing via rndc recursing.</td>
<td>server</td>
</tr>
<tr>
<td>recursion</td>
<td>Defines whether recursion and caching are allowed.</td>
<td>query</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive-clients</td>
<td>Specifies the maximum number of concurrent recursive queries the server can perform.</td>
<td>query</td>
</tr>
<tr>
<td>recursive-only</td>
<td>Toggles whether <em>dns64</em> synthesis occurs only for recursive queries.</td>
<td>query</td>
</tr>
<tr>
<td>referrals-per-second</td>
<td>Limits the number of referrals or delegations to a server for a given domain.</td>
<td>query</td>
</tr>
<tr>
<td>remote-hostname</td>
<td>Specifies the expected hostname in the TLS certificate of the remote server.</td>
<td>security</td>
</tr>
<tr>
<td>request-expires</td>
<td>Specifies whether the local server requests the EDNS EXPIRE value, when acting as a secondary.</td>
<td>query, transfer</td>
</tr>
<tr>
<td>request-ixfr</td>
<td>Controls whether a secondary requests an incremental zone transfer (IXFR) or a full zone transfer (AXFR).</td>
<td>transfer</td>
</tr>
<tr>
<td>request-nssid</td>
<td>Controls whether an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution.</td>
<td>query</td>
</tr>
<tr>
<td>require-cookie</td>
<td>Controls whether responses without a server cookie are accepted</td>
<td>query</td>
</tr>
<tr>
<td>require-server-cookie</td>
<td>Controls whether a valid server cookie is required before sending a full response to a UDP request.</td>
<td>query</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolver-query-timeout</code></td>
<td>Specifies the length of time, in milliseconds, that a resolver attempts to resolve a recursive query before failing.</td>
<td>query</td>
</tr>
<tr>
<td><code>resolver-use-dns64</code></td>
<td>Specifies whether to apply DNS64 mappings when sending queries.</td>
<td>server</td>
</tr>
<tr>
<td><code>response-padding</code></td>
<td>Adds an EDNS Padding option to encrypted messages, to reduce the chance of guessing the contents based on size.</td>
<td>query</td>
</tr>
<tr>
<td><code>response-policy</code></td>
<td>Specifies response policy zones for the view or among global options.</td>
<td>query, server, zone, security</td>
</tr>
<tr>
<td><code>responses-per-second</code></td>
<td>Limits the number of non-empty responses for a valid domain name and record type.</td>
<td>query</td>
</tr>
<tr>
<td><code>retire-safety</code></td>
<td>Increases the amount of time a key remains published after it is no longer active, to allow for unforeseen events.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>reuseport</code></td>
<td>Enables kernel load-balancing of sockets.</td>
<td>server</td>
</tr>
<tr>
<td><code>root-key-sentinel</code></td>
<td>Controls whether BIND 9 responds to root key sentinel probes.</td>
<td>server</td>
</tr>
<tr>
<td><code>rrset-order</code></td>
<td>Defines the order in which equal RR sets (RR sets) are returned.</td>
<td>query</td>
</tr>
<tr>
<td><code>search</code></td>
<td>Specifies whether a Dynamically Loadable Zone (DLZ) module is queried for an answer to a query name.</td>
<td>query</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>Defines a Base64-encoded string to be used as the secret by the algorithm.</td>
<td>security</td>
</tr>
<tr>
<td>secrts-file</td>
<td>Specifies the pathname of the file where the server dumps security roots, when using <code>rndc secrts</code>.</td>
<td>dnssec</td>
</tr>
<tr>
<td>send-cookie</td>
<td>Controls whether a COOKIE EDNS option is sent along with a query.</td>
<td>query</td>
</tr>
<tr>
<td>serial-query-rate</td>
<td>Defines an upper limit on the number of queries per second issued by the server, when querying the SOA RRs used for zone transfers.</td>
<td>transfer</td>
</tr>
<tr>
<td>serial-update-method</td>
<td>Specifies the update method to be used for the zone serial number in the SOA record.</td>
<td>zone</td>
</tr>
<tr>
<td>server</td>
<td>Defines characteristics to be associated with a remote name server.</td>
<td>server</td>
</tr>
<tr>
<td>server-addresses</td>
<td>Specifies a list of IP addresses to which queries should be sent in recursive resolution for a static-stub zone.</td>
<td>query, zone</td>
</tr>
<tr>
<td>server-id</td>
<td>Specifies the ID of the server to return in response to a ID.SERVER query.</td>
<td>server</td>
</tr>
<tr>
<td>server-names</td>
<td>Specifies a list of domain names of name servers that act as authoritative servers of a static-stub zone.</td>
<td>zone</td>
</tr>
<tr>
<td>servfail-ttl</td>
<td>Sets the length of time (in seconds) that a SERVFAIL response is cached.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session-keyalg</code></td>
<td>Specifies the algorithm to use for the TSIG session key.</td>
<td>security</td>
</tr>
<tr>
<td><code>session-keyfile</code></td>
<td>Specifies the pathname of the file where a TSIG session key is written, when generated by <code>named</code> for use by <code>nsupdate -l</code>.</td>
<td>security</td>
</tr>
<tr>
<td><code>session-keyname</code></td>
<td>Specifies the key name for the TSIG session key.</td>
<td>security</td>
</tr>
<tr>
<td><code>session-tickets</code></td>
<td>Enables or disables session resumption through TLS session tickets.</td>
<td>security</td>
</tr>
<tr>
<td><code>severity</code></td>
<td>Defines the priority level of log messages.</td>
<td>logging</td>
</tr>
<tr>
<td><code>sig-signing-nodes</code></td>
<td>Specifies the maximum number of nodes to be examined in each quantum, when signing a zone with a new DNSKEY.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>sig-signing-signatures</code></td>
<td>Specifies the threshold for the number of signatures that terminates processing a quantum, when signing a zone with a new DNSKEY.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>sig-signing-type</code></td>
<td>Specifies a private RDATA type to use when generating signing-state records.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>sig-validity-interval</code></td>
<td></td>
<td>obsolete</td>
</tr>
<tr>
<td><code>sig0checks-quota</code></td>
<td>Specifies the maximum number of concurrent SIG(0) signature checks that can be processed by the server.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sig0checks-quota-exempt</code></td>
<td>Exempts specific clients or client groups from SIG(0) signature checking quota.</td>
<td>server</td>
</tr>
<tr>
<td><code>signatures-jitter</code></td>
<td>Specifies a range for signatures expirations.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>signatures-refresh</code></td>
<td>Specifies how frequently an RRSIG record is refreshed.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>signatures-validity</code></td>
<td>Indicates the validity period of an RRSIG record.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>signatures-validity-dnskey</code></td>
<td>Indicates the validity period of DNSKEY records.</td>
<td>dnssec</td>
</tr>
<tr>
<td><code>slip</code></td>
<td>Sets the number of &quot;slipped&quot; responses to minimize the use of forged source addresses for an attack.</td>
<td>query</td>
</tr>
<tr>
<td><code>sortlist</code></td>
<td>Controls the ordering of RRs returned to the client, based on the client’s IP address.</td>
<td>query, deprecated</td>
</tr>
<tr>
<td><code>stale-answer-client-timeout</code></td>
<td>Defines the amount of time (in milliseconds) that <em>named</em> waits before attempting to answer a query with a stale RRset from cache.</td>
<td>query, server</td>
</tr>
<tr>
<td><code>stale-answer-enable</code></td>
<td>Enables the returning of &quot;stale&quot; cached answers when the name servers for a zone are not answering.</td>
<td>query, server</td>
</tr>
<tr>
<td><code>stale-answer-ttl</code></td>
<td>Specifies the time to live (TTL) to be returned on stale answers, in seconds.</td>
<td>query</td>
</tr>
<tr>
<td><code>stale-cache-enable</code></td>
<td>Enables the retention of &quot;stale&quot; cached answers.</td>
<td>query, server</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>stale-refresh-time</td>
<td>Sets the time window for the return of &quot;stale&quot; cached answers before the next attempt to contact, if the name servers for a given zone are not responding.</td>
<td>query, server</td>
</tr>
<tr>
<td>startup-notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent when the name server is first starting, or when new zones have been added.</td>
<td>zone, transfer</td>
</tr>
<tr>
<td>statistics-channels</td>
<td>Specifies the communication channels to be used by system administrators to access statistics information on the name server.</td>
<td>logging</td>
</tr>
<tr>
<td>statistics-file</td>
<td>Specifies the pathname of the file where the server appends statistics, when using <code>rndc stats</code>.</td>
<td>server, logging</td>
</tr>
<tr>
<td>stderr</td>
<td>Directs the logging channel output to the server’s standard error stream.</td>
<td>logging</td>
</tr>
<tr>
<td>streams-per-connection</td>
<td>Specifies the maximum number of concurrent HTTP/2 streams over an HTTP/2 connection.</td>
<td>query, server</td>
</tr>
<tr>
<td>suffix</td>
<td>Defines trailing bits for mapped IPv4 address bits in <code>dns64</code>.</td>
<td>query</td>
</tr>
<tr>
<td>synth-from-dnssec</td>
<td>Enables support for RFC 8198, Aggressive Use of DNSSEC-Validated Cache.</td>
<td>dnssec</td>
</tr>
<tr>
<td>syslog</td>
<td>Directs the logging channel to the system log.</td>
<td>logging</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp-advertised-timeout</td>
<td>Sets the timeout value (in milliseconds) that the server sends in responses containing the EDNS TCP keepalive option.</td>
<td>query</td>
</tr>
<tr>
<td>tcp-clients</td>
<td>Specifies the maximum number of simultaneous client TCP connections accepted by the server.</td>
<td>server</td>
</tr>
<tr>
<td>tcp-idle-timeout</td>
<td>Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is not in use.</td>
<td>query</td>
</tr>
<tr>
<td>tcp-initial-timeout</td>
<td>Sets the amount of time (in milliseconds) that the server waits on a new TCP connection for the first message from the client.</td>
<td>query, server</td>
</tr>
<tr>
<td>tcp-keepalive</td>
<td>Adds EDNS TCP keepalive to messages sent over TCP.</td>
<td>server</td>
</tr>
<tr>
<td>tcp-keepalive-timeout</td>
<td>Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is in use.</td>
<td>query</td>
</tr>
<tr>
<td>tcp-listen-queue</td>
<td>Sets the listen-queue depth.</td>
<td>server</td>
</tr>
<tr>
<td>tcp-only</td>
<td>Sets the transport protocol to TCP.</td>
<td>server</td>
</tr>
<tr>
<td>tcp-receive-buffer</td>
<td>Sets the operating system’s receive buffer size for TCP sockets.</td>
<td>server</td>
</tr>
<tr>
<td>tcp-send-buffer</td>
<td>Sets the operating system’s send buffer size for TCP sockets.</td>
<td>server</td>
</tr>
</tbody>
</table>

continues on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tkey-domain</code></td>
<td>Sets the domain appended to the names of all shared keys generated with <strong>TKEY</strong>.</td>
<td>security</td>
</tr>
<tr>
<td><code>tkey-gssapi-credential</code></td>
<td>Sets the security credential for authentication keys requested by the GSS-TSIG protocol.</td>
<td>security</td>
</tr>
<tr>
<td><code>tkey-gssapi-keytab</code></td>
<td>Sets the KRB5 keytab file to use for GSS-TSIG updates.</td>
<td>security</td>
</tr>
<tr>
<td><code>tls</code></td>
<td>Configures a TLS connection.</td>
<td>security</td>
</tr>
<tr>
<td><code>tls-port</code></td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-TLS protocol traffic.</td>
<td>query, server</td>
</tr>
<tr>
<td><code>transfer-format</code></td>
<td>Controls whether multiple records can be packed into a message during zone transfers.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>transfer-message-size</code></td>
<td>Limits the uncompressed size of DNS messages used in zone transfers over TCP.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>transfer-source</code></td>
<td>Defines which local IPv4 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>transfer-source-v6</code></td>
<td>Defines which local IPv6 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.</td>
<td>transfer</td>
</tr>
<tr>
<td><code>transfers</code></td>
<td>Limits the number of concurrent inbound zone transfers from a server.</td>
<td>server</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
<td>Tags</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>transfers-in</td>
<td>Limits the number of concurrent in-bound zone transfers.</td>
<td>transfer</td>
</tr>
<tr>
<td>transfers-out</td>
<td>Limits the number of concurrent out-bound zone transfers.</td>
<td>transfer</td>
</tr>
<tr>
<td>transfers-per-ns</td>
<td>Limits the number of concurrent in-bound zone transfers from a remote server.</td>
<td>transfer</td>
</tr>
<tr>
<td>trust-anchor-telemetry</td>
<td>Instructs named to send specially formed queries once per day to domains for which trust anchors have been configured.</td>
<td>dnssec</td>
</tr>
<tr>
<td>trust-anchors</td>
<td>Defines DNSSEC trust anchors.</td>
<td>dnssec</td>
</tr>
<tr>
<td>trusted-keys</td>
<td></td>
<td>deprecated</td>
</tr>
<tr>
<td>try-tcp-refresh</td>
<td>Specifies that BIND 9 should attempt to refresh a zone using TCP if UDP queries fail.</td>
<td>transfer</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the kind of zone in a given configuration.</td>
<td>zone</td>
</tr>
<tr>
<td>type forward</td>
<td>Contains forwarding statements that apply to queries within a given domain.</td>
<td>zone</td>
</tr>
<tr>
<td>type hint</td>
<td>Contains the initial set of root name servers to be used at BIND 9 startup.</td>
<td>zone</td>
</tr>
<tr>
<td>type mirror</td>
<td>Contains a DNSSEC-validated duplicate of the main data for a zone.</td>
<td>zone</td>
</tr>
<tr>
<td>type primary</td>
<td>Contains the main copy of the data for a zone.</td>
<td>zone</td>
</tr>
</tbody>
</table>

continues on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>type redirect</td>
<td>Contains information to answer queries when normal resolution would return NXDOMAIN.</td>
<td>zone</td>
</tr>
<tr>
<td>type secondary</td>
<td>Contains a duplicate of the data for a zone that has been transferred from a primary server.</td>
<td>zone</td>
</tr>
<tr>
<td>type static-stub</td>
<td>Contains a duplicate of the NS records of a primary zone, but statically configured rather than transferred from a primary server.</td>
<td>zone</td>
</tr>
<tr>
<td>type stub</td>
<td>Contains a duplicate of the NS records of a primary zone.</td>
<td>zone</td>
</tr>
<tr>
<td>udp-receive-buffer</td>
<td>Sets the operating system’s receive buffer size for UDP sockets.</td>
<td>server</td>
</tr>
<tr>
<td>udp-send-buffer</td>
<td>Sets the operating system’s send buffer size for UDP sockets.</td>
<td>server</td>
</tr>
<tr>
<td>unix</td>
<td>Specifies a Unix domain socket as a control channel.</td>
<td>obsolete</td>
</tr>
<tr>
<td>update-check-ksk</td>
<td>obsolete</td>
<td></td>
</tr>
<tr>
<td>update-policy</td>
<td>Sets fine-grained rules to allow or deny dynamic updates (DDNS), based on requester identity, updated content, etc.</td>
<td>transfer</td>
</tr>
<tr>
<td>update-quota</td>
<td>Specifies the maximum number of concurrent DNS UPDATE messages that can be processed by the server.</td>
<td>server</td>
</tr>
<tr>
<td>use-v4-udp-ports</td>
<td>Specifies a list of ports that are valid sources for UDP/IPv4 messages.</td>
<td>deprecated</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-v6-udp-ports</td>
<td>Specifies a list of ports that are valid sources for UDP/IPv6 messages.</td>
<td>deprecated</td>
</tr>
<tr>
<td>v6-bias</td>
<td>Indicates the number of milliseconds of preference to give to IPv6 name servers.</td>
<td>query, server</td>
</tr>
<tr>
<td>validate-except</td>
<td>Specifies a list of domain names at and beneath which DNSSEC validation should not be performed.</td>
<td>dnssec</td>
</tr>
<tr>
<td>version</td>
<td>Specifies the version number of the server to return in response to a version.bind query.</td>
<td>server</td>
</tr>
<tr>
<td>view</td>
<td>Allows a name server to answer a DNS query differently depending on who is asking.</td>
<td>view</td>
</tr>
<tr>
<td>window</td>
<td>Specifies the length of time during which responses are tracked.</td>
<td>query</td>
</tr>
<tr>
<td>zero-no-soa-ttl</td>
<td>Specifies whether to set the time to live (TTL) of the SOA record to zero, when returning authoritative negative responses to SOA queries.</td>
<td>query, server, zone</td>
</tr>
<tr>
<td>zero-no-soa-ttl-cache</td>
<td>Sets the time to live (TTL) to zero when caching a negative response to an SOA query.</td>
<td>query, server, zone</td>
</tr>
<tr>
<td>zone</td>
<td>Specifies the zone in a BIND 9 configuration.</td>
<td>zone</td>
</tr>
<tr>
<td>zone-propagation-delay</td>
<td>Sets the propagation delay from the time a zone is first updated to when the new version of the zone is served by all secondary servers.</td>
<td>zone, dnssec</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone-statistics</td>
<td>Controls the level of statistics gathered for all zones.</td>
<td>logging, zone</td>
</tr>
</tbody>
</table>

8.4 Statements by Tag

These tables group the various statements permissible in named.conf by their corresponding tag.

8.4.1 DNSSEC Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindkeys-file</td>
<td>Specifies the pathname of a file to override the built-in trusted keys provided by named.</td>
</tr>
<tr>
<td>cdnskey</td>
<td>Specifies whether a CDNSKEY record should be published during KSK rollover.</td>
</tr>
<tr>
<td>cds-digest-types</td>
<td>Specifies the digest types to use for CDS resource records.</td>
</tr>
<tr>
<td>check-dup-records</td>
<td>Checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS.</td>
</tr>
<tr>
<td>checkds</td>
<td>Controls whether DS queries are sent to parental agents.</td>
</tr>
<tr>
<td>disable-algorithms</td>
<td>Disables DNSSEC algorithms from a specified zone.</td>
</tr>
<tr>
<td>disable-ds-digests</td>
<td>Disables DS digest types from a specified zone.</td>
</tr>
<tr>
<td>dnskey-ttl</td>
<td>Specifies the time to live (TTL) for DNSKEY resource records.</td>
</tr>
<tr>
<td>dnssec-accept-expired</td>
<td>Instructs BIND 9 to accept expired DNSSEC signatures when validating.</td>
</tr>
</tbody>
</table>
### Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnssec-loadkeys-interval</code></td>
<td>Sets the frequency of automatic checks of the DNSSEC key repository.</td>
</tr>
<tr>
<td><code>dnssec-policy</code></td>
<td>Defines a key and signing policy (KASP) for zones.</td>
</tr>
<tr>
<td><code>dnssec-validation</code></td>
<td>Enables DNSSEC validation in <code>named</code>.</td>
</tr>
<tr>
<td><code>inline-signing</code></td>
<td>Specifies whether BIND 9 maintains a separate signed version of a zone.</td>
</tr>
<tr>
<td><code>key-directory</code></td>
<td>Indicates the directory where public and private DNSSEC key files are found.</td>
</tr>
<tr>
<td><code>key-store</code></td>
<td>Configures a DNSSEC key store.</td>
</tr>
<tr>
<td><code>managed-keys-directory</code></td>
<td>Specifies the directory in which to store the files that track managed DNSSEC keys.</td>
</tr>
<tr>
<td><code>max-rsa-exponent-size</code></td>
<td>Sets the maximum RSA exponent size (in bits) when validating.</td>
</tr>
<tr>
<td><code>nsec3param</code></td>
<td>Specifies the use of NSEC3 instead of NSEC, and sets NSEC3 parameters.</td>
</tr>
<tr>
<td><code>nta-lifetime</code></td>
<td>Specifies the lifetime, in seconds, for negative trust anchors added via <code>rndc nta</code>.</td>
</tr>
<tr>
<td><code>nta-recheck</code></td>
<td>Specifies the time interval for checking whether negative trust anchors added via <code>rndc nta</code> are still necessary.</td>
</tr>
<tr>
<td><code>parent-ds-ttl</code></td>
<td>Sets the time to live (TTL) of the DS RRset used by the parent zone.</td>
</tr>
<tr>
<td><code>parent-propagation-delay</code></td>
<td>Sets the propagation delay from the time the parent zone is updated to when the new version is served by all of the parent zone's name servers.</td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parental-source</td>
<td>Specifies which local IPv4 source address is used to send parental DS queries.</td>
</tr>
<tr>
<td>parental-source-v6</td>
<td>Specifies which local IPv6 source address is used to send parental DS queries.</td>
</tr>
<tr>
<td>pkcs11-uri</td>
<td></td>
</tr>
<tr>
<td>publish-safety</td>
<td>Increases the amount of time between when keys are published and when they become active, to allow for unforeseen events.</td>
</tr>
<tr>
<td>purge-keys</td>
<td>Specifies the amount of time after which DNSSEC keys that have been deleted from the zone can be removed from disk.</td>
</tr>
<tr>
<td>retire-safety</td>
<td>Increases the amount of time a key remains published after it is no longer active, to allow for unforeseen events.</td>
</tr>
<tr>
<td>secroots-file</td>
<td>Specifies the pathname of the file where the server dumps security roots, when using <code>rndc secroots</code>.</td>
</tr>
<tr>
<td>sig-signing-nodes</td>
<td>Specifies the maximum number of nodes to be examined in each quantum, when signing a zone with a new DNSKEY.</td>
</tr>
<tr>
<td>sig-signing-signatures</td>
<td>Specifies the threshold for the number of signatures that terminates processing a quantum, when signing a zone with a new DNSKEY.</td>
</tr>
<tr>
<td>sig-signing-type</td>
<td>Specifies a private RDATA type to use when generating signing-state records.</td>
</tr>
<tr>
<td>signatures-jitter</td>
<td>Specifies a range for signatures expirations.</td>
</tr>
<tr>
<td>signatures-refresh</td>
<td>Specifies how frequently an RRSIG record is refreshed.</td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signatures-validity</td>
<td>Indicates the validity period of an RRSIG record.</td>
</tr>
<tr>
<td>signatures-validity-dnskey</td>
<td>Indicates the validity period of DNSKEY records.</td>
</tr>
<tr>
<td>synth-from-dnssec</td>
<td>Enables support for RFC 8198, Aggressive Use of DNSSEC-Validated Cache.</td>
</tr>
<tr>
<td>trust-anchor-telemetry</td>
<td>Instructs named to send specially formed queries once per day to domains for which trust anchors have been configured.</td>
</tr>
<tr>
<td>trust-anchors</td>
<td>Defines DNSSEC trust anchors.</td>
</tr>
<tr>
<td>validate-except</td>
<td>Specifies a list of domain names at and beneath which DNSSEC validation should not be performed.</td>
</tr>
<tr>
<td>zone-propagation-delay</td>
<td>Sets the propagation delay from the time a zone is first updated to when the new version of the zone is served by all secondary servers.</td>
</tr>
</tbody>
</table>

8.4.2 Logging Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffered</td>
<td>Controls flushing of log messages.</td>
</tr>
<tr>
<td>category</td>
<td>Specifies the type of data logged to a particular channel.</td>
</tr>
<tr>
<td>channel</td>
<td>Defines a stream of data that can be independently logged.</td>
</tr>
<tr>
<td>dnstap</td>
<td>Enables logging of dnstap messages.</td>
</tr>
<tr>
<td>dnstap-identity</td>
<td>Specifies an identity string to send in dnstap messages.</td>
</tr>
</tbody>
</table>

continues on next page
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dnstap-output</code></td>
<td>Configures the path to which the <code>dnstap</code> frame stream is sent.</td>
</tr>
<tr>
<td><code>dnstap-version</code></td>
<td>Specifies a <code>version</code> string to send in <code>dnstap</code> messages.</td>
</tr>
<tr>
<td><code>dump-file</code></td>
<td>Indicates the pathname of the file where the server dumps the database after <code>rndc dumpdb</code>.</td>
</tr>
<tr>
<td><code>fstrm-set-buffer-hint</code></td>
<td>Sets the number of accumulated bytes in the output buffer before forcing a buffer flush.</td>
</tr>
<tr>
<td><code>fstrm-set-flush-timeout</code></td>
<td>Sets the number of seconds that unflushed data remains in the output buffer.</td>
</tr>
<tr>
<td><code>fstrm-set-input-queue-size</code></td>
<td>Sets the number of queue entries to allocate for each input queue.</td>
</tr>
<tr>
<td><code>fstrm-set-output-notify-threshold</code></td>
<td>Sets the number of outstanding queue entries allowed on an input queue before waking the I/O thread.</td>
</tr>
<tr>
<td><code>fstrm-set-output-queue-model</code></td>
<td>Sets the queuing semantics to use for queue objects.</td>
</tr>
<tr>
<td><code>fstrm-set-output-queue-size</code></td>
<td>Sets the number of queue entries allocated for each output queue.</td>
</tr>
<tr>
<td><code>fstrm-set-reopen-interval</code></td>
<td>Sets the number of seconds to wait between attempts to reopen a closed output stream.</td>
</tr>
<tr>
<td><code>log-only</code></td>
<td>Tests rate-limiting parameters without actually dropping any requests.</td>
</tr>
<tr>
<td><code>logging</code></td>
<td>Configures logging options for the name server.</td>
</tr>
<tr>
<td><code>memstatistics</code></td>
<td>Controls whether memory statistics are written to the file specified by <code>memstatistics-file</code> at exit.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>memstatistics-file</td>
<td>Sets the pathname of the file where the server writes memory usage statistics on exit.</td>
</tr>
<tr>
<td>null</td>
<td>Causes all messages sent to the logging channel to be discarded.</td>
</tr>
<tr>
<td>print-category</td>
<td>Includes the category in log messages.</td>
</tr>
<tr>
<td>print-severity</td>
<td>Includes the severity in log messages.</td>
</tr>
<tr>
<td>print-time</td>
<td>Specifies the time format for log messages.</td>
</tr>
<tr>
<td>querylog</td>
<td>Specifies whether query logging should be active when <code>named</code> first starts.</td>
</tr>
<tr>
<td>severity</td>
<td>Defines the priority level of log messages.</td>
</tr>
<tr>
<td>statistics-channels</td>
<td>Specifies the communication channels to be used by system administrators to access statistics information on the name server.</td>
</tr>
<tr>
<td>statistics-file</td>
<td>Specifies the pathname of the file where the server appends statistics, when using <code>rndc stats</code>.</td>
</tr>
<tr>
<td>stderr</td>
<td>Directs the logging channel output to the server’s standard error stream.</td>
</tr>
<tr>
<td>syslog</td>
<td>Directs the logging channel to the system log.</td>
</tr>
<tr>
<td>zone-statistics</td>
<td>Controls the level of statistics gathered for all zones.</td>
</tr>
</tbody>
</table>
### 8.4.3 Query Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-per-second</td>
<td>Limits UDP responses of all kinds.</td>
</tr>
<tr>
<td>allow-query</td>
<td>Specifies which hosts (an IP address list) are allowed to send queries to this resolver.</td>
</tr>
<tr>
<td>allow-query-cache</td>
<td>Specifies which hosts (an IP address list) can access this server’s cache and thus effectively controls recursion.</td>
</tr>
<tr>
<td>allow-query-cache-on</td>
<td>Specifies which hosts (an IP address list) can access this server’s cache. Used on servers with multiple interfaces.</td>
</tr>
<tr>
<td>allow-query-on</td>
<td>Specifies which local addresses (an IP address list) are allowed to send queries to this resolver. Used in multi-homed configurations.</td>
</tr>
<tr>
<td>allow-recursion</td>
<td>Defines an address_match_list of clients that are allowed to perform recursive queries.</td>
</tr>
<tr>
<td>allow-recursion-on</td>
<td>Specifies which local addresses can accept recursive queries.</td>
</tr>
<tr>
<td>answer-cookie</td>
<td>Controls whether COOKIE EDNS replies are sent in response to client queries.</td>
</tr>
<tr>
<td>auth-nxdomain</td>
<td>Controls whether BIND, acting as a resolver, provides authoritative NXDOMAIN (domain does not exist) answers.</td>
</tr>
<tr>
<td>blackhole</td>
<td>Defines an address_match_list of hosts to ignore. The server will neither respond to queries from nor send queries to these addresses.</td>
</tr>
<tr>
<td>break-dnssec</td>
<td>Enables dns64 synthesis even if the validated result would cause a DNSSEC validation failure.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>check-dup-records</td>
<td>Checks primary zones for records that are treated as different by DNSSEC but are semantically equal in plain DNS.</td>
</tr>
<tr>
<td>check-names</td>
<td>Restricts the character set and syntax of certain domain names in primary files and/or DNS responses received from the network.</td>
</tr>
<tr>
<td>clients</td>
<td>Specifies an access control list (ACL) of clients that are affected by a given <code>dns64</code> directive.</td>
</tr>
<tr>
<td>deny-answer-addresses</td>
<td>Rejects A or AAAA records if the corresponding IPv4 or IPv6 addresses match a given <code>address_match_list</code>.</td>
</tr>
<tr>
<td>deny-answer-aliases</td>
<td>Rejects CNAME or DNAME records if the &quot;alias&quot; name matches a given list of <code>domain_name</code> elements.</td>
</tr>
<tr>
<td>dns64</td>
<td>Instructs <code>named</code> to return mapped IPv4 addresses to AAAA queries when there are no AAAA records.</td>
</tr>
<tr>
<td>edns-udp-size</td>
<td>Sets the maximum advertised EDNS UDP buffer size to control the size of packets received from authoritative servers in response to recursive queries.</td>
</tr>
<tr>
<td>endpoints</td>
<td>Specifies a list of HTTP query paths on which to listen.</td>
</tr>
<tr>
<td>exclude</td>
<td>Allows a list of IPv6 addresses to be ignored if they appear in a domain name's AAAA records in <code>dns64</code>.</td>
</tr>
<tr>
<td>exempt-clients</td>
<td>Exempts specific clients or client groups from rate limiting.</td>
</tr>
<tr>
<td>fetch-quota-params</td>
<td>Sets the parameters for dynamic resizing of the <code>fetches-per-server</code> quota in response to detected congestion.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fetches-per-server</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to be sent by a server to an upstream name server before the server blocks additional queries.</td>
</tr>
<tr>
<td>fetches-per-zone</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to any one domain before the server blocks new queries for data in or beneath that zone.</td>
</tr>
<tr>
<td>forward</td>
<td>Allows or disallows fallback to recursion if forwarding has failed; it is always used in conjunction with the <code>forwarders</code> statement.</td>
</tr>
<tr>
<td>forwarders</td>
<td>Defines one or more hosts to which queries are forwarded.</td>
</tr>
<tr>
<td>http</td>
<td>Configures HTTP endpoints on which to listen for DNS-over-HTTPS (DoH) queries.</td>
</tr>
<tr>
<td>http-port</td>
<td>Specifies the TCP port number the server uses to receive and send unencrypted DNS traffic via HTTP.</td>
</tr>
<tr>
<td>https-port</td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-HTTPS protocol traffic.</td>
</tr>
<tr>
<td>ipv4only-enable</td>
<td>Enables automatic IPv4 zones if a <code>dns64</code> block is configured.</td>
</tr>
<tr>
<td>ipv4only-server</td>
<td>Specifies the name of the server for the IPV4ONLY.ARPA zone created by <code>dns64</code>.</td>
</tr>
<tr>
<td>listener-clients</td>
<td>Specifies a per-listener quota for active connections.</td>
</tr>
<tr>
<td>log-only</td>
<td>Tests rate-limiting parameters without actually dropping any requests.</td>
</tr>
<tr>
<td>mapped</td>
<td>Specifies an access control list (ACL) of IPv4 addresses that are to be mapped to the corresponding A RRset in <code>dns64</code>.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-recursion-queries</td>
<td>Sets the maximum number of iterative queries while servicing a recursive query.</td>
</tr>
<tr>
<td>max-rsa-exponent-size</td>
<td>Sets the maximum RSA exponent size (in bits) when validating.</td>
</tr>
<tr>
<td>max-udp-size</td>
<td>Sets the maximum EDNS UDP message size sent by <code>named</code>.</td>
</tr>
<tr>
<td>message-compression</td>
<td>Controls whether DNS name compression is used in responses to regular queries.</td>
</tr>
<tr>
<td>min-table-size</td>
<td>Sets the minimum size of the table used to track requests and rate-limit responses.</td>
</tr>
<tr>
<td>minimal-any</td>
<td>Controls whether the server replies with only one of the RRsets for a query name, when generating a positive response to a query of type ANY over UDP.</td>
</tr>
<tr>
<td>minimal-responses</td>
<td>Controls whether the server only adds records to the authority and additional data sections when they are required (e.g. delegations, negative responses). This improves server performance.</td>
</tr>
<tr>
<td>nocookie-udp-size</td>
<td>Sets the maximum size of UDP responses that are sent to queries without a valid server COOKIE.</td>
</tr>
<tr>
<td>nodata-per-second</td>
<td>Limits the number of empty (NODATA) responses for a valid domain name.</td>
</tr>
<tr>
<td>nxdomain-redirect</td>
<td>Appends the specified suffix to the original query name, when replacing an NXDOMAIN with a redirect namespace.</td>
</tr>
<tr>
<td>nxdomains-per-second</td>
<td>Limits the number of undefined subdomains for a valid domain name.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Specifies the UDP/TCP port number the server uses to receive and send DNS protocol traffic.</td>
</tr>
<tr>
<td>preferred-glue</td>
<td>Controls the order of glue records in an A or AAAA response.</td>
</tr>
<tr>
<td>prefetch</td>
<td>Specifies the &quot;trigger&quot; time-to-live (TTL) value at which prefetch of the current query takes place.</td>
</tr>
<tr>
<td>qname-minimization</td>
<td>Controls QNAME minimization behavior in the BIND 9 resolver.</td>
</tr>
<tr>
<td>qps-scale</td>
<td>Tightens defenses during DNS attacks by scaling back the ratio of the current query-per-second rate.</td>
</tr>
<tr>
<td>query-source</td>
<td>Controls the IPv4 address from which queries are issued.</td>
</tr>
<tr>
<td>query-source-v6</td>
<td>Controls the IPv6 address from which queries are issued.</td>
</tr>
<tr>
<td>rate-limit</td>
<td>Controls excessive UDP responses, to prevent BIND 9 from being used to amplify reflection denial-of-service (DoS) attacks.</td>
</tr>
<tr>
<td>recursion</td>
<td>Defines whether recursion and caching are allowed.</td>
</tr>
<tr>
<td>recursive-clients</td>
<td>Specifies the maximum number of concurrent recursive queries the server can perform.</td>
</tr>
<tr>
<td>recursive-only</td>
<td>Toggles whether dns64 synthesis occurs only for recursive queries.</td>
</tr>
<tr>
<td>referrals-per-second</td>
<td>Limits the number of referrals or delegations to a server for a given domain.</td>
</tr>
<tr>
<td>request-expire</td>
<td>Specifies whether the local server requests the EDNS EXPIRE value, when acting as a secondary.</td>
</tr>
</tbody>
</table>
### Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request-nsid</td>
<td>Controls whether an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution.</td>
</tr>
<tr>
<td>require-cookie</td>
<td>Controls whether responses without a server cookie are accepted</td>
</tr>
<tr>
<td>require-server-cookie</td>
<td>Controls whether a valid server cookie is required before sending a full response to a UDP request.</td>
</tr>
<tr>
<td>resolver-query-timeout</td>
<td>Specifies the length of time, in milliseconds, that a resolver attempts to resolve a recursive query before failing.</td>
</tr>
<tr>
<td>response-padding</td>
<td>Adds an EDNS Padding option to encrypted messages, to reduce the chance of guessing the contents based on size.</td>
</tr>
<tr>
<td>response-policy</td>
<td>Specifies response policy zones for the view or among global options.</td>
</tr>
<tr>
<td>responses-per-second</td>
<td>Limits the number of non-empty responses for a valid domain name and record type.</td>
</tr>
<tr>
<td>rrset-order</td>
<td>Defines the order in which equal RRs (RRsets) are returned.</td>
</tr>
<tr>
<td>search</td>
<td>Specifies whether a Dynamically Loadable Zone (DLZ) module is queried for an answer to a query name.</td>
</tr>
<tr>
<td>send-cookie</td>
<td>Controls whether a COOKIE EDNS option is sent along with a query.</td>
</tr>
<tr>
<td>server-addresses</td>
<td>Specifies a list of IP addresses to which queries should be sent in recursive resolution for a static-stub zone.</td>
</tr>
<tr>
<td>slip</td>
<td>Sets the number of &quot;slipped&quot; responses to minimize the use of forged source addresses for an attack.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sortlist</code></td>
<td>Controls the ordering of RRs returned to the client, based on the client’s IP address.</td>
</tr>
<tr>
<td><code>stale-answer-client-timeout</code></td>
<td>Defines the amount of time (in milliseconds) that <code>named</code> waits before attempting to answer a query with a stale RRset from cache.</td>
</tr>
<tr>
<td><code>stale-answer-enable</code></td>
<td>Enables the returning of &quot;stale&quot; cached answers when the name servers for a zone are not answering.</td>
</tr>
<tr>
<td><code>stale-answer-ttl</code></td>
<td>Specifies the time to live (TTL) to be returned on stale answers, in seconds.</td>
</tr>
<tr>
<td><code>stale-cache-enable</code></td>
<td>Enables the retention of &quot;stale&quot; cached answers.</td>
</tr>
<tr>
<td><code>stale-refresh-time</code></td>
<td>Sets the time window for the return of &quot;stale&quot; cached answers before the next attempt to contact, if the name servers for a given zone are not responding.</td>
</tr>
<tr>
<td><code>streams-per-connection</code></td>
<td>Specifies the maximum number of concurrent HTTP/2 streams over an HTTP/2 connection.</td>
</tr>
<tr>
<td><code>suffix</code></td>
<td>Defines trailing bits for mapped IPv4 address bits in <code>dns64</code>.</td>
</tr>
<tr>
<td><code>tcp-advertised-timeout</code></td>
<td>Sets the timeout value (in milliseconds) that the server sends in responses containing the EDNS TCP keepalive option.</td>
</tr>
<tr>
<td><code>tcp-idle-timeout</code></td>
<td>Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is not in use.</td>
</tr>
<tr>
<td><code>tcp-initial-timeout</code></td>
<td>Sets the amount of time (in milliseconds) that the server waits on a new TCP connection for the first message from the client.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tcp-keepalive-timeout</td>
<td>Sets the amount of time (in milliseconds) that the server waits on an idle TCP connection before closing it, if the EDNS TCP keepalive option is in use.</td>
</tr>
<tr>
<td>tls-port</td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-TLS protocol traffic.</td>
</tr>
<tr>
<td>v6-bias</td>
<td>Indicates the number of milliseconds of preference to give to IPv6 name servers.</td>
</tr>
<tr>
<td>window</td>
<td>Specifies the length of time during which responses are tracked.</td>
</tr>
<tr>
<td>zero-no-soa-ttl</td>
<td>Specifies whether to set the time to live (TTL) of the SOA record to zero, when returning authoritative negative responses to SOA queries.</td>
</tr>
<tr>
<td>zero-no-soa-ttl-cache</td>
<td>Sets the time to live (TTL) to zero when caching a negative response to an SOA query.</td>
</tr>
</tbody>
</table>
### 8.4.4 Security Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithm</code></td>
<td>Defines the algorithm to be used in a key clause.</td>
</tr>
<tr>
<td><code>ca-file</code></td>
<td>Specifies the path to a file containing TLS certificates for trusted CA authorities, used to verify remote peer certificates.</td>
</tr>
<tr>
<td><code>cert-file</code></td>
<td>Specifies the path to a file containing the TLS certificate for a connection.</td>
</tr>
<tr>
<td><code>cipher-suites</code></td>
<td>Specifies a list of allowed cipher suites in the order of preference for TLSv1.3 only.</td>
</tr>
<tr>
<td><code>ciphers</code></td>
<td>Specifies a list of allowed ciphers in the order of preference for TLSv1.2 only.</td>
</tr>
<tr>
<td><code>dhparam-file</code></td>
<td>Specifies the path to a file containing Diffie-Hellman parameters, for enabling cipher suites.</td>
</tr>
<tr>
<td><code>dnsrps-enable</code></td>
<td>Turns on the DNS Response Policy Service (DNSRPS) interface.</td>
</tr>
<tr>
<td><code>dnsrps-library</code></td>
<td>Turns on the DNS Response Policy Service (DNSRPS) interface.</td>
</tr>
<tr>
<td><code>dnsrps-options</code></td>
<td>Provides additional RPZ configuration settings, which are passed to the DNS Response Policy Service (DNSRPS) provider library.</td>
</tr>
<tr>
<td><code>key</code></td>
<td>Defines a shared secret key for use with TSIG or the command channel.</td>
</tr>
<tr>
<td><code>key-file</code></td>
<td>Specifies the path to a file containing the private TLS key for a connection.</td>
</tr>
<tr>
<td><code>keys</code></td>
<td>Specifies one or more <code>server_key</code>s to be used with a remote server.</td>
</tr>
<tr>
<td><code>prefer-server-ciphers</code></td>
<td>Specifies that server ciphers should be preferred over client ones.</td>
</tr>
<tr>
<td><code>protocols</code></td>
<td>Specifies the allowed versions of the TLS protocol.</td>
</tr>
<tr>
<td><code>remote-hostname</code></td>
<td>Specifies the expected hostname in the TLS certificate of the remote server.</td>
</tr>
<tr>
<td><code>response-policy</code></td>
<td>Specifies response policy zones for the view or among global options.</td>
</tr>
<tr>
<td><code>secret</code></td>
<td>Defines a Base64-encoded string to be used as the secret by the algorithm.</td>
</tr>
<tr>
<td><code>session-keyalg</code></td>
<td>Specifies the algorithm to use for the TSIG session key.</td>
</tr>
<tr>
<td><code>session-keyfile</code></td>
<td>Specifies the pathname of the file where a TSIG session key is written, when generated by named for use by nsupdate -l.</td>
</tr>
<tr>
<td><code>session-keyname</code></td>
<td>Specifies the key name for the TSIG session key.</td>
</tr>
<tr>
<td><code>session-tickets</code></td>
<td>Enables or disables session resumption through TLS session tickets.</td>
</tr>
<tr>
<td><code>tkey-domain</code></td>
<td>Sets the domain appended to the names of all shared keys generated with TKEY.</td>
</tr>
<tr>
<td><code>tkey-gssapi-credential</code></td>
<td>Sets the security credential for authentication keys requested by the GSS-TSIG protocol.</td>
</tr>
<tr>
<td><code>tkey-gssapi-keytab</code></td>
<td>Sets the KRB5 keytab file to use for GSS-TSIG updates.</td>
</tr>
<tr>
<td><code>tls</code></td>
<td>Configures a TLS connection.</td>
</tr>
</tbody>
</table>
### 8.4.5 Server Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acl</code></td>
<td>Assigns a symbolic name to an address match list.</td>
</tr>
<tr>
<td><code>allow-new-zones</code></td>
<td>Controls the ability to add zones at runtime via <code>rndc addzone</code>.</td>
</tr>
<tr>
<td><code>allow-proxy</code></td>
<td>Defines an <code>address_match_list</code> for the client addresses allowed to send PROXYv2 headers.</td>
</tr>
<tr>
<td><code>allow-proxy-on</code></td>
<td>Defines an <code>address_match_list</code> for the interface addresses allowed to accept PROXYv2 headers. The option is mostly intended for multi-homed configurations.</td>
</tr>
<tr>
<td><code>allow-recursion-on</code></td>
<td>Specifies which local addresses can accept recursive queries.</td>
</tr>
<tr>
<td><code>automatic-interface-scan</code></td>
<td>Controls the automatic rescanning of network interfaces when addresses are added or removed.</td>
</tr>
<tr>
<td><code>bogus</code></td>
<td>Allows a remote server to be ignored.</td>
</tr>
<tr>
<td><code>ca-file</code></td>
<td>Specifies the path to a file containing TLS certificates for trusted CA authorities, used to verify remote peer certificates.</td>
</tr>
<tr>
<td><code>cert-file</code></td>
<td>Specifies the path to a file containing the TLS certificate for a connection.</td>
</tr>
<tr>
<td><code>check-names</code></td>
<td>Restricts the character set and syntax of certain domain names in primary files and/or DNS responses received from the network.</td>
</tr>
<tr>
<td><code>clients-per-query</code></td>
<td>Sets the initial minimum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>controls</td>
<td>Specifies control channels to be used to manage the name server.</td>
</tr>
<tr>
<td>cookie-algorithm</td>
<td>Sets the algorithm to be used when generating a server cookie.</td>
</tr>
<tr>
<td>cookie-secret</td>
<td>Specifies a shared secret used for generating and verifying EDNS COOKIE options within an anycast cluster.</td>
</tr>
<tr>
<td>dhparam-file</td>
<td>Specifies the path to a file containing Diffie-Hellman parameters, for enabling cipher suites.</td>
</tr>
<tr>
<td>directory</td>
<td>Sets the server’s working directory.</td>
</tr>
<tr>
<td>disable-empty-zone</td>
<td>Disables individual empty zones.</td>
</tr>
<tr>
<td>dns64-contact</td>
<td>Specifies the name of the contact for dns64 zones.</td>
</tr>
<tr>
<td>dns64-server</td>
<td>Specifies the name of the server for dns64 zones.</td>
</tr>
<tr>
<td>dnsrps-enable</td>
<td>Turns on the DNS Response Policy Service (DNSRPS) interface.</td>
</tr>
<tr>
<td>dnsrps-library</td>
<td>Turns on the DNS Response Policy Service (DNSRPS) interface.</td>
</tr>
<tr>
<td>dnsrps-options</td>
<td>Provides additional RPZ configuration settings, which are passed to the DNS Response Policy Service (DNSRPS) provider library.</td>
</tr>
<tr>
<td>dual-stack-servers</td>
<td>Specifies host names or addresses of machines with access to both IPv4 and IPv6 transports.</td>
</tr>
<tr>
<td>edns</td>
<td>Controls the use of the EDNS0 (RFC 2671) feature.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>edns-version</td>
<td>Sets the maximum EDNS VERSION that is sent to the server(s) by the resolver.</td>
</tr>
<tr>
<td>empty-contact</td>
<td>Specifies the contact name in the returned SOA record for empty zones.</td>
</tr>
<tr>
<td>empty-server</td>
<td>Specifies the server name in the returned SOA record for empty zones.</td>
</tr>
<tr>
<td>empty-zones-enable</td>
<td>Enables or disables all empty zones.</td>
</tr>
<tr>
<td>endpoints</td>
<td>Specifies a list of HTTP query paths on which to listen.</td>
</tr>
<tr>
<td>errors-per-second</td>
<td>Limits the number of errors for a valid domain name and record type.</td>
</tr>
<tr>
<td>fetch-quota-params</td>
<td>Sets the parameters for dynamic resizing of the fetches-per-server quota in response to detected congestion.</td>
</tr>
<tr>
<td>fetches-per-server</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to be sent by a server to an upstream name server before the server blocks additional queries.</td>
</tr>
<tr>
<td>fetches-per-zone</td>
<td>Sets the maximum number of simultaneous iterative queries allowed to any one domain before the server blocks new queries for data in or beneath that zone.</td>
</tr>
<tr>
<td>geoip-directory</td>
<td>Specifies the directory containing GeoIP database files.</td>
</tr>
<tr>
<td>hostname</td>
<td>Specifies the hostname of the server to return in response to a hostname.bind query.</td>
</tr>
<tr>
<td>http</td>
<td>Configures HTTP endpoints on which to listen for DNS-over-HTTPS (DoH) queries.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>http-listener-clients</code></td>
<td>Limits the number of active concurrent connections on a per-listener basis.</td>
</tr>
<tr>
<td><code>http-port</code></td>
<td>Specifies the TCP port number the server uses to receive and send unencrypted DNS traffic via HTTP.</td>
</tr>
<tr>
<td><code>http-streams-per-connection</code></td>
<td>Limits the number of active concurrent HTTP/2 streams on a per-connection basis.</td>
</tr>
<tr>
<td><code>https-port</code></td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-HTTPS protocol traffic.</td>
</tr>
<tr>
<td><code>inet</code></td>
<td>Specifies a TCP socket as a control channel.</td>
</tr>
<tr>
<td><code>interface-interval</code></td>
<td>Sets the interval at which the server scans the network interface list.</td>
</tr>
<tr>
<td><code>ipv4-prefix-length</code></td>
<td>Specifies the prefix lengths of IPv4 address blocks.</td>
</tr>
<tr>
<td><code>ipv4only-contact</code></td>
<td>Specifies the contact for the IPV4ONLY.ARPA zone created by <code>dns64</code>.</td>
</tr>
<tr>
<td><code>ipv4only-server</code></td>
<td>Specifies the name of the server for the IPV4ONLY.ARPA zone created by <code>dns64</code>.</td>
</tr>
<tr>
<td><code>ipv6-prefix-length</code></td>
<td>Specifies the prefix lengths of IPv6 address blocks.</td>
</tr>
<tr>
<td><code>key-file</code></td>
<td>Specifies the path to a file containing the private TLS key for a connection.</td>
</tr>
<tr>
<td><code>keys</code></td>
<td>Specifies one or more <code>server_key</code> s to be used with a remote server.</td>
</tr>
<tr>
<td><code>lame-ttl</code></td>
<td>Sets the resolver's lame cache.</td>
</tr>
</tbody>
</table>
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen-on</td>
<td>Specifies the IPv4 addresses on which a server listens for DNS queries.</td>
</tr>
<tr>
<td>listen-on-v6</td>
<td>Specifies the IPv6 addresses on which a server listens for DNS queries.</td>
</tr>
<tr>
<td>listener-clients</td>
<td>Specifies a per-listener quota for active connections.</td>
</tr>
<tr>
<td>lmdb-mapsize</td>
<td>Sets a maximum size for the memory map of the new-zone database in LMDB database format.</td>
</tr>
<tr>
<td>masterfile-format</td>
<td>Specifies the file format of zone files.</td>
</tr>
<tr>
<td>masterfile-style</td>
<td>Specifies the format of zone files during a dump, when the masterfile-format is text.</td>
</tr>
<tr>
<td>match-mapped-addresses</td>
<td>Allows IPv4-mapped IPv6 addresses to match address-match list entries for corresponding IPv4 addresses.</td>
</tr>
<tr>
<td>max-cache-size</td>
<td>Sets the maximum amount of memory to use for an individual cache database and its associated metadata.</td>
</tr>
<tr>
<td>max-cache-ttl</td>
<td>Specifies the maximum time (in seconds) that the server caches ordinary (positive) answers.</td>
</tr>
<tr>
<td>max-clients-per-query</td>
<td>Sets the maximum number of simultaneous recursive clients accepted by the server for any given query before the server drops additional clients.</td>
</tr>
<tr>
<td>max-ncache-ttl</td>
<td>Specifies the maximum retention time (in seconds) for storage of negative answers in the server’s cache.</td>
</tr>
<tr>
<td>max-records</td>
<td>Sets the maximum number of records permitted in a zone.</td>
</tr>
<tr>
<td>max-records-per-type</td>
<td>Sets the maximum number of records that can be stored in an RRset.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max-recursion-depth</code></td>
<td>Sets the maximum number of levels of recursion permitted at any one time while servicing a recursive query.</td>
</tr>
<tr>
<td><code>max-recursion-queries</code></td>
<td>Sets the maximum number of iterative queries while servicing a recursive query.</td>
</tr>
<tr>
<td><code>max-stale-ttl</code></td>
<td>Specifies the maximum time that the server retains records past their normal expiry, to return them as stale records.</td>
</tr>
<tr>
<td><code>max-table-size</code></td>
<td>Sets the maximum size of the table used to track requests and rate-limit responses.</td>
</tr>
<tr>
<td><code>max-types-per-name</code></td>
<td>Sets the maximum number of RR types that can be stored for an owner name.</td>
</tr>
<tr>
<td><code>max-validation-failures-per-fetch</code></td>
<td>Set the maximum number of DNSSEC validation failures that can happen in single fetch</td>
</tr>
<tr>
<td><code>max-validations-per-fetch</code></td>
<td>Set the maximum number of DNSSEC validations that can happen in single fetch.</td>
</tr>
<tr>
<td><code>memstatistics</code></td>
<td>Controls whether memory statistics are written to the file specified by <code>memstatistics-file</code> at exit.</td>
</tr>
<tr>
<td><code>min-cache-ttl</code></td>
<td>Specifies the minimum time (in seconds) that the server caches ordinary (positive) answers.</td>
</tr>
<tr>
<td><code>min-ncache-ttl</code></td>
<td>Specifies the minimum retention time (in seconds) for storage of negative answers in the server’s cache.</td>
</tr>
<tr>
<td><code>no-case-compress</code></td>
<td>Specifies a list of addresses that require case-insensitive compression in responses.</td>
</tr>
<tr>
<td><code>options</code></td>
<td>Defines global options to be used by BIND 9.</td>
</tr>
</tbody>
</table>

continues on next page
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>padding</td>
<td>Adds EDNS Padding options to outgoing messages to increase the packet size.</td>
</tr>
<tr>
<td>pid-file</td>
<td>Specifies the pathname of the file where the server writes its process ID.</td>
</tr>
<tr>
<td>plugin</td>
<td>Configures plugins in named.conf.</td>
</tr>
<tr>
<td>port</td>
<td>Specifies the UDP/TCP port number the server uses to receive and send DNS protocol traffic.</td>
</tr>
<tr>
<td>prefer-server-ciphers</td>
<td>Specifies that server ciphers should be preferred over client ones.</td>
</tr>
<tr>
<td>querylog</td>
<td>Specifies whether query logging should be active when named first starts.</td>
</tr>
<tr>
<td>recursing-file</td>
<td>Specifies the pathname of the file where the server dumps queries that are currently recursing via rndc recursing.</td>
</tr>
<tr>
<td>resolver-use-dns64</td>
<td>Specifies whether to apply DNS64 mappings when sending queries.</td>
</tr>
<tr>
<td>response-policy</td>
<td>Specifies response policy zones for the view or among global options.</td>
</tr>
<tr>
<td>reuseport</td>
<td>Enables kernel load-balancing of sockets.</td>
</tr>
<tr>
<td>root-key-sentinel</td>
<td>Controls whether BIND 9 responds to root key sentinel probes.</td>
</tr>
<tr>
<td>server</td>
<td>Defines characteristics to be associated with a remote name server.</td>
</tr>
</tbody>
</table>

continues on next page
### Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>server-id</code></td>
<td>Specifies the ID of the server to return in response to a <code>ID.SERVER</code> query.</td>
</tr>
<tr>
<td><code>servfail-ttl</code></td>
<td>Sets the length of time (in seconds) that a SERVFAIL response is cached.</td>
</tr>
<tr>
<td><code>sig0checks-quota</code></td>
<td>Specifies the maximum number of concurrent SIG(0) signature checks that can be processed by the server.</td>
</tr>
<tr>
<td><code>sig0checks-quota-exempt</code></td>
<td>Exempts specific clients or client groups from SIG(0) signature checking quota.</td>
</tr>
<tr>
<td><code>stale-answer-client-timeout</code></td>
<td>Defines the amount of time (in milliseconds) that <code>named</code> waits before attempting to answer a query with a stale RRset from cache.</td>
</tr>
<tr>
<td><code>stale-answer-enable</code></td>
<td>Enables the returning of &quot;stale&quot; cached answers when the name servers for a zone are not answering.</td>
</tr>
<tr>
<td><code>stale-cache-enable</code></td>
<td>Enables the retention of &quot;stale&quot; cached answers.</td>
</tr>
<tr>
<td><code>stale-refresh-time</code></td>
<td>Sets the time window for the return of &quot;stale&quot; cached answers before the next attempt to contact, if the name servers for a given zone are not responding.</td>
</tr>
<tr>
<td><code>statistics-file</code></td>
<td>Specifies the pathname of the file where the server appends statistics, when using <code>rndc stats</code>.</td>
</tr>
<tr>
<td><code>streams-per-connection</code></td>
<td>Specifies the maximum number of concurrent HTTP/2 streams over an HTTP/2 connection.</td>
</tr>
<tr>
<td><code>tcp-clients</code></td>
<td>Specifies the maximum number of simultaneous client TCP connections accepted by the server.</td>
</tr>
<tr>
<td><code>tcp-initial-timeout</code></td>
<td>Sets the amount of time (in milliseconds) that the server waits on a new TCP connection for the first message from the client.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp-keepalive</td>
<td>Adds EDNS TCP keepalive to messages sent over TCP.</td>
</tr>
<tr>
<td>tcp-listen-queue</td>
<td>Sets the listen-queue depth.</td>
</tr>
<tr>
<td>tcp-only</td>
<td>Sets the transport protocol to TCP.</td>
</tr>
<tr>
<td>tcp-receive-buffer</td>
<td>Sets the operating system's receive buffer size for TCP sockets.</td>
</tr>
<tr>
<td>tcp-send-buffer</td>
<td>Sets the operating system's send buffer size for TCP sockets.</td>
</tr>
<tr>
<td>tls-port</td>
<td>Specifies the TCP port number the server uses to receive and send DNS-over-TLS protocol traffic.</td>
</tr>
<tr>
<td>transfers</td>
<td>Limits the number of concurrent inbound zone transfers from a server.</td>
</tr>
<tr>
<td>udp-receive-buffer</td>
<td>Sets the operating system's receive buffer size for UDP sockets.</td>
</tr>
<tr>
<td>udp-send-buffer</td>
<td>Sets the operating system's send buffer size for UDP sockets.</td>
</tr>
<tr>
<td>update-quota</td>
<td>Specifies the maximum number of concurrent DNS UPDATE messages that can be processed by the server.</td>
</tr>
<tr>
<td>v6-bias</td>
<td>Indicates the number of milliseconds of preference to give to IPv6 name servers.</td>
</tr>
<tr>
<td>version</td>
<td>Specifies the version number of the server to return in response to a version.bind query.</td>
</tr>
<tr>
<td>zero-no-soa-ttl</td>
<td>Specifies whether to set the time to live (TTL) of the SOA record to zero, when returning authoritative negative responses to SOA queries.</td>
</tr>
</tbody>
</table>
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero-no-soa-ttl-cache</td>
<td>Sets the time to live (TTL) to zero when caching a negative response to an SOA query.</td>
</tr>
</tbody>
</table>

### 8.4.6 Transfer Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-notify</td>
<td>Defines an address_match_list that is allowed to send NOTIFY messages for the zone, in addition to addresses defined in the primaries option for the zone.</td>
</tr>
<tr>
<td>allow-transfer</td>
<td>Defines an address_match_list of hosts that are allowed to transfer the zone information from this server.</td>
</tr>
<tr>
<td>allow-update</td>
<td>Defines an address_match_list of hosts that are allowed to submit dynamic updates for primary zones.</td>
</tr>
<tr>
<td>allow-update-forwarding</td>
<td>Defines an address_match_list of hosts that are allowed to submit dynamic updates to a secondary server for transmission to a primary.</td>
</tr>
<tr>
<td>also-notify</td>
<td>Defines one or more hosts that are sent NOTIFY messages when zone changes occur.</td>
</tr>
<tr>
<td>ixfr-from-differences</td>
<td>Controls how IXFR transfers are calculated.</td>
</tr>
<tr>
<td>max-ixfr-ratio</td>
<td>Sets the maximum size for IXFR responses to zone transfer requests.</td>
</tr>
<tr>
<td>max-journal-size</td>
<td>Controls the size of journal files.</td>
</tr>
<tr>
<td>max-refresh-time</td>
<td>Limits the zone refresh interval to no less often than the specified value, in seconds.</td>
</tr>
<tr>
<td>max-retry-time</td>
<td>Limits the zone refresh retry interval to no less often than the specified value, in seconds.</td>
</tr>
</tbody>
</table>

continues on next page
### Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-transfer-idle-in</td>
<td>Specifies the number of minutes after which inbound zone transfers making no progress are terminated.</td>
</tr>
<tr>
<td>max-transfer-idle-out</td>
<td>Specifies the number of minutes after which outbound zone transfers making no progress are terminated.</td>
</tr>
<tr>
<td>max-transfer-time-in</td>
<td>Specifies the number of minutes after which inbound zone transfers are terminated.</td>
</tr>
<tr>
<td>max-transfer-time-out</td>
<td>Specifies the number of minutes after which outbound zone transfers are terminated.</td>
</tr>
<tr>
<td>min-refresh-time</td>
<td>Limits the zone refresh interval to no more often than the specified value, in seconds.</td>
</tr>
<tr>
<td>min-retry-time</td>
<td>Limits the zone refresh retry interval to no more often than the specified value, in seconds.</td>
</tr>
<tr>
<td>multi-master</td>
<td>Controls whether serial number mismatch errors are logged.</td>
</tr>
<tr>
<td>notify</td>
<td>Controls whether NOTIFY messages are sent on zone changes.</td>
</tr>
<tr>
<td>notify-delay</td>
<td>Sets the delay (in seconds) between sending sets of NOTIFY messages for a zone.</td>
</tr>
<tr>
<td>notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations.</td>
</tr>
<tr>
<td>notify-source</td>
<td>Defines the IPv4 address (and optional port) to be used for outgoing NOTIFY messages.</td>
</tr>
<tr>
<td>notify-source-v6</td>
<td>Defines the IPv6 address (and optional port) to be used for outgoing NOTIFY messages.</td>
</tr>
</tbody>
</table>

continues on next page
Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify-to-soa</td>
<td>Controls whether the name servers in the NS RRset are checked against the SOA MNAME.</td>
</tr>
<tr>
<td>provide-ixfr</td>
<td>Controls whether a primary responds to an incremental zone request (IXFR) or only responds with a full zone transfer (AXFR).</td>
</tr>
<tr>
<td>request-expire</td>
<td>Specifies whether the local server requests the EDNS EXPIRE value, when acting as a secondary.</td>
</tr>
<tr>
<td>request-ixfr</td>
<td>Controls whether a secondary requests an incremental zone transfer (IXFR) or a full zone transfer (AXFR).</td>
</tr>
<tr>
<td>serial-query-rate</td>
<td>Defines an upper limit on the number of queries per second issued by the server, when querying the SOA RRs used for zone transfers.</td>
</tr>
<tr>
<td>startup-notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent when the name server is first starting, or when new zones have been added.</td>
</tr>
<tr>
<td>transfer-format</td>
<td>Controls whether multiple records can be packed into a message during zone transfers.</td>
</tr>
<tr>
<td>transfer-message-size</td>
<td>Limits the uncompressed size of DNS messages used in zone transfers over TCP.</td>
</tr>
<tr>
<td>transfer-source</td>
<td>Defines which local IPv4 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.</td>
</tr>
<tr>
<td>transfer-source-v6</td>
<td>Defines which local IPv6 address(es) are bound to TCP connections used to fetch zones transferred inbound by the server.</td>
</tr>
<tr>
<td>transfers-in</td>
<td>Limits the number of concurrent inbound zone transfers.</td>
</tr>
</tbody>
</table>

continues on next page
Table 6 – continued from previous page

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transfers-out</td>
<td>Limits the number of concurrent outbound zone transfers.</td>
</tr>
<tr>
<td>transfers-per-ns</td>
<td>Limits the number of concurrent inbound zone transfers from a remote server.</td>
</tr>
<tr>
<td>try-tcp-refresh</td>
<td>Specifies that BIND 9 should attempt to refresh a zone using TCP if UDP queries fail.</td>
</tr>
<tr>
<td>update-policy</td>
<td>Sets fine-grained rules to allow or deny dynamic updates (DDNS), based on requester identity, updated content, etc.</td>
</tr>
</tbody>
</table>

8.4.7 View Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attach-cache</td>
<td>Allows multiple views to share a single cache database.</td>
</tr>
<tr>
<td>in-view</td>
<td>Specifies the view in which a given zone is defined.</td>
</tr>
<tr>
<td>match-clients</td>
<td>Specifies a view of DNS namespace for a given subset of client IP addresses.</td>
</tr>
<tr>
<td>match-destinations</td>
<td>Specifies a view of DNS namespace for a given subset of destination IP addresses.</td>
</tr>
<tr>
<td>match-recursive-only</td>
<td>Specifies that only recursive requests can match this view of the DNS namespace.</td>
</tr>
<tr>
<td>view</td>
<td>Allows a name server to answer a DNS query differently depending on who is asking.</td>
</tr>
</tbody>
</table>
## 8.4.8 Zone Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-new-zones</td>
<td>Controls the ability to add zones at runtime via <code>rndc addzone</code>.</td>
</tr>
<tr>
<td>catalog-zones</td>
<td>Configures catalog zones in <code>named.conf</code>.</td>
</tr>
<tr>
<td>check-integrity</td>
<td>Performs post-load zone integrity checks on primary zones.</td>
</tr>
<tr>
<td>check-mx</td>
<td>Checks whether an MX record appears to refer to an IP address.</td>
</tr>
<tr>
<td>check-mx-cname</td>
<td>Sets the response to MX records that refer to CNAMEs.</td>
</tr>
<tr>
<td>check-sibling</td>
<td>Specifies whether to check for sibling glue when performing integrity checks.</td>
</tr>
<tr>
<td>check-spf</td>
<td>Specifies whether to check for a TXT Sender Policy Framework record, if an SPF record is present.</td>
</tr>
<tr>
<td>check-srv-cname</td>
<td>Sets the response to SRV records that refer to CNAMEs.</td>
</tr>
<tr>
<td>check-svcb</td>
<td>Specifies whether to perform additional checks on SVCB records.</td>
</tr>
<tr>
<td>check-wildcard</td>
<td>Checks for non-terminal wildcards.</td>
</tr>
<tr>
<td>database</td>
<td>Specifies the type of database to be used to store zone data.</td>
</tr>
<tr>
<td>disable-ds-digests</td>
<td>Disables DS digest types from a specified zone.</td>
</tr>
<tr>
<td>disable-empty-zone</td>
<td>Disables individual empty zones.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>dlz</em></td>
<td>Configures a Dynamically Loadable Zone (DLZ) database in <em>named.conf</em>.</td>
</tr>
<tr>
<td><em>dyndb</em></td>
<td>Configures a DynDB database in <em>named.conf</em>.</td>
</tr>
<tr>
<td><em>empty-contact</em></td>
<td>Specifies the contact name in the returned SOA record for empty zones.</td>
</tr>
<tr>
<td><em>empty-server</em></td>
<td>Specifies the server name in the returned SOA record for empty zones.</td>
</tr>
<tr>
<td><em>empty-zones-enable</em></td>
<td>Enables or disables all empty zones.</td>
</tr>
<tr>
<td><em>file</em></td>
<td>Specifies the zone's filename.</td>
</tr>
<tr>
<td><em>flush-zones-on-shutdown</em></td>
<td>Controls whether pending zone writes are flushed when the name server exits.</td>
</tr>
<tr>
<td><em>in-view</em></td>
<td>Specifies the view in which a given zone is defined.</td>
</tr>
<tr>
<td><em>inline-signing</em></td>
<td>Specifies whether BIND 9 maintains a separate signed version of a zone.</td>
</tr>
<tr>
<td><em>journal</em></td>
<td>Allows the default journal's filename to be overridden.</td>
</tr>
<tr>
<td><em>masterfile-format</em></td>
<td>Specifies the file format of zone files.</td>
</tr>
<tr>
<td><em>max-records</em></td>
<td>Sets the maximum number of records permitted in a zone.</td>
</tr>
<tr>
<td><em>new-zones-directory</em></td>
<td>Specifies the directory where configuration parameters are stored for zones added by <em>rndc addzone</em>.</td>
</tr>
<tr>
<td><em>notify-delay</em></td>
<td>Sets the delay (in seconds) between sending sets of NOTIFY messages for a zone.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent during normal zone maintenance operations.</td>
</tr>
<tr>
<td>parent-propagation-delay</td>
<td>Sets the propagation delay from the time the parent zone is updated to when the new version is served by all of the parent zone’s name servers.</td>
</tr>
<tr>
<td>parental-agents</td>
<td>Defines a list of delegation agents to be used by primary and secondary zones.</td>
</tr>
<tr>
<td>primaries</td>
<td>Defines one or more primary servers for a zone.</td>
</tr>
<tr>
<td>response-policy</td>
<td>Specifies response policy zones for the view or among global options.</td>
</tr>
<tr>
<td>serial-update-method</td>
<td>Specifies the update method to be used for the zone serial number in the SOA record.</td>
</tr>
<tr>
<td>server-addresses</td>
<td>Specifies a list of IP addresses to which queries should be sent in recursive resolution for a static-stub zone.</td>
</tr>
<tr>
<td>server-names</td>
<td>Specifies a list of domain names of name servers that act as authoritative servers of a static-stub zone.</td>
</tr>
<tr>
<td>startup-notify-rate</td>
<td>Specifies the rate at which NOTIFY requests are sent when the name server is first starting, or when new zones have been added.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the kind of zone in a given configuration.</td>
</tr>
<tr>
<td>type forward</td>
<td>Contains forwarding statements that apply to queries within a given domain.</td>
</tr>
<tr>
<td>type hint</td>
<td>Contains the initial set of root name servers to be used at BIND 9 startup.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type mirror</td>
<td>Contains a DNSSEC-validated duplicate of the main data for a zone.</td>
</tr>
<tr>
<td>type primary</td>
<td>Contains the main copy of the data for a zone.</td>
</tr>
<tr>
<td>type redirect</td>
<td>Contains information to answer queries when normal resolution would return NXDOMAIN.</td>
</tr>
<tr>
<td>type secondary</td>
<td>Contains a duplicate of the data for a zone that has been transferred from a primary server.</td>
</tr>
<tr>
<td>type static-stub</td>
<td>Contains a duplicate of the NS records of a primary zone, but statically configured rather than transferred from a primary server.</td>
</tr>
<tr>
<td>type stub</td>
<td>Contains a duplicate of the NS records of a primary zone.</td>
</tr>
<tr>
<td>zero-no-soa-ttl</td>
<td>Specifies whether to set the time to live (TTL) of the SOA record to zero, when returning authoritative negative responses to SOA queries.</td>
</tr>
<tr>
<td>zero-no-soa-ttl-cache</td>
<td>Sets the time to live (TTL) to zero when caching a negative response to an SOA query.</td>
</tr>
<tr>
<td>zone-propagation-delay</td>
<td>Sets the propagation delay from the time a zone is first updated to when the new version of the zone is served by all secondary servers.</td>
</tr>
<tr>
<td>zone-statistics</td>
<td>Controls the level of statistics gathered for all zones.</td>
</tr>
</tbody>
</table>
8.4.9 Deprecated Tag Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avoid-v4-udp-ports</td>
<td>Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv4 messages.</td>
</tr>
<tr>
<td>avoid-v6-udp-ports</td>
<td>Specifies the range(s) of ports to be excluded from use as sources for UDP/IPv6 messages.</td>
</tr>
<tr>
<td>dialup</td>
<td>Concentrates zone maintenance so that all transfers take place once every heartbeat-interval, ideally during a single call.</td>
</tr>
<tr>
<td>dnssec-must-be-secure</td>
<td>Defines hierarchies that must or may not be secure (signed and validated).</td>
</tr>
<tr>
<td>heartbeat-interval</td>
<td>Sets the interval at which the server performs zone maintenance tasks for all zones marked as dialup.</td>
</tr>
<tr>
<td>managed-keys</td>
<td></td>
</tr>
<tr>
<td>max-zone-ttl</td>
<td>Specifies a maximum permissible time-to-live (TTL) value, in seconds.</td>
</tr>
<tr>
<td>sortlist</td>
<td>Controls the ordering of RRs returned to the client, based on the client's IP address.</td>
</tr>
<tr>
<td>trusted-keys</td>
<td></td>
</tr>
<tr>
<td>use-v4-udp-ports</td>
<td>Specifies a list of ports that are valid sources for UDP/IPv4 messages.</td>
</tr>
<tr>
<td>use-v6-udp-ports</td>
<td>Specifies a list of ports that are valid sources for UDP/IPv6 messages.</td>
</tr>
</tbody>
</table>

8.5 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.

Incoming Zone Transfers
Information about in-progress incoming zone transfers.

This section describes the information, which can be seen in the HTML table about in-progress incoming zone transfers. It lists the meaning, units and possible range of values of each column, and the key/attribute/element name (in parentheses) for the JSON and XML output formats.

Zone Name (name)
Text string. This is the name of the zone being transferred, as specified in the zone declaration on this server.

Zone Type (type)
Text string. This is the type of zone being transferred, as specified in the zone declaration on this server. Possible values are: secondary, stub, redirect, mirror.

Local Serial (serial)
32 bit unsigned Integer. This is the current (old) serial number of the zone being transferred. It comes from the SOA record held on the current server.

Remote Serial (remoteserial)
32 bit unsigned Integer. This is the new serial number of the zone being transferred. It comes from the SOA record held on the primary server from which the zone is being transferred.

IXFR (ixfr)
Boolean. This says whether the transfer is incremental (using IXFR) or full (using AXFR). Possible values are: Yes, No.

State (state)
Text string. This is the current state of the transfer for this zone. Possible values and their meanings are:

Needs Refresh
The zone needs a refresh, but the process hasn’t started yet, which can be due to different factors, like the retry interval of the zone.

Pending
The zone is flagged for a refresh, but the process is currently in the queue and will start shortly, or is in a waiting state because of rate-limiting, see serial-query-rate. The Duration (s) timer starts before entering this state.

Refresh SOA
Sending a refresh SOA query to get the zone serial number, then initiate a zone transfer, if necessary. If this step is successful, the SOA Query and Got SOA states will be skipped. Otherwise, the zone transfer procedure can still be initiated, and the SOA request will be attempted using the same transport as the zone transfer. The Duration (s) timer restarts before entering this state, and for each attempted connection (note that in UDP mode there can be several retries during one “connection” attempt).

Deferred
The zone is going to be refreshed, but the process was deferred due to quota, see transfers-in and transfers-per-ns. The Duration (s) timer restarts before entering this state.

SOA Query
Sending SOA query to get the zone serial number, then follow with a zone transfer, if necessary. The Duration (s) timer restarts before entering this state.

Got SOA
An answer for the SOA query from the previous step is received, initiating a transfer.
Zone Transfer Request
Waiting for the zone transfer to start. The Duration (s) timer restarts before entering this state.

First Data
Waiting for the first data record of the transfer.

Receiving IXFR Data
Receiving data for an IXFR type incremental zone transfer.

Finalizing IXFR
Finalizing an IXFR type incremental zone transfer.

Receiving AXFR Data
Receiving data for an AXFR type zone transfer.

Finalizing AXFR
Finalizing an AXFR type zone transfer.

Note: State names can change between BIND versions.

Additional Refresh Queued(refreshqueued)
Boolean. This shows that the zone is flagged for a refresh. This can be set to Yes either when the zone transfer is still in one of the pending states (see the description of the State column), or when the transfer is in a running state, but the zone was marked for another refresh again (e.g. because of “notify” request from a primary server). Possible values are: Yes, No.

Local Address(localaddr)
IP address - IPv4 or IPv6, as appropriate, and port number. This shows the source address used to establish the connection for the transfer.

Remote Address(remoteaddr)
IP address - IPv4 or IPv6, as appropriate, and port number. This shows the destination address used to establish the connection for the transfer.

SOA Transport(soatransport)
Text string. This is the transport protocol in use for the SOA query. Note, that this value can potentially change during the process. For example, when the transfer is in the Refresh SOA state, the SOA Transport of the ongoing query can be shown as UDP. If that query fails or times out, it then can be retried using another transport, or the transfer process can be initiated in “SOA before” mode, where the SOA query will be attempted using the same transport as the zone transfer. See the description of the State field for more information. Possible values are: UDP, TCP, TLS, None.

Transport(transport)
Text string. This is the transport protocol in use for the transfer. Possible values are: TCP, TLS.

TSIG Key Name(tsigkeyname)
Text string. This is the name of the TSIG key specified for use with this zone in the zone declaration (if any).

Duration (s)(duration)
64 bit unsigned Integer. This is the time, in seconds, that the current major state of the transfer process has been running so far. The timer starts after the refresh SOA request is queued (before the Pending state), then it restarts several times during the whole process to indicate the duration of the current major state. See the descriptions of the different states to find out the states, before which this timer restarts.

Messages Received(nmsg)
64 bit unsigned Integer. This is the number of DNS messages received. It does not include transport overheads, such as TCP ACK.
Records Received (**nrecs**)

64 bit unsigned Integer. This is the number of individual RRs received so far. If an address record has, for example, five addresses associated with the same name, it counts as five RRs.

Bytes Received (**nbytes**)

64 bit unsigned Integer. This is the number of usable bytes of DNS data. It does not include transport overhead.

**Note:** Depending on the current state of the transfer, some of the values may be empty or set to – (meaning “not available”). Also, in the case of the JSON output format, the corresponding keys can be missing or values can be set to NULL. For example, it isn’t known whether a transfer is using AXFR or IXFR until the first data is received (see the description of the **State** column).

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** 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.
8.5.1 The Statistics File

The text format statistics dump begins with a line, like:

```plaintext
+++ 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:

```plaintext
++ 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:

```plaintext
--- Statistics Dump --- (973798949)
```

8.5.2 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).

- **ReqBadEDNSver**
  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 referral 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 dropped by the server as a result of configured limits. These limits include the settings of the fetches-per-zone, fetches-per-server, clients-per-query, and max-clients-per-query options, as well as the rate-limit option. 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 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.

**UpdateQuota**  
This indicates the number of times a dynamic update or update forwarding request was rejected because the number of pending requests exceeded *update-quota*.

**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.

QueryCurUDP
This indicates the number of UDP queries in progress.

QueryCurTCP
This indicates the number of TCP queries in progress.

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.

NumFetch
This indicates the number of active fetches.

BucketSize
This indicates the number the resolver’s internal buckets (a static number).

REFUSED
This indicates the number of REFUSED responses received.
ClientCookieOut
This indicates the number of COOKIE sent with client cookie only.

ServerCookieOut
This indicates the number of COOKIE sent with client and server cookie.

CookieIn
This indicates the number of COOKIE replies received.

CookieClientOk
This indicates the number of COOKIE client ok.

BadEDNSVersion
This indicates the number of bad EDNS version replies received.

BadCookieRcode
This indicates the number of bad cookie rcode replies received.

ZoneQuota
This indicates the number of queries spilled due to zone quota.

ServerQuota
This indicates the number of queries spilled due to server quota.

ClientQuota
This indicates the number of queries spilled due to clients per query quota.

NextItem
This indicates the number of waits for next item, when an invalid response is received.

Priming
This indicates the number of priming fetches performed by the resolver.

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), and TCP6 (TCP/IPv6). 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.

<Type>OpenFail
This indicates the number of failures to open sockets.

<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 type.
<TYPE>Accept
This indicates the number of incoming connections successfully accepted. This counter does not apply to the UDP type.

<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.
CHAPTER
NINE

TROUBLESHOOTING

9.1 Common Problems

9.1.1 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.

9.1.2 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.
9.1.3 Inspecting Encrypted DNS Traffic

**Note:** This feature requires support from the cryptographic library that BIND 9 is built against. For OpenSSL, version 1.1.1 or newer is required (use `named -V` to check).

By definition, TLS-encrypted traffic (e.g. DNS over TLS, DNS over HTTPS) is opaque to packet sniffers, which makes debugging problems with encrypted DNS close to impossible. However, Wireshark offers a solution to this problem by being able to read key log files. In order to make named prepare such a file, set the `SSLKEYLOGFILE` environment variable to either:

- the string `config` (`SSLKEYLOGFILE=config`); this requires defining a logging channel which will handle messages belonging to the `sslkeylog` category,
- the path to the key file to write (`SSLKEYLOGFILE=/path/to/file`); this is equivalent to the following logging configuration:

  ```
  channel default_sslkeylogfile {
    file "${SSLKEYLOGFILE}" versions 10 size 100m suffix timestamp;
  };
  category sslkeylog {
    default_sslkeylogfile;
  };
  ```

**Note:** When using `SSLKEYLOGFILE=config`, augmenting the log channel output using options like `print-time` or `print-severity` is strongly discouraged as it will likely make the key log file unusable.

When the `SSLKEYLOGFILE` environment variable is set, each TLS connection established by named (both incoming and outgoing) causes about 1 kilobyte of data to be written to the key log file.

**Warning:** Due to the limitations of the current logging code in BIND 9, enabling TLS pre-master secret logging adversely affects named performance.

9.2 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.
9.3 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.
To build on a Unix or Linux system, use:

```
$ autoreconf -fi ### (only if building from the git repository)
$ ./configure
$ make
```

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>The C compiler to use. <code>configure</code> tries to figure out the right one for supported systems.</td>
</tr>
<tr>
<td>CFLAGS</td>
<td>The C compiler flags. Defaults to include <code>-g</code> and/or <code>-O2</code> as supported by the compiler. Please include <code>-g</code> if <code>CFLAGS</code> needs to be set.</td>
</tr>
<tr>
<td>LD-FLAGS</td>
<td>The linker flags. Defaults to an empty string.</td>
</tr>
</tbody>
</table>

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 using Emacs, the `make tags` command may be helpful.

### 10.1 Required Libraries

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

- a C11-compliant compiler
- `libcrypto`, `libssl`
- `liburcu`
- `libuv`
- `perl`
- `pkg-config`, `pkgconfig`, `pkgconf`

BIND 9.20 requires `libuv 1.34.0` or higher, using `libuv >= 1.40.0` is recommended. Compiling or running with `libuv 1.35.0` or `1.36.0` is not supported, as this could lead to an assertion failure in the UDP receive code. 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. To use a PKCS#11 hardware service module for cryptographic operations, engine_pkcs11 from the OpenSC project must be compiled and used.

The Userspace RCU library liburcu (https://liburcu.org/) is used for lock-free data structures and concurrent safe memory reclamation.

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.

To build BIND from the git repository, the following tools must also be installed:

* autoconf (includes autoreconf)
* automake
* libtool

### 10.2 Optional Features

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

To improve performance, use of the jemalloc library (https://jemalloc.net/) is strongly recommended. Version 4.0.0 or newer is required when in use.

To support DNS over HTTPS (DoH), the server must be linked with libnghttp2 (https://nghttp2.org/). If the library is unavailable, --disable-doh can be used to disable DoH support.

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.
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. The `--enable-singletrace` option turns on the same verbose tracing, but allows an individual query to be separately traced by setting its query ID to 0. These options should only be enabled when debugging, because they have 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`.

### 10.3 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.)
Contents

• Release Notes
  – Introduction
  – Supported Platforms
  – Download
  – Known Issues
  – Notes for BIND 9.20.0
    ∗ New Features
    ∗ Removed Features
    ∗ Deprecated Features
    ∗ Feature Changes
    ∗ Bug Fixes
    ∗ Known Issues
  – License
  – End of Life
  – Thank You
11.1 Introduction

BIND 9.20 is a stable branch, suitable for production use. This document summarizes significant changes since the last production release on the 9.18 branch.

11.2 Supported Platforms

See the Supported Platforms section in the Resource Requirements chapter.

11.3 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, and source code.

11.4 Known Issues

- On some platforms, including FreeBSD, named must be run as root to use the rndc control channel on a privileged port (i.e., with a port number less than 1024; this includes the default rndc port, 953). Currently, using the named -u option to switch to an unprivileged user makes rndc unusable. This will be fixed in a future release; in the meantime, mac_portacl can be used as a workaround, as documented in https://kb.isc.org/docs/aa-00621. [GL #4793]

11.5 Notes for BIND 9.20.0

Note: This section only lists changes since BIND 9.18.28, the most recent release on the previous stable branch of BIND at the time of the publication of BIND 9.20.0.

11.5.1 New Features

- The forwarders statement now supports the tls argument, to be used to forward queries to DoT-enabled servers. [GL #3726]
- named now supports forwarding Dynamic DNS updates through DNS-over-TLS (DoT). [GL #3512]
- The nsupdate tool now supports DNS-over-TLS (DoT). [GL #6752]
- The tls block was extended with a new cipher-suites option that allows permitted cipher suites for TLSv1.3 to be set. Please consult the documentation for additional details. [GL #3504]
- Initial support for the PROXYv2 protocol was added. named can now accept PROXYv2 headers over all currently implemented DNS transports and dig can insert these headers into the queries it sends. Please consult the related documentation (allow-proxy, allow-proxy-on, listen-on, and listen-on-v6 for named, dig +proxy and dig +proxy-plain for dig) for additional details. [GL #4388]
• The client-side support of the EDNS EXPIRE option has been expanded to include IXFR and AXFR query types. This enhancement enables named to perform AXFR and IXFR queries while incorporating the EDNS EXPIRE option. [GL #4170]

• A new configuration option require-cookie has been introduced. It specifies whether there should be a DNS COOKIE in the response for a given prefix; if not, named falls back to TCP. This is useful if it is known that a given server supports DNS COOKIE. It can also be used to force all non-DNS COOKIE responses to fall back to TCP. [GL #2295]

• The check-svcb option has been added to control the checking of additional constraints on SVCB records. This change affects named, named-checkconf, named-checkzone, named-compilezone, and nsupdate. [GL #3576]

• The new resolver-use-dns64 option enables named to apply dns64 rules to IPv4 server addresses when sending recursive queries, so that resolution can be performed over a NAT64 connection. [GL #608]

• A new option to dnssec-policy has been added, cdnskey, that allows users to enable or disable the publication of CDNSKEY records. [GL #4050]

• When using dnssec-policy, it is now possible to configure the digest type to use when CDS records need to be published with cds-digest-types. Also, publication of specific CDNSKEY/CDS records can now be set with dnssec-signzone -G. [GL #3837]

• Support for multi-signer model 2 (RFC 8901) when using inline-signing was added. [GL #2710]

• HSM support was added to dnssec-policy. Keys can now be configured with a key-store that allows users to set the directory where key files are stored and to set a PKCS#11 URI string. The latter requires OpenSSL 3 and a valid PKCS#11 provider to be configured for OpenSSL. [GL #1129]

• A new DNSSEC tool dnssec-ksr has been added to create Key Signing Request (KSR) and Signed Key Response (SKR) files. [GL #1128]

• dnssec-verify and dnssec-signzone now accept a -J option to specify a journal file to read when loading the zone to be verified or signed. [GL #2486]

• dnssec-keygen now allows the options -k and -f to be used together. This allows the creation of keys for a given dnssec-policy that match only the KSK (-fK) or ZSK (-fZ) roles. [GL #1128]

• The response-policy statement was extended with a new argument ede. It enables an RFC 8914 Extended DNS Error (EDE) code of choice to be set for responses which have been modified by a given RPZ. [GL #3410]

• A new way of configuring the preferred source address when talking to remote servers, such as primaries and parental-agents, has been added: setting the source and/or source-v6 arguments for a given statement is now possible. This new approach is intended to eventually replace statements such as parental-source, parental-source-v6, transfer-source, etc. [GL #3762]

• The new command-line delv +ns option activates name server mode, to more accurately reproduce the behavior of named when resolving a query. In this mode, delv uses an internal recursive resolver rather than an external server. All messages sent and received during the resolution and validation process are logged. This can be used in place of dig +trace. [GL #3842]

• The read timeout in rndc can now be specified on the command line using the -t option, allowing commands that take a long time to complete sufficient time to do so. [GL #4046]

• The statistics channel now includes information about incoming zone transfers that are currently in progress. [GL #3883]

• Information on incoming zone transfers in the statistics channel now also shows the zones’ “first refresh” flag, which indicates that a zone is not fully ready and that its first ever refresh is pending or is in progress. The number of such zones is now also exposed by the rndc status command. [GL #4241]
• Added a new statistics variable `recursive high-water` that reports the maximum number of simultaneous recursive clients BIND has handled while running. [GL #4668]

• A new command, `rndc fetchlimit`, prints a list of name server addresses that are currently rate-limited due to `fetches-per-server` and domain names that are rate-limited due to `fetches-per-zone`. [GL #665]

• Queries and responses now emit distinct `dnstap` entries for DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH), and `dnstap-read` understands these entries. [GL #4523]

• `dnstap-read` can now print long timestamps with millisecond precision. [GL #2360]

• Support for libsystemd's `sd_notify()` function was added, enabling `named` to report its status to the init system. This allows systemd to wait until `named` is fully ready before starting other services that depend on name resolution. [GL #1176]

• Support for User Satically Defined Tracing (USDT) probes has been added. These probes enable fine-grained application tracing and introduce no overhead when they are not enabled. [GL #4041]

11.5.2 Removed Features

• Support for Red Hat Enterprise Linux version 7 (and clones) has been dropped. A C11-compliant compiler is now required to compile BIND 9. [GL #3729]

• Compiling with `jemalloc` versions older than 4.0.0 is no longer supported; those versions do not provide the features required by current BIND 9 releases. [GL #4296]

• The `auto-dnssec` configuration statement has been removed. Please use `dnssec-policy` or manual signing instead. See article how to migrate from `auto-dnssec` to `dnssec-policy`.

The following statements have become obsolete: `dnskey-sig-validity`, `dnssec-dnskey-kskonly`, `dnssec-update-mode`, `sig-validity-interval`, and `update-check-ksk`. [GL #3672]

• Dynamic updates that add and remove DNSKEY and NSEC3PARAM records no longer trigger key rollovers and denial-of-existence operations. This also means that the `dnssec-secure-to-insecure` option has been obsoleted. [GL #3686]

• The `glue-cache` option has been removed. The glue cache `feature` still works and is now permanently enabled. [GL #2147]

• Configuring the control channel to use a Unix domain socket has been a fatal error since BIND 9.18. The feature has now been completely removed and `named-checkconf` now reports it as a configuration error. [GL #4311]

• The statements setting alternate local addresses for inbound zone transfers (`alt-transfer-source`, `alt-transfer-source-v6`, and `use-alt-transfer-source`) have been removed. [GL #3714]

• The `resolver-nonbackoff-tries` and `resolver-retry-interval` statements have been removed. Using them is now a fatal error. [GL #4405]

• BIND 9 no longer supports non-zero `stale-answer-client-timeout` values, when the feature is turned on. When using a non-zero value, `named` now generates a warning log message, and treats the value as 0. [GL #4447]

• The Differentiated Services Code Point (DSCP) feature has been removed: configuring DSCP values in `named.conf` is now a configuration error. [GL #3789]

• The `keep-response-order` option has been declared obsolete and the functionality has been removed. `named` expects DNS clients to be fully compliant with RFC 7766. [GL #3140]

• Zone type `delegation-only`, and the `delegation-only` and `root-delegation-only` statements, have been removed. Using them is a configuration error.
These statements were created to address the SiteFinder controversy, in which certain top-level domains redirected misspelled queries to other sites instead of returning NXDOMAIN responses. Since top-level domains are now DNSSEC-signed, and DNSSEC validation is active by default, the statements are no longer needed. [GL #3953]

- The `coresize`, `datasize`, `files`, and `stacksize` options have been removed. The limits these options set should be enforced externally, either by manual configuration (e.g., using `ulimit`) or via the process supervisor (e.g., `systemd`). [GL #3676]

- Support for using AES as the DNS COOKIE algorithm (`cookie-algorithm aes;`) has been removed. The only supported DNS COOKIE algorithm is now the current default, SipHash-2-4. [GL #4421]

- The TKEY Mode 2 (Diffie-Hellman Exchanged Keying Mode) has been removed and using TKEY Mode 2 is now a fatal error. Users are advised to switch to TKEY Mode 3 (GSS-API). [GL #3905]

- Special-case code that was originally added to allow GSS-TSIG to work around bugs in the Windows 2000 version of Active Directory has now been removed, since Windows 2000 is long past end-of-life. The `-c` option and the `oldgsstsig` command to `nsupdate` have been deprecated, and are now treated as synonyms for `-g` and `gsstsig` respectively. [GL #4012]

- Support for the `lock-file` statement and the `named -X` command-line option has been removed. An external process supervisor should be used instead. [GL #4391]

  Alternatively, the `flock` utility (part of `util-linux`) can be used on Linux systems to achieve the same effect as `lock-file` or `named -X`:

  ```
  flock -n -x <directory>/named.lock <path>/named <arguments>
  ```

- The `named` command-line option `-U`, which specified the number of UDP dispatches, has been removed. Using it now returns a warning. [GL #1879]

- The `--with-tuning` option for `configure` has been removed. Each of the compile-time settings that required different values based on the “workload” (which were previously affected by the value of the `--with-tuning` option) has either been removed or changed to a sensible default. [GL #3664]

- The functions that were in the `libbind9` shared library have been moved to the `libisc` and `libisccfg` libraries. The now-empty `libbind9` has been removed and is no longer installed. [GL #3903]

- The `irs_resconf` module has been moved to the `libdns` shared library. The now-empty `libirs` library has been removed and is no longer installed. [GL #3904]

### 11.5.3 Deprecated Features

Features listed in this section still work but are scheduled for eventual removal.

- The use of the `max-zone-ttl` option in `options` and `zone` blocks has been deprecated; it should now be configured as part of `dnssec-policy`. A warning is logged if this option is used in `options` or `zone` blocks. In a future release, it will become nonoperational. [GL #2918]

- The `sortlist` option has been deprecated and will be removed in a future BIND 9.21.x release. Users should not rely on a specific order of resource records in DNS messages. [GL #4593]

- The `fixed` value for the `rrset-order` option and the corresponding `configure` script option have been deprecated and will be removed in a future BIND 9.21.x release. Users should not rely on a specific order of resource records in DNS messages. [GL #4446]
11.5.4 Feature Changes

- BIND now depends on liburcu, Userspace RCU, for lock-free data structures. [GL #3934]
- On Linux, libcap is now a required dependency to help named keep needed privileges. [GL #3583]
- Compiling BIND 9 now requires at least libuv version 1.34.0 or higher. libuv should be available on all supported platforms either as a native package or as a backport. [GL #3567]
- Outgoing zone transfers are no longer enabled by default. An explicit allow-transfer ACL must now be set at the zone, view, or options level to enable outgoing transfers. [GL #4728]
- DNS zones signed using dnssec-policy now automatically detect their parent servers, and BIND queries them to check the content of the DS RRset. This allows DNSSEC key rollovers to safely and automatically proceed when the parent zone is updated with new DNSSEC keys, i.e. using the CDS/CDNSKEY mechanism. This behavior is facilitated by the new checks feature, which automatically populates parental-agents by resolving the parent NS records. These parent name servers are queried to check the DS RRset during a KSK rollover initiated by dnssec-policy. [GL #3901]
- The responsiveness of named was improved, when serving as an authoritative DNS server for a delegation-heavy zone(s) shortly after loading such zone(s). [GL #4045]
- To improve query-processing latency under load, the uninterrupted time spent on resolving long chains of cached domain names has been reduced. [GL #4185]
- QNAME minimization is now used when looking up the addresses of name servers during the recursive resolution process. [GL #4209]
- BIND now returns BADCOOKIE for out-of-date or otherwise bad but well-formed DNS server cookies. [GL #4194]
- The DNS name compression algorithm used in BIND 9 has been revised: it now compresses more thoroughly than before, so responses containing names with many labels might have a smaller encoding than before. [GL #3661]
- Processing large incremental transfers (IXFR) has been offloaded to a separate work thread so that it does not prevent networking threads from processing regular traffic in the meantime. [GL #4367]
- Querying the statistics channel no longer blocks DNS communication on the networking event loop level. [GL #4680]
- The inline-signing zone option is now ignored if there is no dnssec-policy configured for the zone. This means that unsigned zones no longer create redundant signed versions of the zone. [GL #4349]
- The inline-signing statement can now also be set inside dnssec-policy. The built-in policies default and insecure enable the use of inline-signing. If inline-signing is set at the zone level, it overrides the value set in dnssec-policy. [GL #3677]
- Following RFC 9276 recommendations, dnssec-policy now only allows an NSEC3 iteration count of 0 for the DNSSEC-signed zones using NSEC3 that the policy manages. [GL #4363]
- The maximum number of NSEC3 iterations allowed for validation purposes has been lowered from 150 to 50. DNSSEC responses containing NSEC3 records with iteration counts greater than 50 are now treated as insecure. [GL #4363]
- The dnssec-validation yes option now requires an explicitly configured trust-anchors statement. If using manual trust anchors is not operationally required, then please consider using dnssec-validation auto instead. [GL #4373]
- named-compilezone no longer performs zone integrity checks by default; this allows faster conversion of a zone file from one format to another. [GL #4364]

Zone checks can be performed by running named-checkzone separately, or the previous default behavior can be restored by using:
The red-black tree data structure used in the RBTDB (the default database implementation for cache and zone databases), has been replaced with QP-tries. This is expected to improve performance and scalability, though in the current implementation large zones require roughly 15% more memory than the old red-black tree data structure.

A side effect of this change is that zone files that are created with `masterfile-style relative` for example, the output of `dnssec-signzone` will no longer have multiple different `$ORIGIN` statements. There should be no other changes to server behavior.

The old RBT-based database still exists for now, and can be used by specifying `database rbt` in a `zone` statement in `named.conf`, or by compiling with `configure --with-zonedb=rbt --with-cachedb=rbt`. [GL #4411] [GL #4614]

Multiple RNDC messages are now processed when sent in a single TCP message.

ISC would like to thank Dominik Thalhammer for reporting the issue and preparing the initial patch. [GL #4416]

The DNSSEC signing data included in zone statistics identified keys only by the key ID; this caused confusion when two keys using different algorithms had the same ID. Zone statistics now identify keys using the algorithm number, followed by “+”, followed by the key ID: for example, 8+54274. [GL #3525]

The TTL of the NSEC3PARAM record for every NSEC3-signed zone was previously set to 0. It is now changed to match the SOA MINIMUM value for the given zone. [GL #3570]

On startup, `named` now sets the limit on the number of open files to the maximum allowed by the operating system, instead of trying to set it to “unlimited”. [GL #3676]

When an international domain name is not valid according to IDNA2008, `dig` now tries to convert it according to IDNA2003 rules, or pass it through unchanged, instead of stopping with an error message. The `idna2` utility can be used to check IDNA syntax. [GL #3527]

The memory statistics have been reduced to a single counter, `InUse;Malloced` is an alias that holds the same value. The other counters were usable with the old BIND 9 internal memory allocator, but they are unnecessary now that the latter has been removed. [GL #3718]

The log message `resolver priming query complete` has been moved from the INFO log level to the DEBUG(1) log level, to prevent `delv` from emitting that message when setting up its internal resolver. [GL #3842]

Worker threads’ event loops are now managed by a new “loop manager” API, significantly changing the architecture of the task, timer, and networking subsystems for improved performance and code flow. [GL #3508]

The code for DNS over TCP and DNS over TLS transports has been replaced with a new, unified transport implementation. [GL #3374]

### 11.5.5 Bug Fixes

- When the same `notify-source` address and port number was configured for multiple destinations and zones, an unresponsive server could tie up the relevant network socket until it timed out; in the meantime, NOTIFY messages for other servers silently failed. `named` will now retry sending such NOTIFY messages over TCP. Furthermore, NOTIFY failures are now logged at the INFO level. [GL #4001] [GL #4002]

- DNS compression is no longer applied to the root name (.) if it is repeatedly used in the same RRset. [GL #3423]

- `named` could incorrectly return non-truncated, glueless referrals for responses whose size was close to the UDP packet size limit. This has been fixed. [GL #1967]
11.5.6 Known Issues

- On some platforms, including FreeBSD, named must be run as root to use the \texttt{rndc} control channel on a privileged port (i.e., with a port number less than 1024; this includes the default \texttt{rndc port}, 953). Currently, using the \texttt{named -u} option to switch to an unprivileged user makes \texttt{rndc} unusable. This will be fixed in a future release; in the meantime, \texttt{mac_portacl} can be used as a workaround, as documented in \url{https://kb.isc.org/docs/aa-00621}. [GL #4793]
- See \textit{above} for a list of all known issues affecting this BIND 9 branch.

11.6 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 \url{https://www.isc.org/contact/}.

11.7 End of Life

BIND 9.20 is a stable branch, suitable for production use. After it has been in production use for a while it will be designated as an Extended Support Version (ESV). Until then, the current ESV is BIND 9.18, which will be supported until at least December 2025. See \url{https://kb.isc.org/docs/aa-00896} for details of ISC’s software support policy.

11.8 Thank You

Thank you to everyone who assisted us in making this release possible.
12.1 Preface

12.1.1 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.

12.1.2 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. Ihnen, J. Dickinson, and W. Mekking, subsequently published as RFC 7583.
12.2 Introduction

12.2.1 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.

12.2.2 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.

12.2.3 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.

## 12.2.4 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](#).

### 12.2.5 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.

### 12.2.6 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), of 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.

### 12.2.7 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.

### 12.2.8 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. 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

---

1 The Office of Management and Budget (OMB) for the US government published a [memo in 2008](https://www.whitehouse.gov/the-press-office/2008/09/27/memorandum-for-chairman-ipc-and-secretary-omb-establishing-policy-dnssec), 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.
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.

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.

12.2.9 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.

12.3 Getting Started

12.3.1 Software Requirements

This guide assumes BIND 9.18.0 or newer, although the more elaborate manual procedures do work with all versions of BIND later than 9.9.

We recommend running the latest stable version to get the most complete DNSSEC configuration, as well as the latest security fixes.

12.3.2 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. Contrary to popular belief, the increase in resource consumption is very modest:

1. CPU: a validating resolver executes cryptographic functions on cache-miss answers, which leads to increased CPU usage. Thanks to standard DNS caching and contemporary CPUs, the increase in CPU-time consumption in a steady state is negligible - typically on the order of 5%. For a brief period (a few minutes) after the resolver starts, the increase might be as much as 20%, but it quickly decreases as the DNS cache fills in.

2. System memory: DNSSEC leads to larger answer sets and occupies more memory space. With typical ISP traffic and the state of the Internet as of mid-2022, memory consumption for the cache increases by roughly 20%.

3. Network interfaces: although DNSSEC does increase the amount of DNS traffic overall, in practice this increase is often within measurement error.
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 expected changes to resource consumption caused by serving DNSSEC-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. The final size depends on the structure of the zone, the signing algorithm, the number of keys, the choice of NSEC or NSEC3, the ratio of signed delegations, the zone file format, etc. Usually, the size of a signed zone ranges from a negligible increase to as much as three times the size of the unsigned zone.

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. The final memory consumption also depends on all the variables listed above: in the typical case the increase is around half of the unsigned zone memory consumption, but it can be as high as three times for some corner cases.

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. As of mid-2022, measurements by APNIC show 41% of Internet users send DNSSEC-aware queries. 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.

### 12.3.3 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.

### 12.3.4 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.

12.4 Validation

12.4.1 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:

```plaintext
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, 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. 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.

---

2 Of course, something like this could happen for reasons other than DNSSEC: for example, the root publishing the wrong addresses for the .org nameservers.
12.4.2 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 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.

**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)
;; 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 133 3 300 { 20200401191851 20200302184340 27566 isc.org. e9Vkb6/6aHMqk/t23m71oi0Du0B6sncduoW9+As14 L3TZtpLvZ5+zudTJC2coI4D/D9AXte1cD6FV6iS6PQ== }
```

(continues on next page)
The important detail in this output is the presence of the \texttt{ad} flag in the header. This signifies that BIND has retrieved all related DNSSEC information related to the target of the query (\texttt{ftp.isc.org}) and that the answer received has passed the validation process described in \textit{How Are Answers Verified?}. We can have confidence in the authenticity and integrity of the answer, that \texttt{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 \texttt{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 \texttt{dnssec-validation no;} to the “options” section of the configuration file, i.e.:

\begin{verbatim}
options {
    ...
    dnssec-validation no;
    ...
};
\end{verbatim}

If the server is restarted (to ensure a clean cache) and the same \texttt{dig} command executed, the result is very similar:

\begin{verbatim}
$ dig @10.53.0.1 ftp.isc.org. A +dnssec +multiline
;; 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

;; 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/t23Im71ioiDUhB06sncoduW9+Aa14 L3TZtpLwZ5+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
\end{verbatim}

However, this time there is no \texttt{ad} flag in the header. Although \texttt{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 \texttt{dnssec-validation no;} line from the configuration file) before continuing.
### 12.4.3 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:

```bash
$ 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: 69c30831485458701000005e67bb57f5f90ff2688e455d (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:

```bash
$ 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: 251eee58208917f9010000005e67bb6829f6dabc5ae6b7b9 (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 successfully 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 `dig +cd` flag. If the query succeeds with `dig +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*.

### 12.4.4 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 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.

When using yes, please note that if trust-anchors does not include a valid root key, then validation does not take place for names which are not covered by any of the configured trust anchors.

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.

---

3 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.
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.

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.

![Signature Generation](image)

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.
```

(continues on next page)
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 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.
12.4.5 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.

12.4.6 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, and automatically updates it when the root key changes. It looks something like this:

```bash
trust-anchors {
    # This key (20326) was published in the root zone in 2017.
    . initial-key 257 3 "AwEAAaz/
    tAm8yTn4Mfeh5ey196WSexTBAvkMgJzkkKT0iW1vkbzxeF3
    /+4/RgWq7HrxRixH1FxO1Ze15emLvN7SWXgnLh4+B5xQ1NVz8Og88k
    ArMtNRoxVQuc5n1DD5LKycWrbRd2n9WGe2R8PzgCmr3EqVLrjyBxWezF
    0jLhwVNEfS3rcj/EWgvIWgb9tarpVUDK/b58Da-sqqls3eNbv7pr+e
    oZG+SrDF6nWzl3c565Apzx7LjVc1uTI4sIXxuOLYA4/i1BmSVIzuDWfd
    RUHdY6+cn8HFRm=2hM8AnXGxws9555KrUB5qihylGa8subX2N6UwN
    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`:

```bash
options {
    dnssec-validation yes;
};
```

Then, download the root key manually from a trustworthy source, and put it into a `trust-anchors` statement as shown below:

```bash
trust-anchors {
    # This key (20326) was published in the root zone in 2017.
    . static-key 257 3 "AwEAAaz/tAm8yTn4Mfeh5ey196WSexTBAvkMgJzkkKT0iW1vkbzxeF3
    /+4/RgWq7HrxRixH1FxO1Ze15emLvN7SWXgnLh4+B5xQ1NVz8Og88k
    ArMtNRoxVQuc5n1DD5LKycWrbRd2n9WGe2R8PzgCmr3EqVLrjyBxWezF
    0jLhwVNEfS3rcj/EWgvIWgb9tarpVUDK/b58Da-sqqls3eNbv7pr+e
    oZG+SrDF6nWzl3c565Apzx7LjVc1uTI4sIXxuOLYA4/i1BmSVIzuDWfd
    RUHdY6+cn8HFRm=2hM8AnXGxws9555KrUB5qihylGa8subX2N6UwN
    R1AkUTV74bU=";}
};
```

---

4 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.
While this `trust-anchors` statement looks similar to the built-in version above, the built-in key 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` modifier, 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 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.

### 12.4.7 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 `dig +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:

```bash
$ dig @10.53.0.1 www.isc.org. A +dnssec +multiline

;; 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
```

(continues on next page)
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/](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.

12.5 Signing

12.5.1 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.
Using `dnssec-policy` requires dynamic DNS or `inline-signing` to be enabled.

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 16:02:55.048 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):
We want to edit the copy to be in the `trust-anchors` format, so that it looks like this:

```bash
# cat /etc/bind/Kexample.com.+013+10376.key
...
example.com. 3600 IN DNSKEY 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:

```bash
$ 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/yu167aKQ2vFyC7BiQe1WEn1ogdLTF
og4yBQumOhO5WX61LjA1711DuLWcd/ASw1U2WFGBYQ== }
```

## 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:**
   ```text
   example.com. 3600 IN DS 10376 13 2 B92E22CAE0...33B8312EF0
   ```

2. **DNSKEY format:**

12.5. Signing
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 workflow 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.

12.5.2 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
dnssec-dsfromkey example.com
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
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
dnssec-dsfromkey example.com
example.com. 600 IN SOA ns1.example.com. admin.example.com. ( 2020040703 ; serial
1800 ; refresh (30 minutes)
900 ; retry (15 minutes)
(continues on next page)
```
2419200 ; expire (4 weeks)
300 ; minimum (5 minutes)
}

example.com. 300 IN RRSIG NSEC 13 2 300 {
  20200413050536 20200407140255 10376 example.com.
  drtV1rJbo50Mi650Jtu7Jmg/thgpdTwrrz603Pzt12+B
  oxCmAV3orWWWjfp2n9w5j0rxZ2Mt2ev7MOOG8IOUCA==
}

example.com. 300 IN NSEC ftp.example.com. NS SOA RRSIG NSEC DNSKEY TYPE65534

example.com. 600 IN RRSIG NS 13 2 600 {
  2020041310638 20200407140255 10376 example.com.
  2ipzm1Ei6vF9OLwPMsxLRBjcrPwPGWJoekwZbbux
  MLffZOxN8cl0Q12U9iCPdyoQryuCjiohH2d6nrw==
}

example.com. 600 IN RRSIG SOA 13 2 600 {
  20200421150255 20200407140255 10376 example.com.
  jbsz92zWAcGMNV/yu167aKQZvFyC7Biqe1WEnLogLTF
  og4yBQumOho5WX61lJaj171DulJcd/ASw1U2WFCYQ==
}

example.com. 0 IN RRSIG TYPE65534 13 2 0 {
  20200413050536 20200407140255 10376 example.com.
  Xjkom24N6geCJjg9BMUFuWf=euLe2Z169DvLjYFPZN1m
  GgMzrcuDPIo6VpBwU6JE5DNSjuGjgpjXC55pC42qg==
}

example.com. 3600 IN RRSIG DNSKEY 13 2 3600 {
  20200421150255 20200407140255 10376 example.com.
  mako5728oVdYtc13VwJtshugKL2ZW+q+c47Lea6kn
  Xj77kXclNogNdUor5am/606cnPeJKIWsnmTL1Sm62g==
}

example.com. 0 IN TYPE65534 \# _ ( 0D28880001 )

example.com. 3600 IN DNSKEY 257 3 13 {
  6sa1q99dBb5b4G4cx13cPjFTr1vUs3NW44SvbHorHb
  kxWozeGAWyPORN+pwEV/LP9+FAHF/JZaJYdqpo0dw==
}; KSK; alg = ECDSA256SHA256; key id = 10376

example.com. 600 IN NS ns1.example.com.

ftp.example.com. 600 IN RRSIG A 13 3 600 {
  2020041310638 20200407140255 10376 example.com.
  UYo1njeUA49VhKnPSS3J04G+/XdP2d4mVaaanced191yz
  BIouoEBAGpaczEM2BNrgR0op1EWSus9tG8SM1ZHGQ==
}

ftp.example.com. 300 IN RRSIG NSEC 13 3 300 {
  2020041310638 20200407140255 10376 example.com.
  rPAdrAMAPIPSF3S450SYkXBTYMS3nrz2g4wqj7qRL+/b
  soKyh604MBiBjq+YW69dBjKoTSeEGSCSt3uIrxyA==
}

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 {
  2020041310638 20200407140255 10376 example.com.
  Yeoij7qrJmxL6ULTnALwKU5byNldZ9ggj5XjcbpFvuj0
  ocG/ovGBg6pdugXC9uxE39bcD18dua1frjDcRCC2AA==
}

ns1.example.com. 300 IN RRSIG NSEC 13 3 300 {
  2020041310638 20200407140255 10376 example.com.
  vukgQme6k7JwCF/mjJOxHZxB3fKsro/Kc1076dHMsC
  oM1/oXio2vqB29cKrQhIAUt7r1KUNrUwM6Je36wWFA==
}

ns1.example.com. 300 IN NSEC 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 {
  2020041310638 20200407140255 10376 example.com.
  JXi4WypofD5geUoWvlgJyHvcRnsvl/ONhTBaUCw5Y
  XtifkAXHRWlU1H1w737YPLf5uVu90RfKWLjGqTQ==
}

web.example.com. 300 IN RRSIG NSEC 13 3 300 {
  2020041310638 20200407140255 10376 example.com.
  XF4Hsd58dall+s6Qu99bG80PQyMf72rHEzdIEf1RuykP
(continues on next page)
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.

### 12.5.3 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 { 6saiq99gDBb5b4G4cx13cPjTrIvUs3NW44SybbHorHb kXwOEeGAWyP0RN+pEv/LP9+FHAf/JzA3Yd0p+00d== }
; 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 now comes with at least one corresponding signature, known as an RRSIG.

```
$ dig @192.168.1.13 example.com. SOA +dnssec +multiline

;; 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: 75adff4f4ce916b2010D00005e8c99c0de47eabb7951b2f5 (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/yu167aKQZvFyC7BiQe1WEn1ogdLTF
 oq4yBQumO8Ox61LjA1711DuLWcd/ASw1UZWFQCYQ== )
```

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

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.
• .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.

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.

12.5.4 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.

dnssec-policy is the preferred way to run DNSSEC in a zone, but sometimes a more “hands-on” approach to signing and key maintenance is needed. For this reason, we cover manual signing techniques in Manual Signing.
| .    | Found 3 DNSKEY records for .  
|      | DS=20326/SHA-256 verifies DNSKEY=20326/SEP  
|      | Found 1 RRSIGs over DNSKEY RRset  
|      | RRSIG=20326 and DNSKEY=20326/SEP verifies the DNSKEY RRset  |
| org  | Found 2 DS records for org in the . zone  
|      | DS=9795/SHA-1 has algorithm RSASHA1-NSEC3-SHA1  
|      | DS=9795/SHA-256 has algorithm RSASHA1-NSEC3-SHA1  
|      | Found 1 RRSIGs over DS RRset  
|      | RRSIG=48903 and DNSKEY=48903 verifies the DS RRset  
|      | Found 4 DNSKEY records for org  
|      | DS=9795/SHA-1 verifies DNSKEY=9795/SEP  
|      | Found 3 RRSIGs over DNSKEY RRset  
|      | RRSIG=9795 and DNSKEY=9795/SEP verifies the DNSKEY RRset  |
| isc.org | Found 1 DS records for isc.org in the org zone  
|        | DS=7250/SHA-256 has algorithm ECDSAP256SHA256  
|        | Found 1 RRSIGs over DS RRset  
|        | RRSIG=37022 and DNSKEY=37022 verifies the DS RRset  
|        | Found 2 DNSKEY records for isc.org  
|        | DS=7250/SHA-256 verifies DNSKEY=7250/SEP  
|        | Found 2 RRSIGs over DNSKEY RRset  
|        | RRSIG=7250 and DNSKEY=7250/SEP verifies the DNSKEY RRset  
|        | isc.org A RR has value 149.20.1.66  
|        | Found 1 RRSIGs over A RRset  
|        | RRSIG=27566 and DNSKEY=27566 verifies the A RRset  |

Fig. 3: Verisign DNSSEC Debugger
Fig. 4: DNSViz
12.5.5 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.

**DS Record Format**

Below is an example of a DS record format generated from the KSK we created earlier (Kexample.com.+013+10376.key):

```bash
# cd /etc/bind
dnssec-dsfromkey Kexample.com.+013+10376.key example.com. IN DS 10376 13 2
   =B92E22CAE0B41430EC38D3F7EDF11B3C3A9F4D474856925C15EE33B8312EF0
```

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):

```bash
example.com. 3600 IN DNSKEY 257 3 13 (6saiq99qDB...dqp+0dw==) ; key id = 10376
```

The key itself is easy to find (it’s difficult to miss that long base64 string) in the file.

```bash
# 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)
exmaple.com. 3600 IN DNSKEY 257 3 13 6saiq99qDB...dqp+0dw==
```
12.5.6 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:

```plaintext
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;
  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:

```plaintext
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.

### 12.5.7 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\(^6\).

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"; };
  checkds explicit;
  ...
};
```

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 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.

The example also has `checkds` set to `explicit`. This means that only the addresses defined in `parental-agents` are being queried. If set to `yes`, the parental agents are being looked up by querying for the parent NS records.

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
```

This command should also be used when `checkds` is set to `no`.

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.

### 12.5.8 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?

Although use of the automatic `dnssec-policy` is the preferred way to sign zones in BIND, there are occasions where a manual approach may be needed. `dnssec-policy` does not currently support the use of external hardware, so if your security policy requires it, you need to use manual signing.

\(^6\) The dates can also be modified using an editor, but that is likely to be more error-prone than using `dnssec-settime`.

---

12.5. Signing 383
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.

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.

The first step is to create the keys as described in Generate Keys, then using the BIND-provided tools \texttt{dnssec-keygen} to create the keys and \texttt{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 \texttt{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 \texttt{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.

\textbf{Note:} Again, we assume all configuration files, key files, and zone files are stored in \texttt{/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 \texttt{example.com} that we are converting to a secure version. The secure version uses both a KSK and a ZSK.

\section*{Generate Keys}

Everything in DNSSEC centers around keys, so we begin by generating our own keys.

\begin{verbatim}
# cd /etc/bind/keys
# dnssec-keygen -a ECDSAP256SHA256 example.com
Generating key pair........................++ ..................................++
Kexample.com.+013+34371
# dnssec-keygen -a ECDSAP256SHA256 -f KSK example.com
Generating key pair........................+++ ..................................+++.
Kexample.com.+013+00472
\end{verbatim}

This command generates four key files in \texttt{/etc/bind/keys}:

\begin{itemize}
\item Kexample.com.+013+34371.key
\item Kexample.com.+013+34371.private
\item Kexample.com.+013+00472.key
\item Kexample.com.+013+00472.private
\end{itemize}

The two files ending in \texttt{.key} are the public keys. These contain the DNSKEY resource records that appear in the zone. The two files ending in \texttt{.private} are the private keys, and contain the information that \texttt{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):
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. See chapter Zone keys for more details.

Make sure that these files are readable by named and that the .private files are not readable by anyone else.

Alternatively, 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.

### Setting Key Timing Information

Key files contain time information related to rolling keys. 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.

Note that dnssec-policy does set key timing information, but it uses its own state machine to determine what actions to perform.

But when performing manual signing the key parameters and the timing information in the key files, you can implement any DNSSEC policy you want for your zones.

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. 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.
```

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: 20200616104249 (Tue Jun 16 11:42:49 2020)
; Activate: 20200616104249 (Tue Jun 16 11:42:49 2020)
example.com. IN DNSKEY 256 3 13 AwEAa6fel66...LqkA7cvn8=
```

(The actual key is truncated here to improve readability.)

7 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.
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:

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Included in Zone File?</th>
<th>Used to Sign Data?</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>No</td>
<td>No</td>
<td>Recording of key creation</td>
</tr>
<tr>
<td>Publish</td>
<td>Yes</td>
<td>No</td>
<td>Introduction of a key soon to be active</td>
</tr>
<tr>
<td>Activate</td>
<td>Yes</td>
<td>Yes</td>
<td>Activation date for new key</td>
</tr>
<tr>
<td>Revoke</td>
<td>Yes</td>
<td>Yes</td>
<td>Notification of a key soon to be retired</td>
</tr>
<tr>
<td>Inactive</td>
<td>Yes</td>
<td>No</td>
<td>Inactivation or retirement of a key</td>
</tr>
<tr>
<td>Delete</td>
<td>No</td>
<td>No</td>
<td>Deletion or removal of a key from a zone</td>
</tr>
</tbody>
</table>

The publication date is the date the key should be introduced into the zone. The activation date can be used to determine when to sign resource records. With “Inactive” you signal when the signer should stop generating new signatures with the given key, and the “Delete” metadata specifies when the key should be 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.

### Signing the Zone

Now, edit the zone file to make sure the proper DNSKEY entries are included. The public keys should be inserted into the zone file by including the `.key` files using `$INCLUDE` statements.

Use the command `dnssec-signzone`. 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.

The following command signs the zone, assuming it is in a file called `zone.child.example`, using manually specified keys:
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 can be manually verified for correctness.

dnssec-signzone also produces keys set and dsset files. These are used to provide the parent zone administrators with the DNSKEY records (or their corresponding DS records) that are the secure entry point to the zone.

By default, all zone keys which have an available private key are used to generate signatures. You can use the `-S` to only include keys that have the “Activate” timing metadata in the past and the “Inactive” timing metadata in the future (or not present).

### Reloading the Zone

Now it is time to inform BIND that a new signed zone file is available. We can do this with the `rndc reload example.com` command.

### 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.

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.

If not yet done so, you will need to pass that to the administrator of the parent zone, to be placed in the zone. 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.

Re-signing 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.

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.

12.6 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.

12.6.1 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 nameserver, 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)
;; 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:
```
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.

### 12.6.2 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 `dig +cd` flag 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 @10.53.0.2 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 aa rd; 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
```
With `delv`, a “resolution failed” message is output instead:

```bash
$ 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.

### 12.6.3 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.

### 12.6.4 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 ./DNSKEY: 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/DNSKEY: marking as secure (DS)
validating isc.org/DNSKEY: in validator_callback_dnskey
validating isc.org/DNSKEY: dsset with trust secure
validating isc.org/DNSKEY: verify rdataset (keyid=33853): success
validating isc.org/DNSKEY: marking as secure, noqname proof not needed
validating org/DNSKEY: starting
validating org/DNSKEY: attempting positive response validation
   validating org/DNSKEY: starting
   validating org/DNSKEY: attempting positive response validation
   validating org/DNSKEY: keyset with trust secure
   validating org/DNSKEY: verify rdataset (keyid=33853): success
   validating org/DNSKEY: 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/DNSKEY: in fetch_callback_dnskey
   validating isc.org/DNSKEY: keyset with trust secure
   validating isc.org/DNSKEY: resuming validate
   validating isc.org/DNSKEY: verify rdataset (keyid=33209): success
   validating isc.org/DNSKEY: 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.
12.6.5 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 9F3CADC...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 `dig +cd` option to temporarily disable checking and return results, even though they do not pass the validation tests. The `dig +multiline` option causes `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
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
```

(continues on next page)
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.

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.
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]: 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:
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:

```bash
$ dig @10.53.0.3 www.example.com.
;; <<>> DiG 9.16.0 <<>> @10.53.0.3 www.example.org A +dnssec
;; 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
;
;; QUESTION SECTION:
;;www.example.org.  IN  A
```

delv shows a similar result:

```bash
$ 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:

```bash
managed-keys-zone: D NFSKEY 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.

### 12.6.6 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.:

```bash
$ 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.:

```bash
$ 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.

**12.6.7 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 1234567890ABCDEF 1 10 www.example.com
RN7I9ME6E1I6BDKIP91B9TCE4FHJ7LKF (salt=1234567890ABCDEF, hash=1, iterations=10)
```

Zero-length salt can be specified as `-`.

While it is unlikely you would construct a rainbow table of your own zone data, this tool may be useful when troubleshooting NSEC3 problems.

**12.7 Advanced Discussions**

**12.7.1 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.
12.7.2 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:

<table>
<thead>
<tr>
<th>example.com.</th>
<th>300 IN NSEC alice.example.com. A RRSIG NSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice.example.com.</td>
<td>300 IN NSEC edward.example.com. A RRSIG NSEC</td>
</tr>
<tr>
<td>edward.example.com.</td>
<td>300 IN NSEC susan.example.com. A RRSIG NSEC</td>
</tr>
<tr>
<td>susan.example.com.</td>
<td>300 IN NSEC example.com. A RRSIG NSEC</td>
</tr>
</tbody>
</table>

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 potential drawback of NSEC is a bigger zone file and memory consumption; there is no opt-out mechanism for insecure child zones, so each name in the zone will get an additional NSEC record and a RRSIG record to go with it. In practice this is a problem only for parent-zone operators dealing with mostly 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 other names in the zone.

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 0 = JKMA... A RRSIG
JKMA....example.com. 300 IN NSEC3 1 0 0 = NTQ0... A RRSIG
NTQ0....example.com. 300 IN NSEC3 1 0 0 = 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.

Most names published in the DNS are rarely secret or unpredictable. They are published to be memorable, used and consumed by humans. They are often recorded in many other network logs such as email logs, certificate transparency logs, web page links, intrusion detection systems, malware scanners, email archives, etc. Many times a simple dictionary of commonly used domain-name prefixes (www, mail, imap, login, database, etc.) can be used to quickly reveal a large number of labels within a zone. Additionally, if an adversary really wants to expend significant CPU resources to mount an offline dictionary attack on a zone’s NSEC3 chain, they will likely be able to find most of the “guessable” names despite any level of hashing.

Also, it is still possible to 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, use some of the special techniques described in RFC 4470.

NSEC3PARAM

Warning: Before we dive into the details of NSEC3 parametrization, please note: the defaults should not be changed without a strong justification and a full understanding of the potential impact. See RFC 9276.

The above NSEC3 examples used four parameters: 1, 0, 0, and zero-length salt. 1 represents the algorithm, 0 represents the opt-out flag, 0 represents the number of additional iterations, and - is the salt. Let’s look at how each one can be configured:

Algorithm

NSEC3 Hashing Algorithm

The only currently defined value is 1 for SHA-1, so there is no configuration field for it.

Opt-out

Setting this bit to 1 enables NSEC3 opt-out, which is discussed in NSEC3 Opt-Out.

Iterations

Iterations defines the number of _additional_ times to apply the algorithm when generating an NSEC3 hash. More iterations 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.

Warning: Do not use values higher than zero. A value of zero provides one round of SHA-1 hashing and protects from non-determined attackers.

A greater number of additional iterations causes interoperability problems and opens servers to CPU-exhausting DoS attacks, while providing only doubtful security benefits.

Salt

A salt value, which can be combined with an FQDN to influence the resulting hash. Salt is discussed in more detail in NSEC3 Salt.
NSEC3 Opt-Out

First things first: For most DNS administrators who do not manage a huge number of insecure delegations, the NSEC3 opt-out feature is not relevant. See RFC 9276.

Opt-out allows for blocks of unsigned delegations to be covered by a single NSEC3 record. In other words, use of the opt-out allows large registries to only sign as many NSEC3 records as there are signed DS or other RRsets in the zone; with opt-out, unsigned delegations do not require additional NSEC3 records. This sacrifices the tamper-resistance proof of non-existence offered by NSEC3 in order to reduce memory and CPU overheads, and decreases the effectiveness of the cache (RFC 8198).

Why would that ever be desirable? If a significant number of delegations are not yet securely delegated, meaning they lack DS records and are still insecure or unsigned, generating DNSSEC records for all their NS records might consume lots of memory and is not strictly required by the child zones.

This resource-saving typically makes a difference only for huge zones like com. Imagine that you are the operator of busy top-level domains such as com, with millions of insecure delegated domain names. As of mid-2022, around 3% of all com zones are signed. Basically, without opt-out, with 1,000,000 delegations, only 30,000 of which are secure, you still have to generate NSEC RRsets for the other 970,000 delegations; with NSEC3 opt-out, you will have saved yourself 970,000 sets of records.

In contrast, for a small zone the difference is operationally negligible and the drawbacks outweigh the benefits.

If NSEC3 opt-out is truly essential for a zone, the following configuration can be added to `dnssec-policy`; for example, to create an NSEC3 chain using the SHA-1 hash algorithm, with the opt-out flag, no additional iterations, and no extra salt, use:

```
dnssec-policy "nsec3" {
    ...  
    nsec3param iterations 0 optout yes salt-length 0;
};
```

To learn more about how to configure NSEC3 opt-out, please see NSEC3 Opt-Out.

NSEC3 Salt

```
Warning: Counter to popular belief, adding salt provides little value. Each DNS zone is always uniquely salted using the zone name. Operators should use a zero-length salt value.
```

The properties of this extra salt are complicated and beyond scope of this document. For detailed description why the salt in the context of DNSSEC provides little value please see RFC 9276.

NSEC or NSEC3?

So which is better: NSEC or NSEC3? There is no single right answer here that fits everyone; it comes down to a given network’s needs or requirements.

In most cases, NSEC is a good choice for zone administrators. It relieves the authoritative servers and resolver of the additional cryptographic operations that NSEC3 requires, and NSEC is comparatively easier to troubleshoot than NSEC3.

NSEC3 comes with many drawbacks and should be implemented only if zone enumeration prevention is really needed, or when opt-out provides a significant reduction in memory and CPU overheads (in other words, with a huge zone with mostly insecure delegations).
12.7.3 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
;; 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: 5258d7ed09db0d76010000005ealcc8c672d8db27a464e37 (good)
;; QUESTION SECTION:
;example.com. IN DNSKEY

;; ANSWER SECTION:
example.com. 60 IN DNSKEY 256 3 13 (tAeXLtIQ3aVDqqS/1UVrtn9AE6/nzfoAu4tT1V4y4dY12CK
plNcUJxME12//pnGXY/+HqDU7Gr5HkJY8V0W3r5fzlw==
) ; ZSK; alg = ECDSAP256SHA256 ; key id = 63722

example.com. 60 IN DNSKEY 257 3 13 (cxkNesgubBPSXra5ug2p8rWy63B8jTnS4n0lYsS9eW
Vh1yQ0mdgevKUhfg3SE1wbLChJc2FAbvS21qK03Nw==
) ; 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
;; global options: +cmd
```
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 RRs 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.

12.7.4 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 dnssec-policy, 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, 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.
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 these steps 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.

12.7.5 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).

12.7.6 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, 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.

Based on APNIC statistics at https://stats.labs.apnic.net/dnssec/XA
12.8 Recipes

This chapter provides step-by-step “recipes” for some common DNSSEC configurations.

12.8.1 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.

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:

```plaintext
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:

```plaintext
zone "example.com" IN {
  type secondary;
  file "db/example.com.db";
};
```

(continues on next page)
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:

```plaintext
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:

```plaintext
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!
};
```

### 12.8.2 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 RRsset 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.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...1M3NRn/G/R)
; KSK; alg = RSASHA256; key id = 6817
example.com.  600 IN DNSKEY 256 3 8 (AwEAAAbi6Vo...qBW5+iAqNz)
; ZSK; alg = RSASHA256; key id = 51623
example.com.  600 IN DNSKEY 256 3 8 (AwEAAcjGaU...Drzuu551f5)
```

(continues on next page)
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. ( 
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 ( 
20210101000000 20201201230000 17694 example.com. 
2bQ29h9aL4vQ1QdOdPB+7LBeM8qF7s5y/njvjU...MkPS/0k= )

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 dnssec-policy, 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:
exmple.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)
)
exmple.com. 600 IN RRSIG SOA 8 2 600 ( 
20210131000000 20201231230000 51623 example.com. 
J4RMnpJP0mMidsElYBuugjP0RlqX0NqVf0/2AT6yAAvx9X
z3RlLcukhKcyCSL29Z2+zZ2y4u2LyQGrNiondaKdQCOr7
uTqH5WCpoqa1OCbjq7U7c7v1AM2709RD11nzPNpVQ7xPs

(continues on next page)
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:

```bash
$ 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+23550.private
# dnssec-keygen -S Kexample.com.+007+23550
Generating key pair..........................................................++
Kexample.com.+007+23550
# dnssec-dsfromkey -a SHA-1 Kexample.com.+007+23550.key
example.com. IN DS 23550 7 1 54FCF030A1C79C0088FDEC1BD1C37DA2E70DFB
```

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+24828`).

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:
You can check the publish date of the key by looking at the key file:

```bash
# 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:

```bash
$ dig @192.168.1.13 example.com. DNSKEY +dnssec +multiline
...;
;; ANSWER SECTION:
example.com. 300 IN DNSKEY 256 3 7 (
    AwEAAAdYqAc...TiS1rma6Ef ) ; 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. 
    4yi1PVJOrK...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:

1. After logging in, click the green “Launch” button next to the domain name you want to manage.
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.”

---

9 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. 7: Upload DS Record Step #1

Fig. 8: Upload DS Record Step #2
Manage DNSSEC DS Records

Choose how to set up your DS records.

<table>
<thead>
<tr>
<th>Key tag</th>
<th>Algorithm</th>
<th>Digest Type</th>
<th>Digest</th>
<th>Max Sig Life</th>
<th>Flags</th>
<th>Protocol</th>
<th>Key Data Alg</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>24828</td>
<td>7</td>
<td>1</td>
<td>DIA3D8BD1D55DA...</td>
<td>161440D</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Add DS Record

Fig. 9: Upload DS Record Step #3

Create DS Record

Key tag: * 23550
Algorithm: * 7
Digest type: * 1
Digest: * 54FCF030AA1C79C0088FDEC1BD1C37DA2E70DFB
Max sig life: Flags: Protocol: Key data alg: Public key:

Fig. 10: Upload DS Record Step #4
12.8. Recipes

Fig. 11: Upload DS Record Step #5

Fig. 12: Upload DS Record Step #6
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:
extample.com. 21552 IN DS 24828 7 1 D4A33E8DD550A9567B4C4971A34AD6C4B80A6AD3
texample.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.

### 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:
extample.com. 300 IN DNSKEY 256 3 7 (AwEAAAdYqAc...TiSlrma6Ef ) ; ZSK; alg = NSEC3RSASHA1; key id = 29747
texample.com. 300 IN DNSKEY 257 3 7 (AwEAAeTJ+w...O+Zy9j0m63 ) ; KSK; alg = NSEC3RSASHA1; key id = 24828
texample.com. 300 IN DNSKEY 257 3 7 (AwEAAac1BQN...Wdc0qoH21H ) ; KSK; alg = NSEC3RSASHA1; key id = 23550
texample.com. 300 IN RRSIG DNSKEY 7 2 300 (20210201074900 20210101064900 23550 mydnssecgood.org.
S6zTbBTfvU...Ib5eXktbE= )
texample.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 (AwEAAAdYqAc...TlSlrma6Ef ) ; ZSK; alg = NSEC3RSASHA1; key id = 29747
example.com. 300 IN DNSKEY 257 3 7 (AwEAAac1BQN...Wdc0qoH2lH ) ; 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. Ouelp1lpY9...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.

12.8.3 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 0 additional iterations, no opt-out, and a zero-length salt:
Fig. 13: Remove DS Record Step #1

Fig. 14: Remove DS Record Step #2
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,0,-)
```

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
...
5A03TL362CS8VSIH69CVA4MJIKRFQH3.example.com. 300 IN NSEC3 1 0 0 - {
   TQ9QBEAG6CROHEOC8KIH1A2C06IVQ5ER
   NS SOA RRSIG DNSKEY NSEC3PARAM }
...
```

Our example used four parameters: 1, 0, 0, and -, in order. 1 represents the algorithm, 0 represents the opt-out flag, 0 represents the number of additional iterations, and - denotes no salt is used. 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,0,-)
```

You can also query for a name that you know does not exist, and you should no longer see any traces of NSEC3 records.
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.

**Warning:** NSEC3 Opt-Out feature brings benefit only to extremely large zones with lots of insecure delegations. Its use is counterproductive in all other cases as it decreases tamper-resistance of the zone and also decreases efficiency of resolver cache (see RFC 8198).

In other words, don’t enable Opt-Out unless you are serving an equivalent of com. zone.

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: IFA1I3IE7EKCTPHM6R58URO3Q846I52M.example.com
- bbb.example.com: ROJUF3VJSJO6LQ2LC1DNSJ5GBAUJPVHE.example.com
- ccc.example.com: 0VPUT696LUVDPDS5NIHSHBH9KLV20V5K.example.com
- ddd.example.com: UHPBD5U4HRGB84MLC2NQOVEFNAKJU0CA.example.com

```bash
$ dig @192.168.1.13 reisiergiuhiwhiouwe.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
...`
We can enable NSEC3 opt-out with the following configuration, changing the `optout` configuration value from `no` to `yes`:

```plaintext
dnssec-policy "standard" {
    nsec3param iterations 0 optout yes salt-length 0;
};
```

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.

```plaintext
• bbb.example.com: ROJUF3VJSIO6LQ2LC1DNSJ5GBAUPVHE.example.com
• ftp.example.com: 8P15KCUT1RHCSDN46HBQVPI5T532IN1.example.com
• ns1.example.com: GUFVRA2SFI08RSFP7UO41E8AD1KR41FH.example.com
• web.example.com: CVQ4LA4ALPQIAO2H3N2RB61R8UHM91E7.example.com
• www.example.com: MIFDNDT3NFF3OD53O7TLA1HRFF95JKUK.example.com
• example.com: ONIB9MGUB9H0RML3CDF5BGRJ59DKJHVK.example.com
```

To undo NSEC3 opt-out, change the configuration again:

```plaintext
dnssec-policy "standard" {
    nsec3param iterations 0 optout no salt-length 0;
};
```

**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:

```plaintext
# nsec3hash -i 0 www.example.com.
MIFDNDT3NFF3OD53O7TLA1HRFF95JKUK (salt=-, hash=1, iterations=0)
```

### 12.8.4 Reverting to Unsigned

This recipe describes how to revert from a signed zone (DNSSEC) back to an unsigned (DNS) zone.

Here is what `named.conf` looks like when it is signed:

```plaintext
zone "example.com" IN {
    type primary;
    file "db/example.com.db";
    dnssec-policy "default";
};
```
To indicate the reversion to unsigned, change the `dnssec-policy` line:

```
zone "example.com" IN {
    type primary;
    file "db/example.com.db";
    dnssec-policy "insecure";
};
```

Then use `rndc reload` to reload the zone.

The “insecure” policy is a built-in policy (like “default”). It makes sure the zone is still DNSSEC-maintained, to allow for a graceful transition to unsigned. It also publishes the CDS and CDNSKEY DELETE records automatically at the appropriate time.

If the parent zone allows management of DS records via CDS/CDNSKEY, as described in RFC 8078, the DS record should be removed from the parent automatically.

Otherwise, DS records can be removed via the registrar. 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](image)

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.

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 parental agents are configured, the DNSSEC records will be automatically removed after BIND has seen that the parental agents no longer serve the DS RRset for this zone.

After a while, the zone is reverted back to the traditional, insecure DNS format. This can be verified by checking that all DNSKEY and RRSIG records have been removed from the zone.

The `dnssec-policy` line can then be removed from `named.conf` and the zone reloaded. The zone will no longer be subject to any DNSSEC maintenance.
**12.8. Recipes**

Fig. 17: Revert to Unsigned Step #2

Fig. 18: Revert to Unsigned Step #3
12.9 Commonly Asked Questions

Below are some common questions 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. For protection against eavesdropping, the preferred protocol is DNS-over-TLS. DNS-over-HTTPS can also do the job, but it is more complex.

**If I deploy DNS-over-TLS/HTTPS, can I skip deploying DNSSEC?**

No. DNS-over-encrypted-transport stops eavesdroppers on a network, but it does not protect against cache poisoning and answer manipulation in other parts of the DNS resolution chain. In other words, these technologies offer protection only for records when they are in transit between two machines; any compromised server can still redirect traffic elsewhere (or simply eavesdrop). However, DNSSEC provides integrity and authenticity for DNS records, even when these records are stored in caches and on disks.

**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.

**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?**

DNSSEC makes it much more difficult for attackers to spoof DNS responses or perform cache poisoning. It cannot 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.

In other words, DNSSEC is designed to provide confidence that when a DNS response is received 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.

**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. 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 Secure and Insecure; only Bogus and Indeterminate result in a SERVFAIL. As of mid-2022, roughly one-third of users worldwide are using 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.
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.
14.1 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.

14.1.1 Protocol Specifications


---

\(^1\) 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 2931 - D. Eastlake, 3rd. DNS Request and Transaction Signatures (SIG(0)s). September 2000.


---

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.
14.1. Requests for Comment (RFCs)
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.

RFC 7050 is updated by RFC 8880.

Updating of parent zones is not yet implemented.

named does not currently encrypt DNS requests, so the PAD option is accepted but not returned in responses.

Forwarding DNS queries over encrypted transports is not supported yet.

Updating of parent zones is not yet implemented.

Strict TLS and Mutual TLS authentication mechanisms are not supported yet.


14.1.2 Best Current Practice RFCs


RFC 7793 - M. Andrews. Adding 100.64.0.0/10 Prefixes to the IPv4 Locally-Served DNS Zones Registry. May 2016.


14.1.3 For Your Information


---

16 This does not apply to DNS server implementations.

5 Compliance is with loading and serving of A6 records only. A6 records were moved to the experimental category by RFC 3363.

4 Compliance is with loading and serving of DLV records only. DLV records were moved to the historic category by RFC 8749.
14.2 Notes

14.3 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.
15.1 arpaname - translate IP addresses to the corresponding ARPA names

15.1.1 Synopsis

arpaname {ipaddress …}

15.1.2 Description

arpaname translates IP addresses (IPv4 and IPv6) to the corresponding IN-ADDR.ARPA or IP6.ARPA names.

15.1.3 See Also


15.2 ddns-confgen - TSIG key generation tool

15.2.1 Synopsis

ddns-confgen [-a algorithm] [-h] [-k keyname] [-q] [-s name] [-z zone]

15.2.2 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.
15.2.3 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.

15.2.4 See Also


15.3 delv - DNS lookup and validation utility

15.3.1 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…]`
15.3.2 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).

15.3.3 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.

15.3.4 Options

**-a anchor-file**

This option specifies a file from which to read an alternate DNSSEC root zone trust anchor.

By default, keys that do not match the root zone name (.) are ignored. If an alternate key name is desired, it can be specified using the `+root` option.

Note: When reading trust anchors, 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.

15.3. delv - DNS lookup and validation utility
**delv** does not consult the managed-keys database maintained by **named**. This means that if the default key built in to **delv** is revoked, **delv** must be updated to a newer version in order to continue validating.

**-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 IAXFR. 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.
15.3.5 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:

`+cdflag`, `+nocdflag`

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.

`+class`, `+noclass`

This option controls whether to display the CLASS when printing a record. The default is to display the CLASS.

`+hint=file`, `+nohint`

This option specifies a filename from which to load root hints; this will be used to find the root name servers when name server mode (`delv +ns`) is in use. If the option is not specified, built-in root hints will be used.

`+ns`, `+nons`

This option toggles name server mode. When this option is in use, the `delv` process instantiates a full recursive resolver, and uses that to look up the requested query name and type. Turning on this option also activates `+mtrace`, `+strace` and `+rtrace`, so that every iterative query will be logged, including the full response messages from each authoritative server. These logged messages will be written to `stdout` rather than `stderr` as usual, so that the full trace can be captured more easily.

This is intended to be similar to the behavior of `dig +trace`, but because it uses the same code as `named`, it much more accurately replicates the behavior of a recursive name server with a cold cache that is processing a recursive query.

`+qmin[=MODE]`, `+noqmin`

When used with `+ns`, this option enables QNAME minimization mode. Valid options of `MODE` are `relaxed` and `strict`. By default, QNAME minimization is disabled. If `+qmin` is specified but `MODE` is omitted, then `relaxed` mode will be used.

`+ttl`, `+nottl`

This option controls whether to display the TTL when printing a record. The default is to display the TTL.

`+rtrace`, `+nortrace`

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.

`+mtrace`, `+nomtrace`

This option toggles logging of messages received. 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.
+strace, +nostrace
   This option toggles logging of messages sent. This produces a detailed dump of the queries sent by delv in the process of carrying out the resolution and validation process. Turning on this option also activates +mtrace.
   This is equivalent to setting the debug level to 11 for the “packets” module of the “resolver” logging category. Setting the systemwide debug level to 11 using the -d option produces the same output, but affects other logging categories as well.

+vttrace, +novttrace
   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.

+short, +noshort
   This option toggles between verbose and terse answers. The default is to print the answer in a verbose form.

+comments, +nocomments
   This option toggles the display of comment lines in the output. The default is to print comments.

+rrcomments, +norrrcomments
   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.

+crypto, +nocrypto
   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 ].

+trust, +notrust
   This option controls whether to display the trust level when printing a record. The default is to display the trust level.

+split[=W], +nosplit
   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.

+all, +noall
   This option sets or clears the display options +comments, +rrcomments, and +trust as a group.

+multiline, +nomultiline
   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.

+dnssec, +nodnssec
   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.

+root[=ROOT], +noroot
   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.
This option controls whether to use TCP when sending queries. The default is to use UDP unless a truncated response has been received.

+unknownformat, +nounknownformat

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.

+yaml, +noyaml

This option prints response data in YAML format.

15.3.6 Files

/etc/resolv.conf

15.3.7 See Also

dig(1), named(8), RFC 4034, RFC 4035, RFC 4431, RFC 5074, RFC 5155.

15.4 dig - DNS lookup utility

15.4.1 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…]

15.4.2 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.
15.4.3 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.

15.4.4 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.

- `-h`
  Print a usage summary.

- `-k keyfile`
  This option tells `dig` to sign queries using TSIG or SIG(0) 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` for TSIG and by looking up the KEY record in zone data for SIG(0).
This option enables memory usage debugging.

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.

This option specifies the domain name to query. This is useful to distinguish the name from other arguments.

This option indicates that options from ${HOME}/.digrc should not be read. This is useful for scripts that need predictable behavior.

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.

This option indicates that print query times should be provided in microseconds instead of milliseconds.

This option prints the version number and exits.

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.

This option signs queries using TSIG with the given 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; 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.
15.4.5 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 +cddfag. The query options are:

**+aaflag, +noaaflag**

This option is a synonym for +aaonly, +noaalone.

**+aaonly, +noaaonly**

This option sets the aa flag in the query.

**+additional, +nocaalld**

This option displays [or does not display] the additional section of a reply. The default is to display it.

**+adflag, +noadflag**

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.

**+all, +noall**

This option sets or clears all display flags.

**+answer, +noanswer**

This option displays [or does not display] the answer section of a reply. The default is to display it.

**+authority, +noauthority**

This option displays [or does not display] the authority section of a reply. The default is to display it.

**+badcookie, +nobadcookie**

This option retries the lookup with a new server cookie if a BADCOOKIE response is received.

**+besteffort, +nobesteffort**

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 restores the default buffer size.

**+cd, +cddfag, +nccdflags**

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.

**+class, +noclass**

This option displays [or does not display] the CLASS when printing the record.

**+cmd, +nocmd**

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.
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 +cmd, +question, +stats, and +rrcomments.

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.

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 ].

This option, which is deprecated, is treated as a synonym for +search, +nosearch.

Lookup IPV4ONLY.ARPA AAAA and print any DNS64 prefixes found.

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.

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.

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.

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.

This option enables/disables EDNS version negotiation. By default, EDNS version negotiation is enabled.

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.

This option sends an EDNS Expire option.

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.
+fuzztime [=value], +nofuzztime
This option allows the signing time to be specified when generating signed messages. If a value is specified it is the seconds since 00:00:00 January 1, 1970 UTC ignoring leap seconds. If no value is specified 1646972129 (Fri 11 Mar 2022 04:15:29 UTC) is used. The default is +nofuzztime and the current time is used.

+header-only, +noheader-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.

+https [=value], +nohttps
This option indicates whether to use DNS over HTTPS (DoH) when querying name servers. When this option is in use, the port number defaults to 443. The HTTP POST request mode is used when sending the query.

If value is specified, it will be used as the HTTP endpoint in the query URI; the default is /dns-query. So, for example, dig @example.com +https will use the URI https://example.com/dns-query.

+https-get [=value], +nohttps-get
Similar to +https, except that the HTTP GET request mode is used when sending the query.

+https-post [=value], +nohttps-post
Same as +https.

+http-plain [=value], +nohttp-plain
Similar to +https, except that HTTP queries will be sent over a non-encrypted channel. When this option is in use, the port number defaults to 80 and the HTTP request mode is POST.

+http-plain-get [=value], +nohttp-plain-get
Similar to +http-plain, except that the HTTP request mode is GET.

+http-plain-post [=value], +nohttp-plain-post
Same as +http-plain.

+identify, +noidentify
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.

+idn, +noidn
Enable or disable IDN processing. By default IDN is enabled for input query names, and for display when the output is a terminal.

You can also turn off dig's IDN processing by setting the IDN_DISABLE environment variable.

+ignore, +noignore
This option ignores [or does not ignore] truncation in UDP responses instead of retrying with TCP. By default, TCP retries are performed.

+keepalive, +nokeepalive
This option sends [or does not send] an EDNS Keepalive option.

+keepopen, +nokeepopen
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.

+multiline, +nomultiline
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.

+nsid, +nonsid

When enabled, this option includes an EDNS name server ID request when sending a query.

+nssearch, +nonssearch

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.

+onesoa, +noonesoa

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.

+opcode=value, +noopcode

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.

+proxy[=src_addr[#src_port]-dst_addr[#dst_port]], +noproxy

When this option is set, dig adds PROXYv2 headers to the queries. When source and destination addresses are specified, the headers contain them and use the PROXY command. It means for the remote peer that the queries were sent on behalf of another node and that the PROXYv2 header reflects the original connection endpoints. The default source port is 0 and destination port is 53.

For encrypted DNS transports, to prevent accidental information leakage, encryption is applied to the PROXYv2 headers: the headers are sent right after the handshake process has been completed.

For plain DNS transports, no encryption is applied to the PROXYv2 headers.

If the addresses are omitted, PROXYv2 headers, that use the LOCAL command set, are added instead. For the remote peer, that means that the queries were sent on purpose without being relayed, so the real connection endpoint addresses must be used.

+proxy-plain[=src_addr[#src_port]-dst_addr[#dst_port]], +noproxy-plain

The same as +[no]proxy, but instructs dig to send PROXYv2 headers ahead of any encryption, before any handshake messages are sent. That makes dig behave exactly how it is described in the PROXY protocol specification, but not all software expects such behaviour.

Please consult the software documentation to find out if you need this option. (for example, dnsdist expects encrypted PROXYv2 headers sent over TLS when encryption is used, while HAProxy and many other software packages expect plain ones).

For plain DNS transports the option is effectively an alias for the +[no]proxy described above.

+qid=value

This option specifies the query ID to use when sending queries.
+qr, +noqr
This option toggles the display of the query message as it is sent. By default, the query is not printed.

+question, +noquestion
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.

+raflag, +noraflag
This option sets [or does not set] the RA (Recursion Available) bit in the query. The default is +noraflag. This bit is ignored by the server for QUERY.

+rdflag, +nordflag
This option is a synonym for +recurse, +norecurse.

+recurse, +norecurse
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.

+rrcomments, +norrcomments
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.

+search, +nosearch
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.

+short, +noshort
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.

+showbadcookie, +noshowbadcookie
This option toggles whether to show the message containing the BADCOOKIE rcode before retrying the request or not. The default is to not show the messages.

+showsearch, +noshowsearch
This option performs [or does not perform] a search showing intermediate results.

+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.

+stats, +nostats
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.

+subnet=addr[/prefix-length], +nosubnet
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.

**+tcflag, +notcflag**

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.

**+tcp, +notcp**

This option indicates whether to 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. To prevent retry over TCP when TC=1 is returned from a UDP query, use +ignore.

**+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.

**+tls, +notls**

This option indicates whether to use DNS over TLS (DoT) when querying name servers. When this option is in use, the port number defaults to 853.

**+tls-ca [=file-name], +notls-ca**

This option enables remote server TLS certificate validation for DNS transports, relying on TLS. Certificate authorities certificates are loaded from the specified PEM file (*file-name*). If the file is not specified, the default certificates from the global certificates store are used.

**+tls-certfile= file-name, +tls-keyfile= file-name, +notls-certfile, +notls-keyfile**

These options set the state of certificate-based client authentication for DNS transports, relying on TLS. Both certificate chain file and private key file are expected to be in PEM format. Both options must be specified at the same time.

**+tls-hostname= hostname, +notls-hostname**

This option makes *dig* use the provided hostname during remote server TLS certificate verification. Otherwise, the DNS server name is used. This option has no effect if +tls-ca is not specified.

**+trace, +notrace**

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 set when +trace is set, to better emulate the default queries from a name server.

Note that the delv +ns option can also be used for tracing the resolution of a name from the root (see delv).

**+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.

**+ttlid, +notttlid**

This option displays [or does not display] the TTL when printing the record.

**+ttlunits, +notttlunits**

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 +tttlid.
**+unknownformat, +nounknownformat**

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.

**+vc, +novc**

This option uses [or does not use] TCP when querying name servers. This alternate syntax to +tcp is provided for backwards compatibility. The vc stands for “virtual circuit.”

**+yaml, +noyaml**

When enabled, this option prints the responses (and, if +qr is in use, also the outgoing queries) in a detailed YAML format.

**+zflag, +nozflag**

This option sets [or does not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

### 15.4.6 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 +cmd and +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.

### 15.4.7 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
15.4.8 Files

/etc/resolv.conf
$(HOME)/.digrc

15.4.9 See Also

delv(1), host(1), named(8), dnssec-keygen(8), RFC 1035.

15.4.10 Bugs

There are probably too many query options.

15.5 dnssec-cds - change DS records for a child zone based on CDS/CDNSKEY

15.5.1 Synopsis


15.5.2 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, option `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.

### 15.5.3 Options

**-a algorithm**

When converting CDS records to DS records, this option specifies the acceptable digest algorithms. This option can be repeated, so that multiple digest types are allowed. If none of the CDS records use an acceptable digest type, `dnssec-cds` will try to use CDNSKEY records instead; if there are no CDNSKEY records, it reports an error.

When converting CDNSKEY records to DS records, this option specifies the digest algorithm to use. It can be repeated, so that multiple DS records are created for each CDNSKEY 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 only.

**-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.

**-i extension**

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 YYYYMDDHHMMSS 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.
This option writes an \texttt{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 \texttt{dsset-} file, with the \texttt{-T} option, or using the \texttt{nsupdate ttl} command.

\textbf{-v}

This option prints version information.

\textbf{-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.

\textbf{domain}

This indicates the name of the delegation point/child zone apex.

\subsection*{15.5.4 Exit Status}

The \texttt{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.

\subsection*{15.5.5 Examples}

Before running \texttt{dnssec-signzone}, ensure that the delegations are up-to-date by running \texttt{dnssec-cds} on every \texttt{dsset-} file.

To fetch the child records required by \texttt{dnssec-cds}, invoke \texttt{dig} as in the script below. It is acceptable if the \texttt{dig} fails, since \texttt{dnssec-cds} performs all the necessary checking.

\begin{verbatim}
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
\end{verbatim}

When the parent zone is automatically signed by \texttt{named}, \texttt{dnssec-cds} can be used with \texttt{nsupdate} to maintain a delegation as follows. The \texttt{dsset-} file allows the script to avoid having to fetch and validate the parent DS records, and it maintains the replay attack protection time.

\begin{verbatim}
dig +dnssec +noall +answer $d DNSKEY $d CDNSKEY $d CDS |
dnssec-cds -u -i -f /dev/stdin -d $f $d |
nsupdate -l
\end{verbatim}
15.5.6 See Also


15.6 dnssec-dsfromkey - DNSSEC DS RR generation tool

15.6.1 Synopsis

dnssec-dsfromkey [-h | -V]

15.6.2 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`.

15.6.3 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.

-Ç
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.

15.6.4 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 3A1EADA7A74B8D0BA86726B0C227A85AB8BBBD2B2004F41A868A54F0C5EA0B941

15.6.5 Files

The keyfile can be designated by the key identification Knnnn.+aaa+iili or the full file name Knnnn.+aaa+iili.key, as generated by `dnssec-keygen`.

The keyset file name is built from the directory, the string keyset-, and the dnsname.
15.6.6 Caveat

A keyfile error may return “file not found,” even if the file exists.

15.6.7 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).

15.7 dnssec-importkey - import DNSKEY records from external systems so they can be managed

15.7.1 Synopsis


15.7.2 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.

15.7.3 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.

15.7.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS. (which is the format used inside key files), or 'Day Mon DD HH:MM:SS YYYY' (as printed by dnssec-settime -p), or UNIX epoch time (as printed by dnssec-settime -up), or the literal now.

The argument can be followed by + or - and an offset from the given time. The literal now can be omitted before an offset. The offset can be followed by one of the suffixes y, mo, w, d, h, or mi, so that it 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, never, or unset.

All these formats are case-insensitive.

-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.

   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.)

   sync date/offset
      This option sets the date on which the CDS and CDNSKEY records that match this key are to be deleted.

15.7.5 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.

15.7.6 See Also

dnssec-keygen(8), dnssec-signzone(8), BIND 9 Administrator Reference Manual, RFC 5011.

15.7. dnssec-importkey - import DNSKEY records from external systems so they can be managed.
15.8 dnssec-keyfromlabel - DNSSEC key generation tool

15.8.1 Synopsis


15.8.2 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.

15.8.3 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.

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.

This option is mandatory except when using the -S option, which copies the algorithm from the predecessory key.

Changed in version 9.12.0: The default value RSASHA1 for newly generated keys was removed.

-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).

-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.

-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.
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.)
15.8.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS (which is the format used inside key files), or ‘Day Mon DD HH:MM:SS YYYY’ (as printed by dnssec-settime -p), or UNIX epoch time (as printed by dnssec-settime -up), or the literal now.

The argument can be followed by + or - and an offset from the given time. The literal now can be omitted before an offset. The offset can be followed by one of the suffixes y, mo, w, d, h, or mi, so that it 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, never, or unset.

All these formats are case-insensitive.

-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.

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.)

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.
15.8.5 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.

15.8.6 See Also

`dnssec-keygen(8), dnssec-signzone(8), BIND 9 Administrator Reference Manual, RFC 4034, RFC 7512`

15.9 dnssec-keygen: DNSSEC key generation tool

15.9.1 Synopsis

```
```

15.9.2 Description

`dnssec-keygen` generates keys for DNSSEC (Secure DNS), as defined in RFC 2535 and RFC 4034.

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.

15.9.3 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 -3 -a 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, ECDSAP384SHA384, ED25519, or ED448.
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 ECD sap256SHA256, 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`).

\[-f\] flag

This option sets the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flags are ZSK (Zone-Signing Key), KSK (Key-Signing Key) and REVOKE.

Note that ZSK is not a physical flag in the DNSKEY record, it is merely used to explicitly tell that you want to create a ZSK. Setting \( -f \) in conjunction with \( -k \) will result in generating keys that only match the given role set with this option.

\[-F\]

This option turns on FIPS (US Federal Information Processing Standards) mode if the underlying cryptographic library supports running in FIPS mode.

\[-G\]

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-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.

15.9.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS (which is the format used inside key files), or ‘Day Mon DD HH:MM:SS YYYY’ (as printed by dnssec-settime -p), or UNIX epoch time (as printed by dnssec-settime -up), or the literal now.

The argument can be followed by + or - and an offset from the given time. The literal now can be omitted before an offset. The offset can be followed by one of the suffixes y, mo, w, d, h, or mi, so that it 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 unset a date, use none, never, or unset.

-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.

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.)

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.
15.9.5 Generated Keys

When `dnssec-keygen` completes successfully, it prints a string of the form `Knnnn.+aaa+iyyy` 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.
- `iiiiii` is the key identifier (or footprint).

`dnssec-keygen` creates two files, with names based on the printed string. `Knnnn.+aaa+iyyy.key` contains the public key, and `Knnnn.+aaa+iyyy.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.

15.9.6 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
```

15.9.7 See Also


15.10 `dnssec-ksr` - Create signed key response (SKR) files for offline KSK setups

15.10.1 Synopsis

```
```

15.10. `dnssec-ksr` - Create signed key response (SKR) files for offline KSK setups
15.10.2 Description

The `dnssec-ksr` can be used to issue several commands that are needed to generate presigned RRsets for a zone where the private key file of the Key Signing Key (KSK) is typically offline. This requires Zone Signing Keys (ZSKs) to be pregenerated, and the DNSKEY, CDNSKEY, and CDS RRsets to be already signed in advance.

The latter is done by creating Key Signing Requests (KSRs) that can be imported to the environment where the KSK is available. Once there, this program can create Signed Key Responses (SKRs) that can be loaded by an authoritative DNS server.

15.10.3 Options

`-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`).

`-e date/offset`

This option sets the end date for which keys or SKRs need to be generated (depending on the command).

`-F`

This option turns on FIPS (US Federal Information Processing Standards) mode if the underlying cryptographic library supports running in FIPS mode.

`-h`

This option prints a short summary of the options and arguments to `dnssec-ksr`.

`-i date/offset`

This option sets the start date for which keys or SKRs need to be generated (depending on the command).

`-K directory`

This option sets the directory in which the key files are to be read or written (depending on the command).

`-k policy`

This option sets the specific `dnssec-policy` for which keys need to be generated, or signed.

`-l file`

This option provides a configuration file that contains a `dnssec-policy` statement (matching the policy set with `-k`).

`-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.

`command`

The KSR command to be executed. See below for the available commands.

`zone`

The name of the zone for which the KSR command is being executed.
15.10.4 Commands

**keygen**

Pregenerate a number of zone signing keys (ZSKs), given a DNSSEC policy and an interval. The number of generated keys depends on the interval and the ZSK lifetime.

**request**

Create a Key Signing Request (KSR), given a DNSSEC policy and an interval. This will generate a file with a number of key bundles, where each bundle contains the currently published ZSKs (according to the timing metadata).

**sign**

Sign a Key Signing Request (KSR), given a DNSSEC policy and an interval, creating a Signed Key Response (SKR). This will add the corresponding DNSKEY, CDS, and CDNSKEY records for the KSK that is being used for signing.

15.10.5 Exit Status

The `dnssec-ksr` command exits 0 on success, or non-zero if an error occurred.

15.10.6 Examples

When you need to generate keys for the zone “example.com” for the next year, given a `dnssec-policy` named “my-policy”:

```
dnssec-ksr -i now -e +1y -k mypolicy -l named.conf keygen example.com
```

Creating a KSR for the same zone and period can be done with:

```
dnssec-ksr -i now -e +1y -k mypolicy -l named.conf request example.com > ksr.txt
```

Typically you would now transfer the KSR to the system that has access to the KSK.

Signing the KSR created above can be done with:

```
dnssec-ksr -i now -e +1y -k kskpolicy -l named.conf -f ksr.txt sign example.com
```

Make sure that the DNSSEC parameters in `kskpolicy` match those in `mypolicy`.

15.10.7 See Also

`dnssec-keygen(8), dnssec-signzone(8), BIND 9 Administrator Reference Manual.`

15.11 `dnssec-revoke` - set the REVOKE bit on a DNSSEC key

15.11.1 Synopsis

```
```
15.11.2 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.

15.11.3 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).
-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.

15.11.4 See Also


15.12 dnssec-settime: set the key timing metadata for a DNSSEC key

15.12.1 Synopsis

15.12.2 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.

15.12.3 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).
15.12.4 Timing Options

Dates can be expressed in the format YYYYMMDD or YYYYMMDDHHMMSS (which is the format used inside key files), or ‘Day Mon DD HH:MM:SS YYYY’ (as printed by dnssec-settime -p), or UNIX epoch time (as printed by dnssec-settime -up), or the literal now.

The argument can be followed by + or - and an offset from the given time. The literal now can be omitted before an offset. The offset can be followed by one of the suffixes y, mo, w, d, h, or mi, so that it 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 unset a date, use none, never, or unset.

All these formats are case-insensitive.

-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.

   ds date/offset
       This option sets the date on which DS records that match this key have been seen in the parent zone.

   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.)

   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.

   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.

### 15.12.5 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.

### 15.12.6 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`.

15.12. `dnssec-settime: set the key timing metadata for a DNSSEC key` 475
15.12.7 See Also

dnssec-keygen(8), dnssec-signzone(8), BIND 9 Administrator Reference Manual, RFC 5011.

15.13 dnssec-signzone - DNSSEC zone signing tool

15.13.1 Synopsis


15.13.2 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.

15.13.3 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 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).

-F
  This options turns on FIPS (US Federal Information Processing Standards) mode if the underlying cryptographic library supports running in FIPS mode.
-g
This option indicates that DS records for child zones should be generated from a dsset- or keyset- file. Existing DS records are removed.

-G sync-records
This option indicates which CDS and CDNSKEY records should be generated. sync-records is a comma-separated string with the following allowed items: cdnshkey, and cds:<digest-type>, where digest-type is an allowed algorithm such as SHA-256 (2), or SHA-384 (4). Only works in combination with smart signing (-S).

-J filename
This option tells dnssec-signzone to read the journal from the given file when loading the zone file.

-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.

-L serial
When writing a signed zone to “raw” 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.
This option sets the zone origin. If not specified, the name of the zone file is assumed to be the origin.

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 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.

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.

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 6781#4.1.1.1 (“Pre-Publish Key Rollover”).

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.

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 6781#4.1.1.2 (“Double Signature Zone Signing Key Rollover”).

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 unpublishation 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 \texttt{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, \texttt{dnssec-signzone} retains the existing chain when re-signing.

\-v \texttt{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.

\-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.

\-3 \texttt{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.

\textbf{Note:} -3 - is the recommended configuration. Adding salt provides no practical benefits. See RFC 9276.

\-H \texttt{iterations}

This option indicates that, when generating an NSEC3 chain, BIND 9 should use this many iterations. The default is 0.

\textbf{Warning:} Values greater than 0 cause interoperability issues and also increase the risk of CPU-exhausting DoS attacks. See RFC 9276.

\-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.

\textbf{Warning:} Do not use this option unless all its implications are fully understood. This option is intended only for extremely large zones (comparable to com.) with sparse secure delegations. See RFC 9276.
**-AA**

This option 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.

### 15.13.4 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`).

```bash
% 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.

```bash
% cp db.example.com.signed db.example.com
% dnssec-signzone -o example.com db.example.com
% db.example.com.signed
%
```

### 15.13.5 See Also


### 15.14 dnssec-verify - DNSSEC zone verification tool

#### 15.14.1 Synopsis

```
```
15.14.2 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.

15.14.3 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).

-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.

-J filename
   This option tells dnssec-verify to read the journal from the given file when loading the zone file.

-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.
15.14.4 See Also


15.15 dnstap-read - print dnstap data in human-readable form

15.15.1 Synopsis

dnstap-read [-m] [-p] [-x] [-y] {file}

15.15.2 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.

15.15.3 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.

- **-t**
  This option prints long timestamps with millisecond precision.

- **-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.

15.15.4 See Also

named(8), rndc(8), BIND 9 Administrator Reference Manual.

15.16 filter-aaaa.so - filter AAAA in DNS responses when A is present

15.16.1 Synopsis

plugin query "filter-aaaa.so" [{ parameters }];
15.16.2 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 no longer available in `named.conf` but can be passed as parameters to the `filter-aaaa.so` plugin, for example:

```plaintext
plugin query "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.

15.16.3 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`.

15.16.4 See Also

15.17 host - DNS lookup utility

15.17.1 Synopsis

```
```

15.17.2 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`.

15.17.3 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.
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.

This option specifies the port to query on the server. The default is 53.

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.

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.

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.

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.

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.

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.

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.

This option prints the version number and exits.

This option sets “wait forever”: the query timeout is set to the maximum possible. See also the -W option.

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.

### 15.17.4 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.

### 15.17.5 Files

`/etc/resolv.conf`

### 15.17.6 See Also

`dig(1), named(8)`.

### 15.18 mdig - DNS pipelined lookup utility

#### 15.18.1 Synopsis

```bash
mdig [-h]
mdig [@server] {global-opt...} {local-opt...} {query} ...
```

#### 15.18.2 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.

### 15.18.3 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.

### 15.18.4 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:

- **+additional, +noadditional**
  This option displays [or does not display] the additional section of a reply. The default is to display it.

- **+all, +noall**
  This option sets or clears all display flags.

- **+answer, +noanswer**
  This option displays [or does not display] the answer section of a reply. The default is to display it.

- **+authority, +noauthority**
  This option displays [or does not display] the authority section of a reply. The default is to display it.

- **+besteffort, +nobesteffort**
  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.

+cl, +nocl
   This option displays [or does not display] the CLASS when printing the record.

+comments, +nocomments
   This option toggles the display of comment lines in the output. The default is to print comments.

+continue, +nocontinue
   This option toggles continuation on errors (e.g. timeouts).

+crypto, +nocrypto
   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 ].

+multiline, +nomultiline
   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.

+question, +noquestion
   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.

+rrcomments, +norrcomments
   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.

+short, +noshort
   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.

+tcp, +notcp
   This option uses [or does not use] TCP when querying name servers. The default behavior is to use UDP.

+ttmlid, +nottmlid
   This option displays [or does not display] the TTL when printing the record.

+ttmlunits, +nottmlunits
   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 +ttmlid.

+vc, +novc
   This option uses [or does not use] TCP when querying name servers. This alternate syntax to +tcp is provided for backwards compatibility. The vc stands for “virtual circuit”.

15.18. mdig - DNS pipelined lookup utility
15.18.5 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:

+aaflag, +noaaflag
This is a synonym for +aalone, +noaalone.

+aalone, +noaalone
This sets the aa flag in the query.

+adflag, +noadflag
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 an EDNS query to be sent.

+cdflag, +nocdflag
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.

+cookie=####, +nocookie
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.

+dnssec, +nodnssec
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.

+edns[=#], +noedns
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.

+ednsflags[=#], +noednsflags
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.
+ednsopt[=code[:value]], +noednsopt
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.

+expire, +noexpire
This toggles sending of an EDNS Expire option.

+nsid, +nonsid
This toggles inclusion of an EDNS name server ID request when sending a query.

+recurse, +norecurse
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.

+subnet=addr[/prefix-length], +nosubnet
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.

+unknownformat, +nounknownformat
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.

+yaml, +noyaml
This toggles printing of the responses in a detailed YAML format.

+zflag, +nozflag
This sets [or does not set] the last unassigned DNS header flag in a DNS query. This flag is off by default.

15.18.6 See Also
dig(1), RFC 1035.
15.19 named-checkconf - named configuration file syntax checking tool

15.19.1 Synopsis

`named-checkconf [-achjlvz] [-p [-x ]] [-t directory] [filename]`

15.19.2 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.conf` or `rndc.key`, 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.

15.19.3 Options

- `-a`
  
  Don't check the `dnssec-policy`'s DNSSEC key algorithms against those supported by the crypto provider. This is useful when checking a `named.conf` intended to be run on another machine with possibly a different set of supported DNSSEC key algorithms.

- `-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.
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.

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.

15.19.4 Return Values

named-checkconf returns an exit status of 1 if errors were detected and 0 otherwise.

15.19.5 See Also

named(8), named-checkzone(8), BIND 9 Administrator Reference Manual.

15.20 named-checkzone - zone file validation tool

15.20.1 Synopsis


15.20.2 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.

15.20.3 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.

-C mode
This option controls check mode on zone files when loading. Possible modes are check-svcb:fail and check-svcb:ignore.

check-svcb:fail turns on additional checks on _dns SVCB records and check-svcb:ignore disables these checks. The default is check-svcb:fail.

-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), and raw.

-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 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.
When compiling a zone to raw 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.

This option specifies whether MX records should be checked to see if they are addresses. Possible modes are fail, warn (the default), and ignore.

This option checks whether a MX record refers to a CNAME. Possible modes are fail, warn (the default), and ignore.

This option specifies whether NS records should be checked to see if they are addresses. Possible modes are fail, warn (the default), and ignore.

This option writes the zone output to filename. If filename is -, then the zone output is written to standard output.

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.

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.

This option checks whether an SRV record refers to a CNAME. Possible modes are fail, warn (the default), and ignore.

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.

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.

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.

This option dumps the zone file in canonical format.

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.

This indicates the domain name of the zone being checked.
filename
This is the name of the zone file.

15.20.4 Return Values

named-checkzone returns an exit status of 1 if errors were detected and 0 otherwise.

15.20.5 See Also

named(8), named-checkconf(8), named-compilezone(8), RFC 1035, BIND 9 Administrator Reference Manual.

15.21 named-compilezone - zone file converting tool

15.21.1 Synopsis


15.21.2 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.

Unlike named-checkzone, zone contents are not strictly checked by default. If the output is to be used as an actual zone file to be loaded by named, then the check levels should be manually configured to be at least as strict as those specified in the named configuration file.

Running named-checkzone on the input prior to compiling will ensure that the zone compiles with the default requirements of named.

15.21.3 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.

-C mode
   This option controls check mode on zone files when loading. Possible modes are check-svcb:fail and check-svcb:ignore.
   check-svcb:fail turns on additional checks on _dns SVCB records and check-svcb:ignore disables these checks. The default is check-svcb:ignore.

-i mode
   This option performs post-load zone integrity checks. Possible modes are full, full-sibling, local, local-sibling, and none (the default).
   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), and raw.

-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 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, and ignore (the default).

-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 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.
This option specifies whether MX records should be checked to see if they are addresses. Possible modes are fail, warn, and ignore (the default).

This option checks whether a MX record refers to a CNAME. Possible modes are fail, warn, and ignore (the default).

This option specifies whether NS records should be checked to see if they are addresses. Possible modes are fail, warn, and ignore (the default).

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.

This option checks for records that are treated as different by DNSSEC but are semantically equal in plain DNS. Possible modes are fail, warn, and ignore (the default).

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 option checks whether an SRV record refers to a CNAME. Possible modes are fail, warn, and ignore (the default).

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.

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 and ignore (the default).

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.

This option dumps the zone file in canonical format. This is always enabled for named-compilezone.

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 and ignore (the default).

This indicates the domain name of the zone being checked.

This is the name of the zone file.
15.21.4 Return Values

`named-compilezone` returns an exit status of 1 if errors were detected and 0 otherwise.

15.21.5 See Also

`named(8), named-checkconf(8), named-checkzone(8), RFC 1035, BIND 9 Administrator Reference Manual`

15.22 named-journalprint - print zone journal in human-readable form

15.22.1 Synopsis

`named-journalprint [-c serial] [-dux] {journal}`

15.22.2 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.
15.22.3 See Also


15.23 named-nzd2nzd - convert an NZD database to NZF text format

15.23.1 Synopsis

`named-nzd2nzd [filename]`

15.23.2 Description

`named-nzd2nzd` 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.

15.23.3 Arguments

`filename`

This is the name of the `.nzd` file whose contents should be printed.

15.23.4 See Also


15.24 named-rrchecker - syntax checker for individual DNS resource records

15.24.1 Synopsis


15.24.2 Description

`named-rrchecker` reads an individual DNS resource record from standard input and checks whether it is syntactically correct.
15.24.3 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, -P
  These options print out the known class, standard type, and private type mnemonics, respectively.

15.24.4 See Also

RFC 1034, RFC 1035, named(8).

15.25 named.conf - configuration file for named

15.25.1 Synopsis

named.conf

15.25.2 Description

named.conf is the configuration file for named.

For complete documentation about the configuration statements, please refer to the Configuration Reference section in the BIND 9 Administrator Reference Manual.

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 <string> { <address_match_element>; ... }; // may occur multiple times

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

(continues on next page)
dlz <string> {  
database <string>;  
search <boolean>;  
}; // may occur multiple times

dnssec-policy <string> {  
cdnskey <boolean>;  
cds-digest-types { <string>; ... };  
dnskey-ttl <duration>;  
inline-signing <boolean>;  
keys { ( csk | ksk | zsk ) [ key-directory | key-store <string> ] 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-jitter <duration>;  
signatures-refresh <duration>;  
signatures-validity <duration>;  
signatures-validity-dnskey <duration>;  
zone-propagation-delay <duration>;  
}; // may occur multiple times

dyndb <string> <quoted_string> { <unspecified-text> }; // may occur multiple times

http <string> {  
endpoints { <quoted_string>; ... };  
lsteller-clients <integer>;  
streams-per-connection <integer>;  
}; // may occur multiple times

key <string> {  
algorithm <string>;  
secret <string>;  
}; // may occur multiple times

key-store <string> {  
directory <string>;  
pkcs11-uri <quoted_string>;  
}; // may occur multiple times

logging {  
category <string> { <string>; ... }; // may occur multiple times  
channel <string> {  
buffered <boolean>;  
file <quoted_string> [ versions ( unlimited | <integer> ) ] [ size  
←<size> ] [ suffix ( increment | timestamp ) ];  
nul1;  
print-category <boolean>;  
print-severity <boolean>;  
print-time ( iso8601 | iso8601-utc | local | <boolean> );  
severity <log_severity>;  
}
mandated-keys { ( static-key | initial-key | static-ds | initial-ds ) <integer> <integer> <integer> quoted-string; ... }; // may occur multiple times, deprecated

options {
  allow-new-zones <boolean>;
  allow-query { <address_match_element>; ... };
  allow-proxy { <address_match_element>; ... }; // experimental
  allow-proxy-on { <address_match_element>; ... };
  allow-query-on { <address_match_element>; ... };
  allow-recursion { <address_match_element>; ... };
  allow-recursion-on { <address_match_element>; ... };
  allow-transfer { port <integer> } [ transport string ]; { <address_match_element> ... };
  also-notify { port <integer> } [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key string ] [ tls string ] } ... ;
  answer-cookie <boolean>;
  attach-cache <string>;
  automatic-interface-scan <boolean>;
  avoid-v4-udp-ports { <portrange>; ... }; // deprecated
  avoid-v6-udp-ports { <portrange>; ... }; // deprecated
  bindkeys-file quoted_string; // test only
  blackhole { <address_match_element>; ... };
  catalog-zones { zone string [ default-primaries [ port <integer> ] [ source-v4 ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] [ ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key string ] [ tls 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-svcb <boolean>;
  check-wildcard <boolean>;
  clients-per-query <integer>;
  cookie-algorithm ( siphash24 );
  cookie-secret <string>; // may occur multiple times
  deny-answer-addresses { <address_match_element>; ... } [ except-from { <string> ... } ];
  deny-answer-aliases { <string> ... } [ except-from { <string> ... } ];
  deny-query-addresses { <address_match_element>; ... } [ except-from { <string> ... } ];
  deny-query-aliases { <string> ... } [ except-from { <string> ... } ];
  deny-query-chains { <string> ... } [ except-from { <string> ... } ];
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
  deny-query-chains-on { <address_match_element>; ... };
}; // may occur multiple times

stderr;
  syslog [ <syslog_facility> ];
}; // may occur multiple times

(continues from previous page)
dialup ( notify | notify-passive | passive | refresh | <boolean> ); // _

--deprecated
directory <quoted_string>;
disable-algorithms <string> { <string>; ... }; // may occur multiple times
disable-ds-digests <string> { <string>; ... }; // may occur multiple times
disable-empty-zone <string>; // may occur multiple times
dns64 <netprefix> {
  break-dnssec <boolean>;
  clients { <address_match_element>; ... };
  exclude { <address_match_element>; ... };
  mapped { <address_match_element>; ... };
  recursive-only <boolean>;
  suffix <ipv6_address>;
}; // may occur multiple times
dns64-contact <string>;
dns64-server <string>;
dnskey-sig-validity <integer>; // obsolete
dnsrsps-enable <boolean>; // not configured
dnsrsps-library <quoted_string>; // not configured
dnsrsps-options { <unspecified-text> }; // not configured
dnssec-accept-expired <boolean>;
dnssec-dnskey-kskonly <boolean>; // obsolete
dnssec-loadkeys-interval <integer>;
dnssec-must-be-secure <string> <boolean>; // may occur multiple times,

--deprecated
dnssec-policy <string>;
dnssec-secure-to-insecure <boolean>; // obsolete
dnssec-update-mode ( maintain | no-resign ); // obsolete
dnssec-validation ( yes | no | auto );
dnstap { ( all | auth | client | forwarder | resolver | update ) [ ( query | _
--response ) ]; ... }; // not configured
dnstap-identity ( <quoted_string> | none | hostname ); // not configured
dnstap-output ( file | unix ) <quoted_string> [ size ( unlimited | <size> ) ];
--[ versions ( unlimited | <integer> ) ] [ suffix ( increment | timestamp ) ]; // not...
--configured
dnstap-version ( <quoted_string> | none ); // not configured
dual-stack-servers [ port <integer> ] { ( ( <quoted_string> [ port <integer> ] _
-- | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <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 } ];
flush-zones-on-shutdown <boolean>;
forward ( first | only );
forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_
--address> ) [ port <integer> ] [ tls <string> ]; ... };
fstrm-set-buffer-hint <integer>; // not configured
fstrm-set-flush-timeout <integer>; // not configured
fstrm-set-input-queue-size <integer>; // not configured
fstrm-set-output-notify-threshold <integer>; // not configured
fstrm-set-output-queue-model ( mpsc | spsc ); // not configured
fstrm-set-output-queue-size <integer>; // not configured
(continues on next page)
geoip-directory ( <quoted_string> | none );
heartbeat-interval <integer>; // deprecated
hostname ( <quoted_string> | none );
http-listener-clients <integer>;
http-port <integer>;
http-streams-per-connection <integer>;
https-port <integer>;
interface-interval <duration>;
ipv4only-contact <string>;
ipv4only-enable <boolean>;
ipv4only-server <string>;
ixfr-from-differences { primary | master | secondary | slave | <boolean> };
keep-response-order { <address_match_element>; ... }; // obsolete
key-directory <quoted_string>;
lame-ttl <duration>;
listen-on [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http --> <string> ] [ <address_match_element>; ... ]; // may occur multiple times
listen-on-v6 [ port <integer> ] [ proxy <string> ] [ tls <string> ] [ http --> <string> ] [ <address_match_element>; ... ]; // may occur multiple times
lmdb-mapsize <sizeval>;
managed-keys-directory <quoted_string>;
masterfile-format ( raw | text );
masterfile-style ( full | relative );
matched-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-records-per-type <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-types-per-name <integer>;
max-udp-size <integer>;
max-validation-failures-per-fetch <integer>; // experimental
max-validations-per-fetch <integer>; // experimental
max-zone-ttl { unlimited | <duration> }; // deprecated
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>;

(continues on next page)
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> | * };
notify-source-v6 { <ipv6_address> | * };
notify-to-soa <boolean>;
nsec3-test-zone <boolean>; // test only
nta-lifetime <duration>;
nta-recheck <duration>;
nxdomain-redirect <string>;
parental-source { <ipv4_address> | * };
parental-source-v6 { <ipv6_address> | * };
pid-file { <quoted_string> | none };
port <integer>;
pref-file <boolean>;
pref-file <string>;
pref-fetch <integer> [ <integer> ];
provide-ixfr <boolean>;
qname-minimization { strict | relaxed | disabled | off };
query-source { <ipv4_address> | * };
query-source-v6 { <ipv6_address> | * };
querylog <boolean>;
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>;
nxdomain-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>;
resolver-query-timeout <integer>;
resolver-use-dns64 <boolean>;
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> ] [ ede <string> ]; ... } [ add-soa <boolean> ] [ break-dnssec <boolean> ] [ max-policy-ttl <duration> ] [ min-update-interval <duration> ] [ min-ns-dots
<integer>] [ nslp-wait-recurse <boolean> ] [ nsdname-wait-recurse <boolean> ] [ qname-wait-recurse <boolean> ] [ recursive-only <boolean> ] [ nslp-enable <boolean> ] [ nsdname-enable <boolean> ] [ dnsrps-enable <boolean> ] [ dnsrps-options { ...
- <specified-text> } ];
- reuseport <boolean>;
- root-key-sentinel <boolean>;
- rsset-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> ]; // obsolete
- sig0checks-quota <integer>; // experimental
- sig0checks-quota-exempt { <address_match_element>; ... }; // experimental
- sortlist { <address_match_element>; ... }; // deprecated
- 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>;
- tcp-receive-buffer <integer>;
- tcp-send-buffer <integer>;
- tkey-domain <quoted_string>;
- tkey-gssapi-credential <quoted_string>;
- tkey-gssapi-keytab <quoted_string>;
- tls-port <integer>;
- transfer-format ( many-answers | one-answer );
- transfer-message-size <integer>;
- transfer-source { <ipv4_address> | * };
- transfer-source-v6 { <ipv6_address> | * };
- transfers-in <integer>;
- transfers-out <integer>;
- transfers-per-ns <integer>;
- trust-anchor-telemetry <boolean>;
- try-tcp-refresh <boolean>;
- udp-receive-buffer <integer>;
- udp-send-buffer <integer>;
- update-check-ksk <boolean>; // obsolete
- update-quota <integer>;

(continues on next page)
use-v4-udp-ports { <portrange>; ... }; // deprecated
use-v6-udp-ports { <portrange>; ... }; // deprecated
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 <string> [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... } // may occur multiple times

plugin ( query ) <string> [ { <unspecified-text> } ]; // may occur multiple times

primaries <string> [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... } // may occur multiple times

server <netprefix> {
  bogus <boolean>;
edns <boolean>;
edns-udp-size <integer>;
edns-version <integer>;
keys <server_key>;
max-udp-size <integer>;
notify-source ( <ipv4_address> | * );
notify-source-v6 ( <ipv6_address> | * );
padding <integer>;
provide-ixfr <boolean>;
query-source [ address ] ( <ipv4_address> | * );
query-source-v6 [ address ] ( <ipv6_address> | * );
request-expire <boolean>;
request-ixfr <boolean>;
request-nsid <boolean>;
require-cookie <boolean>;
send-cookie <boolean>;
tcp-keepalive <boolean>;
tcp-only <boolean>;
transfer-format ( many-answers | one-answer );
transfer-source ( <ipv4_address> | * );
transfer-source-v6 ( <ipv6_address> | * );
transfers <integer>;
}; // may occur multiple times

statistics-channels {
  inet ( <ipv4_address> | <ipv6_address> | * ) [ port ( <integer> | * ) ] [ allow { <address_match_element>; ... } ]; // may occur multiple times
}; // may occur multiple times

tls <string> {
  ca-file <quoted_string>;
cert-file <quoted_string>;
cipher-suites <string>;
} (continues on next page)
ciphers <string>;
dhparam-file <quoted_string>;
key-file <quoted_string>;
prefer-server-ciphers <boolean>;
protocols { <string>; ... };
remote-hostname <quoted_string>;
session-tickets <boolean>;
}; // may occur multiple times

trust-anchors { <string> { static-key | initial-key | static-ds | initial-ds }<integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times

trusted-keys { <string> <integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times, deprecated

view <string> [ <class> ] {
  allow-new-zones <boolean>;
  allow-notify { <address_match_element>; ... };
  allow-proxy { <address_match_element>; ... } // experimental
  allow-proxy-on { <address_match_element>; ... } // experimental
  allow-query { <address_match_element>; ... };
  allow-query-cache { <address_match_element>; ... };
  allow-query-on { <address_match_element>; ... };
  allow-recursion { <address_match_element>; ... };
  allow-recursion-on { <address_match_element>; ... };
  allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_element>; ... };
  allow-update { <address_match_element>; ... };
  allow-update-forwarding { <address_match_element>; ... };
  also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 <ipv6_address> | * ] 
  { ( <remote-servers> | <ipv4_address> [ port <integer> ] | 
  <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ] } ... };
  attach-cache <string>;
  auth-nxdomain <boolean>;
  catalog-zones { zone <string> [ default-primaries [ port <integer> ] [ source-v4 <ipv4_address> | * ] ] [ source-v6 <ipv6_address> | * ] 
  { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <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 ); // may occur multiple times
  check-sibling <boolean>;
  check-spf ( warn | ignore );
  check-srv-cname ( fail | warn | ignore );
  check-svcb <boolean>;
  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> ); // deprecated
disable-algorithms <string> { <string>; ... }; // may occur multiple times
disable-ds-digests <string> { <string>; ... }; // may occur multiple times
disable-empty-zone <string>; // may occur multiple times
dlz <string> {
    database <string>;
    search <boolean>;
}; // may occur multiple times
dns64 <netprefix> {
    break-dnssec <boolean>;
    clients { <address_match_element>; ... };
    exclude { <address_match_element>; ... };
    mapped { <address_match_element>; ... };
    recursive-only <boolean>;
    suffix <ipv6_address>;
}; // may occur multiple times
dns64-contact <string>;
dns64-server <string>;
dnskey-sig-validity <integer>; // obsolete
dnspb-enable <boolean>; // not configured
dnspb-options { <unspecified-text> }; // not configured
dnssc-accept-expired <boolean>;
dnssc-dnskey-kskonly <boolean>; // obsolete
dnssc-loadkeys-interval <integer>;
dnssc-must-be-secure <string> <boolean>; // may occur multiple times;
   -- deprecated
    dnssc-policy <string>;
    dnssc-secure-to-insecure <boolean>; // obsolete
dnssc-update-mode ( maintain | no-resign ); // obsolete
dnssc-validation ( yes | no | auto );
    dnstap { ( all | auth | client | forwarder | resolver | update ) [ ( query | ...
   -- response ) ]; ... ); // not configured
dual-stack-servers [ port <integer> ] { ( <quoted_string> [ port <integer> ] ... |
ipv4_address > [ port <integer> ] | <ipv6_address> [ port <integer> ] ); ... };
dyndb <string> <quoted_string> { <unspecified-text> }; // may occur multiple times;
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> } [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
    ipv4only-contact <string>;
    ipv4only-enable <boolean>;
    ipv4only-server <string>;
    ixfr-from-differences ( primary | master | secondary | slave | <boolean> );
    key <string> {
        algorithm <string>;
        secret <string>;
    }; // may occur multiple times
    key-directory <quoted_string>;
    lame-ttl <duration>;
    lmdb-mapsize <sizeval>;
    managed-keys { <string> ( static-key | initial-key | static-ds | initial-ds )
(continues on next page)
→<integer> <integer> <integer> <quoted_string>; ... }; // may occur multiple times,
→deprecated
masterfile-format ( 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-recurses <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-types-per-name <integer>;
max-udp-size <integer>;
max-validation-failures-per-fetch <integer>; // experimental
max-validations-per-fetch <integer>; // experimental
max-zone-ttl { unlimited | <duration> }; // deprecated
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>;
nzds { <address_match_element>; ... };
nocookie-udp-size <integer>;
nocoop-recheck <duration>;
nocoop-recheck <duration>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;
nxdomain-redirect <string>;

15.25. named.conf - configuration file for named
(continues on next page)
query-source [ address ] ( <ipv4_address> | * );
query-source-v6 [ address ] ( <ipv6_address> | * );
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-query-timeout <integer>
resolver-use-dns64 <boolean>
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 | nsdomain | passthru | tcp-only <quoted_
  string> ) ] [ recursive-only <boolean> ] [ nsname-enable <boolean> ] [ nsdname-enable
  <boolean> ] [ ede <string> ]; ... } [ add-soa <boolean> ] [ break-dnssec <boolean>
  disabled | drop | given | no-op | nodata | nsdomain | passthru | tcp-only <quoted_
  string> ) ] [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [ nsip-wait-
  recurse <boolean> ] [ recursive-only <boolean> ] [ nsip-enable <boolean> ] [ nsip-wait-
  recurse <boolean> ] [ nsname-enable <boolean> ] [ nsdname-enable <boolean> ] [ dnsrps-enable <boolean> ] [ dnsrps-options {
  <unspecified-text> } ]
  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> | * )
  notify-source-v6 ( <ipv6_address> | * )
  padding <integer>
  provide-ixfr <boolean>
  query-source [ address ] ( <ipv4_address> | * )
  query-source-v6 [ address ] ( <ipv6_address> | * )
  request-expire <boolean>
  request-ixfr <boolean>
}
request-nsid <boolean>;
require-cookie <boolean>;
send-cookie <boolean>;
tcp-keepalive <boolean>;
tcp-only <boolean>;
transfer-format { many-answers | one-answer };
transfer-source { <ipv4_address> | * };
transfer-source-v6 { <ipv6_address> | * };
transfers <integer>;
} // may occur multiple times
servfail-ttl <duration>;
sig-signing-nodes <integer>;
sig-signing-signatures <integer>;
sig-signing-type <integer>;
sig-validity-interval <integer> [ <integer> ]; // obsolete
sortlist { <address_match_element>; ... } // deprecated
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> | * };
transfer-source-v6 { <ipv6_address> | * };
trust-anchor-telemetry <boolean>;
trust-anchors { <string> { static-key | initial-key | static-ds | initial-ds
→<integer> <integer> <integer> <quoted_string>; ... } // may occur multiple times
trusted-keys { <string> <integer> <integer> <integer> <quoted_string>; ... };
→// may occur multiple times, deprecated
try-tcp-refresh <boolean>;
update-check-ksk <boolean>; // obsolete
v6-bias <integer>;
validate-except { <string>; ... };
zero-no-soa-ttl <boolean>;
zero-no-soa-ttl-cache <boolean>;
zone-statistics { full | terse | none | <boolean> };
} // may occur multiple times

Any of these zone statements can also be set inside the view statement.

zone <string> [ <class> ] {
  type primary;
  allow-query { <address_match_element>; ... };
  allow-query-on { <address_match_element>; ... };
  allow-transfer [ port <integer> ] [ transport <string> ] { <address_match_→element>; ... };
  allow-update { <address_match_element>; ... };
  also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6
→{ <ipv6_address> | * } ] { { <remote-servers> | <ipv4_address> [ port <integer> ] |
→<ipv6_address> [ port <integer> ] } [ key <string> ] [ tls <string> ]; ... };
  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 );
}; // may occur multiple times

(continues on next page)
check-sibling <boolean>
check-spf ( warn | ignore )
check-srv-cname ( fail | warn | ignore )
check-svcb <boolean>
check-wildcard <boolean>
checkdns ( explicit | <boolean>

debug <string>

dialup ( notify | notify-passive | passive | refresh | <boolean> ); //

--deprecated

dlz <string>
dnskey-sig-validity <integer> // obsolete
dnssec-dnskey-kskonly <boolean> // obsolete
dnssec-loadkeys-interval <integer>
dnssec-policy <string>
dnssec-secure-to-insecure <boolean> // obsolete
dnssec-update-mode ( maintain | no-resign ); // obsolete

dnsfile <quoted_string>
forward ( first | only );
forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_
address> ) [ port <integer> ] [ tls <string> ]; ... }

inline-signing <boolean>
ixfr-from-differences <boolean>
journal <quoted_string>
key-directory <quoted_string>
masterfile-format ( raw | text );
masterfile-style ( full | relative )
max-ixfr-ratio ( unlimited | <percentage> );
max-journal-size ( default | unlimited | <sizeval> )
max-records <integer>
max-records-per-type <integer>
max-transfer-idle-out <integer>
max-transfer-time-out <integer>
max-types-per-name <integer>

max-zone-ttl ( unlimited | <duration> ); // deprecated
notify ( explicit | master-only | primary-only | <boolean> )
notify-delay <integer>
notify-source ( <ipv4_address> | * )
notify-source-v6 ( <ipv6_address> | * )
notify-to-soa <boolean>
nsec3-test-zone <boolean>; // test only
parental-agents [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-
v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer>]
address ) [ port <integer> ] [ key <string> ] [ tls <string> ]; ... }
parental-source ( <ipv4_address> | * )
parental-source-v6 ( <ipv6_address> | * )
serial-update-method ( date | increment | unixtime )
sig-signing-nodes <integer>
sig-signing-signatures <integer>
sig-signing-type <integer>
sig-validity-interval <integer> [ <integer> ]; // obsolete

update-check-ksk <boolean> // obsolete
update-check-policy ( local | { ( deny | grant ) <string> ( 6to4-self | external |
krb5-self | krb5-selfsubs | krb5-subdomain | krb5-subdomain-self-rhs | ms-self | ms-
selfsub | ms-subdomain | ms-subdomain-self-rhs | name | self | selfsub | selfwild |
subdomain | tcp-self | wildcard | zonesub ) [ <string> ] [ rrtypelist; ... ] )

zero-no-soa-ttl <boolean>
zone-statistics ( full | terse | none | <boolean> );

(continues from previous page)
zone <string> [ <class> ] {
  type secondary;
  allow-query { <address_match_element>; ... };
  allow-transfer { port <integer> } [ transport <string> ] { <address_match_element>; ... };
  allow-update-forwarding { <address_match_element>; ... };
  also-notify [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6-<ipv6_address> | * ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };
  check-names ( fail | warn | ignore );
  checks ( explicit | <boolean> );
  database <string>;
  dialup ( notify | notify-passive | passive | refresh | <boolean> ); //...
  deprecated
dlz <string>;
dnskey-sig-validity <integer>; // obsolete
dnssec-dnskey-kskonly <boolean>; // obsolete
dnssec-loadkeys-interval <integer>;
dnssec-policy <string>;
dnssec-update-mode ( maintain | no-resign ); // obsolete
d file <quoted_string>;
forward ( first | only );
forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
inline-signing <boolean>;
ixfr-from-differences <boolean>;
journal <quoted_string>;
key-directory <quoted_string>;
masterfile-format ( raw | text );
masterfile-style ( full | relative );
max-ixfr-ratio ( unlimited | <percentage> );
max-journal-size ( default | unlimited | <sizeval> );
max-records <integer>;
max-records-per-type <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-types-per-name <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> | * );
notify-source-v6 ( <ipv6_address> | * );
notify-to-soa <boolean>;
nsec3-test-zone <boolean>; // test only
parental-agents [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };
zone <string> [ <class> ] {  
    type mirror;
    allow-notify { <address_match_element>; ... };
    allow-query { <address_match_element>; ... };
    allow-query-on { <address_match_element>; ... };
    allow-transfer { port <integer> } [ transport <string> ] { <address_match_element>; ... };
    allow-update-forwarding { <address_match_element>; ... };
    also-notify { port <integer> } [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { <remote-servers> | <ipv4_address> [ port <integer> ] | <ipv6_address> [ port <integer> ] } [ key <string> ] [ tls <string> ]; ... };
    check-names ( fail | warn | ignore );
    database <string>;
    file <quoted_string>;
    iffr-from-differences <boolean>;
    journal <quoted_string>;
    masterfile-format ( raw | text );
    masterfile-style ( full | relative );
    max-iffr-ratio ( unlimited | <percentage> );
    max-journal-size ( default | unlimited | <sizeval> );
    max-records <integer>;
    max-records-per-type <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-types-per-name <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> | * };
    notify-source-v6 { <ipv6_address> | * };
}

(continued from previous page)

(continues on next page)
(continued from previous page)

```plaintext
zone <string> [ <class> ] {
  type forward;
  forward (first | only);
  forwarders [port <integer>] [tls <string>] { ( <ipv4_address> | <ipv6_address> ) [port <integer>] [tls <string>]; ... }
};
```

```plaintext
zone <string> [ <class> ] {
  type hint;
  check-names {fail | warn | ignore};
  file <quoted_string>;
};
```

```plaintext
zone <string> [ <class> ] {
  type redirect;
  allow-query { <address_match_element>; ... }
  allow-query-on { <address_match_element>; ... }
  dlz <string>;
  file <quoted_string>;
  masterfile-format {raw | text};
  masterfile-style {full | relative};
  max-records <integer>;
  max-records-per-type <integer>;
  max-types-per-name <integer>;
  max-zone-ttl {unlimited | <duration>}; // deprecated
  primaries [port <integer>] [source {<ipv4_address> | *}] [source-v6 {<ipv6_address> | *}] { <remote-servers> | <ipv4_address> [port <integer>] | <ipv6_address> [port <integer>] } [key <string>] [tls <string>]; ... }
  zone-statistics {full | terse | none | <boolean>};
};
```

```plaintext
zone <string> [ <class> ] {
  type static-stub;
  allow-query { <address_match_element>; ... }
  allow-query-on { <address_match_element>; ... }
  forward (first | only);
  forwarders [port <integer>] [tls <string>] { ( <ipv4_address> | <ipv6_address> ) [port <integer>] [tls <string>]; ... }
  max-records <integer>;
  max-records-per-type <integer>;
  max-types-per-name <integer>;
  server-addresses { ( <ipv4_address> | <ipv6_address> ); ... }
  server-names { <string>; ... }
};
```

(continues on next page)
zone <string> [ <class> ] {
  type stub;
  allow-query { <address_match_element>; ... };
  allow-query-on { <address_match_element>; ... };
  check-names { fail | warn | ignore };
  database <string>;
  dialup ( notify | notify-passive | passive | refresh | <boolean> ); //...
  —deprecated
  file <quoted_string>;
  forward ( first | only );
  forwarders [ port <integer> ] [ tls <string> ] { ( <ipv4_address> | <ipv6_address> ) [ port <integer> ] [ tls <string> ]; ... };
  masterfile-format ( raw | text );
  masterfile-style ( full | relative );
  max-records <integer>;
  max-records-per-type <integer>;
  max-refresh-time <integer>;
  max-retry-time <integer>;
  max-transfer-idle-in <integer>;
  max-transfer-time-in <integer>;
  max-types-per-name <integer>;
  min-refresh-time <integer>;
  min-retry-time <integer>;
  multi-master <boolean>;
  primaries [ port <integer> ] [ source ( <ipv4_address> | * ) ] [ source-v6 ( <ipv6_address> | * ) ] { ( <remote-servers> | <ipv4_address> ) [ port <integer> ] | ( <ipv6_address> ) [ port <integer> ] ) [ key <string> ] [ tls <string> ]; ... };
  transfer-source { <ipv4_address> | * };
  transfer-source-v6 { <ipv6_address> | * };
  zone-statistics ( full | terse | none | <boolean> );
};

zone <string> [ <class> ] {
  in-view <string>;
};

15.25.3 Files
/etc/named.conf

15.25.4 See Also
named(8), named-checkconf(8), rndc(8), rndc-confgen(8), tsig-keygen(8), BIND 9 Administrator Reference Manual.
15.26 named - Internet domain name server

15.26.1 Synopsis

```
```

15.26.2 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.

15.26.3 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 a possible directory option in the configuration file, config-file should be an absolute pathname.

- `-C`
  This option prints out the default built-in configuration and exits.

  NOTE: This is for debugging purposes only and is not an accurate representation of the actual configuration used by named at runtime.

- `-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).

- `-f`
  This option runs the server in the foreground (i.e., do not daemonize).
-F
This option turns on FIPS (US Federal Information Processing Standards) mode if the underlying cryptographic library supports running in FIPS mode.

-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 (comma-separated) memory context options. The possible flags are:

  • fill: fill blocks of memory with tag values when they are allocated or freed, to assist debugging of memory problems; this is the implicit default if named has been compiled with --enable-developer.
  • nofill: disable the behavior enabled by fill; this is the implicit default unless named has been compiled with --enable-developer.

-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 value
This option specifies the port(s) on which the server will listen for queries. If value is of the form <portnum> or dns=<portnum>, the server will listen for DNS queries on portnum; if not not specified, the default is port 53. If value is of the form tls=<portnum>, the server will listen for TLS queries on portnum; the default is 853. If value is of the form https=<portnum>, the server will listen for HTTPS queries on portnum; the default is 443. If value is of the form http=<portnum>, the server will listen for HTTP queries on portnum; the default is 80.

-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.

-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 has been removed. Attempts to use it now result in a warning.
-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, build options, supported cryptographics algorithms, and exits.

-X lock-file

This option has been removed and using it will cause a fatal error.

15.26.4 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.

15.26.5 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.

15.26.6 Files

/etc/named.conf

The default configuration file.

/run/named.pid

The default process-id file.
15.26.7 See Also


15.27 nsec3hash - generate NSEC3 hash

15.27.1 Synopsis

nsec3hash {salt} {algorithm} {iterations} {domain}
nsec3hash -r {algorithm} {flags} {iterations} {salt} {domain}

15.27.2 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.

15.27.3 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.
15.27.4 See Also


15.28 nslookup - query Internet name servers interactively

15.28.1 Synopsis

nslookup [-option] [name [-] [server]]

15.28.2 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.

15.28.3 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.

15.28.4 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|lsserver domain

These commands change the default server to domain: lsserver 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`. 

524 Chapter 15. Manual Pages
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.

norecurs
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.

15.28.5 Return Values

nslookup returns with an exit status of 1 if any query failed, and 0 otherwise.

15.28.6 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.

15.28.7 Files

/etc/resolv.conf

15.28.8 See Also

dig(1), host(1), named(8).
15.29 nsupdate - dynamic DNS update utility

15.29.1 Synopsis

```
```

15.29.2 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.

Transactions 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.

15.29.3 Options

-4

This option sets use of IPv4 only.

-6

This option sets use of IPv6 only.

-A tlscafile

This option specifies the file of the certificate authorities (CA) certificates (in PEM format) in order to verify the remote server TLS certificate when using DNS-over-TLS (DoT), to achieve Strict or Mutual TLS. When used, it will override the certificates from the global certificates store, which are otherwise used by default when -S is enabled. This option can not be used in conjuction with -C, and it implies -S.

-C

Overrides the default resolv.conf file. This is only intended for testing.
-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.

-Dr
   This option sets extra debug mode.

-E tiscertfile
   This option sets the certificate(s) file for authentication for the DNS-over-TLS (DoT) transport to the remote server. The certificate chain file is expected to be in PEM format. This option implies -S, and can only be used with -K.

-g
   This option enables standard GSS-TSIG mode.

-H tishostname
   This option makes nsupdate use the provided hostname during remote server TLS certificate verification. Otherwise, the DNS server name is used. This option implies -S.

-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.

-K tlskeyfile
   This option sets the key file for authenticated encryption for the DNS-over-TLS (DoT) transport with the remote server. The private key file is expected to be in PEM format. This option implies -S, and can only be used with -E.

-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 /run/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.

-o
   This option is deprecated. Previously, it enabled a non-standards-compliant variant of GSS-TSIG that was used by Windows 2000. Since that OS is now long past its end of life, this option is now treated as a synonym for -g.

-O
   This option enables Opportunistic TLS. When used, the remote peer’s TLS certificate will not be verified. This option should be used for debugging purposes only, and it is not recommended to use it in production. This option can not be used in conjunction with -A, and it implies -S.

-p port
   This option sets the port to use for connections to a name server. The default is 53.
This option prints the list of private BIND-specific resource record types whose format is understood by `nsupdate`. See also the `-T` option.

This option sets the number of UDP retries. The default is 3. If zero, only one update request is made.

This option indicates whether to use DNS-over-TLS (DoT) when querying name servers specified by `servername port` syntax in the input file, and the primary server discovered through a SOA request. When the `-K` and `-E` options are used, then the specified TLS client certificate and private key pair are used for authentication (Mutual TLS). This option implies `-v`.

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 for TCP mode. For UDP mode, the option `-u` takes precedence over this option, unless the option `-u` is set to zero, in which case the interval is computed from the `-t` timeout interval and the number of UDP retries. For UDP mode, the timeout can not be disabled, and will be rounded up to 1 second in case if both `-t` and `-u` are set to zero.

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>`.

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.

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.

This option prints the version number and exits.

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.
**15.29.4 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.

*Note:* This command has no effect when GSS-TSIG is in use.

**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 is deprecated and will be removed in a future release. Previously, it caused `nsupdate` to use the Windows 2000 version of GSS-TSIG to sign updates. It is now treated as a synonym for gsstsig.

**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 [boolean]**

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.

**check-svbc [boolean]**

This command turns on or off check-svcb processing on records to be added. Check-svcb has no effect on prerequisites or records to be deleted. By default check-svcb processing is on. If check-svcb processing fails, the record is not added to the UPDATE message.

**lease time [keytime]**

Set the EDNS Update Lease (UL) option to value to `time` and optionally also set the key lease time to `keytime` in seconds. If `time` is `none` the lease times are cleared.

**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.
15.29.5 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.)

15.29.6 Files

/etc/resolv.conf
- Used to identify the default name server

/run/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`

15.29.7 See Also

15.29.8 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.

15.30 rndc-confgen - rndc key generation tool

15.30.1 Synopsis

```
```

15.30.2 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.

15.30.3 Options

- `a`
  This option sets automatic `rndc` configuration, which creates a file `/etc/rndc.key` 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.
-q
This option prevents printing the written path in automatic configuration mode.

-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.

15.30.4 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
```

15.30.5 See Also


15.31 rndc.conf - rndc configuration file

15.31.1 Synopsis

```
rndc.conf
```

15.31.2 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 an 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.

### 15.31.3 Example

```plaintext
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+1z";
};

key testkey {
    algorithm hmac-sha256;
    secret "R3HI8P6RKw92wXwN3VZKuQ==";
};
```

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
```

### 15.31.4 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.

### 15.31.5 See Also

`rndc(8), rndc-confgen(8), mmencode(1), BIND 9 Administrator Reference Manual`

### 15.32 rndc - name server control utility

#### 15.32.1 Synopsis

```
```

#### 15.32.2 Description

`rndc` controls the operation of a name server. 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 server_key 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.
15.32.3 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.).

-t timeout
   This option sets the idle timeout period for rndc to timeout seconds. The default is 60 seconds, and the maximum settable value is 86400 seconds (1 day). If set to 0, there is no timeout.

-V
   This option enables verbose logging.

-y server_key
   This option indicates use of the key server_key from the configuration file. For control message validation to succeed, server_key must be known by named with the same algorithm and secret string. If no server_key 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.
15.32.4 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:

```bash
rndc addzone example.com '{ type primary; 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 | withdrawn) zone [class [view]]

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` informs `named` that the DS for a specified zone’s key-signing key has been confirmed to be published in, or withdrawn from, the parent zone. This is required in order to complete a KSK rollover. The `-key id` and `-alg algorithm` arguments can be used to specify a particular KSK, if necessary; if there is only one key acting as a KSK for the zone, these arguments can be omitted. The time of publication or withdrawal for the DS is set to the current time by default, but can be overridden to a specific time with the argument `-when time`, where time is expressed in YYYYMMDDHHMMSS notation.

`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.)

**fetchlimit** [view]

This command dumps a list of servers that are currently being rate-limited as a result of fetches-per-server settings, and a list of domain names that are currently being rate-limited as a result of fetches-per-zone settings.

**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, 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 \texttt{-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 \texttt{-class}. The default is class \texttt{IN}, which is the only class for which DNSSEC is currently supported.

All of these options can be shortened, i.e., to \texttt{-l}, \texttt{-r}, \texttt{-d}, \texttt{-f}, and \texttt{-c}.

Unrecognized options are treated as errors. To refer to a domain or view name that begins with a hyphen, use a double-hyphen (\texttt{--}) on the command line to indicate the end of options.

\textbf{querylog} \([\text{(on | off)}]\)

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 \texttt{queries category} to a channel in the \texttt{logging} section of \texttt{named.conf}, or by specifying \texttt{querylog yes;} in the \texttt{options} section of \texttt{named.conf}.

\textbf{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 \texttt{rndc reload} when there is a large number of zones, because it avoids the need to examine the modification times of the zone files.

\textbf{recursing}

This command dumps the list of queries \texttt{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 \texttt{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).

\textbf{refresh} \texttt{zone} [\texttt{class [view]}]

This command schedules zone maintenance for the given zone.

\textbf{reload}

This command reloads the configuration file and zones.

\textbf{zone} [\texttt{class [view]}]

If a zone is specified, this command reloads only the given zone. If no zone is specified, the reloading happens asynchronously.

\textbf{retransfer} \texttt{zone} [\texttt{class [view]}]

This command retransfers the given secondary zone from the primary server.

If the zone is configured to use \texttt{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.

\textbf{scan}

This command scans the list of available network interfaces for changes, without performing a full \texttt{rndc reconfig} or waiting for the \texttt{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 caching and serving of stale answers is currently enabled or disabled. 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, and also requires the zone to be configured to allow dynamic DNS. (See “Dynamic Update Policies” in the 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.

The only recommended configuration is `rndc signing -nsec3param 1 0 0 - zone`, i.e. no salt, no additional iterations, no opt-out.

**Warning:** Do not use extra iterations, salt, or opt-out unless all their implications are fully understood. A higher number of iterations causes interoperability problems and opens servers to CPU-exhausting DoS attacks.

```
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'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. If no zone is specified, the reloading happens asynchronously.

See also `rndc freeze`.

**trace** `[level]`
If no level is specified, this command increments the server's debugging level by one.

**level**
If specified, this command sets the server's debugging level to the provided value.
See also `rndc notrace`.

**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”.)

### 15.32.5 Limitations

There is currently no way to provide the shared secret for a `server_key` without using the configuration file. Several error messages could be clearer.

### 15.32.6 See Also


### 15.33 tsig-keygen - TSIG key generation tool

#### 15.33.1 Synopsis

`tsig-keygen [-a algorithm] [-h] [name]`

#### 15.33.2 Description

`tsig-keygen` is an utility that generates keys for use with TSIG (Transaction Signatures) as defined in RFC 2845. 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`.
15.33.3 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.

15.33.4 See Also

Symbols

+aaflag
  dig command line option, 448
  mdig command line option, 490
+aaonly
  dig command line option, 448
  mdig command line option, 490
+additional
  dig command line option, 448
  mdig command line option, 488
+adflag
  dig command line option, 448
  mdig command line option, 488
+all
  delv command line option, 444
  dig command line option, 448
  mdig command line option, 488
+answer
  dig command line option, 448
  mdig command line option, 488
+authority
  dig command line option, 448
  mdig command line option, 488
+badcookie
  dig command line option, 448
+besteffort
  dig command line option, 448
  mdig command line option, 488
+bufsize
  dig command line option, 448
  mdig command line option, 488
+burst
  mdig command line option, 488
+cd
  dig command line option, 448
+cdflag
  delv command line option, 443
  dig command line option, 448
  mdig command line option, 490
+cl
  mdig command line option, 489
+class
  delv command line option, 443
  dig command line option, 448
+cmd
  dig command line option, 448
+comments
  delv command line option, 444
  dig command line option, 448
  mdig command line option, 489
+continue
  mdig command line option, 489
+cookie
  dig command line option, 449
  mdig command line option, 490
+crypto
  delv command line option, 444
  dig command line option, 449
  mdig command line option, 489
+defname
  dig command line option, 449
+dns64prefix
  dig command line option, 449
+dnssec
  delv command line option, 444
  dig command line option, 449
  mdig command line option, 490
+do
  dig command line option, 449
+domain
  dig command line option, 449
+edns
  dig command line option, 449
  mdig command line option, 490
+ednsflags
  dig command line option, 449
  mdig command line option, 490
+ednsnegotiation
  dig command line option, 449
+ednsopt
  dig command line option, 449
  mdig command line option, 490
+expire
  dig command line option, 449

545
mdig command line option, 491
+fail
dig command line option, 449
+fuzztime
dig command line option, 449
+header-only
dig command line option, 450
+hint
delv command line option, 443
+http-plain
dig command line option, 450
+http-plain-get
dig command line option, 450
+http-plain-post
dig command line option, 450
+https
dig command line option, 450
+https-get
dig command line option, 450
+https-post
dig command line option, 450
+identify
dig command line option, 450
+idn
dig command line option, 450
+ignore
dig command line option, 450
+keepalive
dig command line option, 450
+keepopen
dig command line option, 450
+mtrace
delv command line option, 443
+multiline
delv command line option, 444
dig command line option, 450
+ndots
dig command line option, 450
+noaaflag
dig command line option, 448
+noadditional
dig command line option, 448
+noadflag
dig command line option, 448
+noall
delv command line option, 444
dig command line option, 448
+noanswer
dig command line option, 448
+noauthority
dig command line option, 448
+nobadcookie
dig command line option, 448
+nobesteffort
dig command line option, 448
+nocodeflag
delv command line option, 443
dig command line option, 448
+nocl
dig command line option, 448
+noclass
delv command line option, 444
dig command line option, 448
+nocmd
dig command line option, 448
+nocomments
delv command line option, 444
dig command line option, 448
+nocookie
dig command line option, 449
+nocrypto
delv command line option, 444
dig command line option, 448
+nodns64prefix
dig command line option, 449
+nodnssec
delv command line option, 444
dig command line option, 449
+nodefname
dig command line option, 449
+nodnsflags
dig command line option, 449
+nodefname
+nodnsnegotiation
dig command line option, 449
+noednsopt
    dig command line option, 449
    mdig command line option, 490
+noexpire
    dig command line option, 449
    mdig command line option, 491
+nofail
    dig command line option, 449
    mdig command line option, 491
+nofuzztime
    dig command line option, 449
+noheader-only
    dig command line option, 450
+nohint
    delv command line option, 443
+nohttp-plain
    dig command line option, 450
    mdig command line option, 450
+nohttp-plain-get
    dig command line option, 450
+nohttp-plain-post
    dig command line option, 450
+nohttps
    dig command line option, 450
    mdig command line option, 450
+nohttps-get
    dig command line option, 450
+nohttps-post
    dig command line option, 450
+noidentifier
    dig command line option, 450
+noindent
    dig command line option, 450
+noignore
    dig command line option, 450
+nokeepalive
    dig command line option, 450
+nokeepopen
    dig command line option, 450
+nomtrace
    delv command line option, 443
+nomultiline
    delv command line option, 444
    dig command line option, 450
    mdig command line option, 489
+nons
    delv command line option, 443
+nonsid
    dig command line option, 451
    mdig command line option, 491
+nonssearch
    dig command line option, 451
+nonesoa
    dig command line option, 451
+nopcode
    dig command line option, 451
+noproxy
<table>
<thead>
<tr>
<th>Option</th>
<th>Command Line Option</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+notls-certfile</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+notls-hostname</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+notls-keyfile</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+notrace</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+notrust</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+nottl</td>
<td>delv</td>
<td>443</td>
</tr>
<tr>
<td>+nottlid</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+nottlunits</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+unknownformat</td>
<td>delv</td>
<td>445</td>
</tr>
<tr>
<td>+ns</td>
<td>delv</td>
<td>443</td>
</tr>
<tr>
<td>+nsid</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+nssearch</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+onesoa</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+opcode</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+padding</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+proxy</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+proxy-plain</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+qid</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+qmin</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+qr</td>
<td>dig</td>
<td>451</td>
</tr>
<tr>
<td>+question</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+raflag</td>
<td>dig</td>
<td>452</td>
</tr>
<tr>
<td>+rdflag</td>
<td>dig</td>
<td>452</td>
</tr>
<tr>
<td>+retrace</td>
<td>delv</td>
<td>443</td>
</tr>
<tr>
<td>+retry</td>
<td>dig</td>
<td>452</td>
</tr>
<tr>
<td>+root</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+rrcomments</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+search</td>
<td>dig</td>
<td>452</td>
</tr>
<tr>
<td>+short</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+split</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+stats</td>
<td>delv</td>
<td>444</td>
</tr>
<tr>
<td>+strace</td>
<td>delv</td>
<td>443</td>
</tr>
<tr>
<td>+subnet</td>
<td>dig</td>
<td>452</td>
</tr>
<tr>
<td>+tcflag</td>
<td>dig</td>
<td>453</td>
</tr>
<tr>
<td>+tcp</td>
<td>delv</td>
<td>445</td>
</tr>
<tr>
<td>+time</td>
<td>dig</td>
<td>453</td>
</tr>
</tbody>
</table>
+tls
dig command line option, 453
+tls-ca
dig command line option, 453
+tls-certfile
dig command line option, 453
+tls-hostname
dig command line option, 453
+tls-keyfile
dig command line option, 453
+trace
dig command line option, 453
+tries
dig command line option, 453
mdig command line option, 491
+trust
delv command line option, 444
+tts
delv command line option, 443
+ttl
delv command line option, 443
+ttllid
dig command line option, 453
mdig command line option, 489
+ttlns
dig command line option, 453
mdig command line option, 489
+udptimeout
mdig command line option, 491
+unknownformat
delv command line option, 445
dig command line option, 453
mdig command line option, 491
+vc
dig command line option, 454
mdig command line option, 489
+vtrace
delv command line option, 444
+yaml
delv command line option, 445
dig command line option, 454
mdig command line option, 491
+zflag
dig command line option, 454
mdig command line option, 491

$TTL, 33
-1
dnssec-dsfromkey command line option, 458
-2
dnssec-dsfromkey command line option, 458
-3
dnssec-keyfromlabel command line option, 462
dnssec-keygen command line option, 465
dnssec-signzone command line option, 480
-4
delv command line option, 442
dig command line option, 446
host command line option, 485
mdig command line option, 488
named command line option, 519
nsupdate command line option, 526
rndc command line option, 536
-6
delv command line option, 442
dig command line option, 446
host command line option, 485
mdig command line option, 488
named command line option, 519
nsupdate command line option, 526
rndc command line option, 536
-A
dnssec-dsfromkey command line option, 458
dnssec-keyfromlabel command line option, 464
dnssec-keygen command line option, 468
dnssec-settime command line option, 474
dnssec-signzone command line option, 480
host command line option, 485
nsupdate command line option, 526
rndc-confgen command line option, 532
-AA
dnssec-signzone command line option, 480
-A

dnssec-dsfromkey command line option, 459
dnssec-keyfromlabel command line option, 462
dnssec-keygen command line option, 466
dnssec-signzone command line option, 476
host command line option, 485
named command line option, 519
named-checkzone command line option, 494
named-compilezone command line option, 497
named-rrchecker command line option, 501
named-compilezone command line option, 497
nsupdate command line option, 527

-M dnssec-signzone command line option, 477
named command line option, 520
named-checkzone command line option, 495
named-compilezone command line option, 498

-N dnssec-signzone command line option, 478
host command line option, 485

-O dnssec-signzone command line option, 479
nsupdate command line option, 527

-P dnssec-importkey command line option, 461
dnssec-keyfromlabel command line option, 464
dnssec-keygen command line option, 468
dnssec-settime command line option, 474
dnssec-signzone command line option, 479
named-rrchecker command line option, 501
nsupdate command line option, 527

-Q dnssec-signzone command line option, 479

-R dnssec-keyfromlabel command line option, 464
dnssec-keygen command line option, 468
dnssec-revoke command line option, 472
dnssec-settime command line option, 474
dnssec-signzone command line option, 479
host command line option, 486

-S dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 467
dnssec-settime command line option, 467

-dnssec-signzone command line option, 474
named-checkzone command line option, 495
named-compilezone command line option, 498
nsupdate command line option, 528

-T dnssec-cds command line option, 456
dnssec-dsfromkey command line option, 459
dnssec-keygen command line option, 469
dnssec-signzone command line option, 467
dnssec-signzone command line option, 480
host command line option, 486
named-checkzone command line option, 495
named-compilezone command line option, 498
named-rrchecker command line option, 501
nsupdate command line option, 528

-U host command line option, 486
named command line option, 520

-V dnssec-cds command line option, 457
dnssec-dsfromkey command line option, 459
dnssec-importkey command line option, 461
dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 467
dnssec-ksr command line option, 470
dnssec-revoke command line option, 472
dnssec-settime command line option, 473
dnssec-signzone command line option, 478
dnssec-verify command line option, 482
host command line option, 486
named command line option, 521
nsupdate command line option, 528
rndc command line option, 536

-W host command line option, 486
named-checkzone command line option, 495
named-compilezone command line option, 497
-X
dnssec-signzone command line option, 477
named command line option, 521
-a
ddns-confgen command line option, 440
delv command line option, 441
dnssec-cds command line option, 456
dnssec-dsfromkey command line option, 458
dnssec-keyfromlabel command line option, 462
dnssec-keygen command line option, 465
dnssec-signzone command line option, 476
host command line option, 485
named-checkconf command line option, 492
rndc-confgen command line option, 532
tsig-keygen command line option, 544
-b
delv command line option, 442
dig command line option, 446
dnssec-keygen command line option, 466
mdig command line option, 488
rndc command line option, 536
rndc-confgen command line option, 532
-d
delv command line option, 442
dnssec-cds command line option, 456
dnssec-keygen command line option, 466
dnssec-settime command line option, 475
dnssec-signzone command line option, 476
host command line option, 485
named command line option, 519
named-checkzone command line option, 493
named-compilezone command line option, 496
nsupdate command line option, 526
-e
dnssec-ksr command line option, 470
dnssec-signzone command line option, 477
-f
dig command line option, 446
dnssec-cds command line option, 456
dnssec-dsfromkey command line option, 459
dnssec-importkey command line option, 460
dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 466
dnssec-revoke command line option, 472
dnssec-settime command line option, 473
dnssec-signzone command line option, 477
mdig command line option, 488
named command line option, 519
named-checkzone command line option, 494
named-compilezone command line option, 497
-g
dnssec-settime command line option, 475
dnssec-signzone command line option, 476
named command line option, 520
nsupdate command line option, 527
-h
ddns-confgen command line option, 440
Index 553

delv command line option, 442
dig command line option, 446
dnssec-dsfromkey command line option, 459
dnssec-importkey command line option, 460
dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 466
dnssec-krr command line option, 470
dnssec-revoke command line option, 472
dnssec-settime command line option, 473
dnssec-signzone command line option, 477
mdig command line option, 488
named-checkconf command line option, 492
named-checkzone command line option, 493
named-compilezone command line option, 496
ddns-confgen command line option, 440
dig command line option, 446
dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 467
dnssec-krr command line option, 470
dnssec-settime command line option, 475
dnssec-signzone command line option, 477
named-checkzone command line option, 494
named-compilezone command line option, 497
nsupdate command line option, 527
rndc command line option, 536
rndc-confgen command line option, 532

-d

delv command line option, 442
dnssec-cds command line option, 456
dnssec-keyfromlabel command line option, 464
dnssec-keygen command line option, 468
dnssec-krr command line option, 470

dnssec-settime command line option, 474
dnssec-signzone command line option, 478
named-checkconf command line option, 492
named-checkzone command line option, 494
named-compilezone command line option, 497
nsupdate command line option, 527

-m

dnssec-signzone command line option, 478
named-checkconf command line option, 492
named-checkzone command line option, 493

dnssec-keyfromlabel command line option, 462
dnssec-keygen command line option, 467
dnssec-krr command line option, 470
host command line option, 485
named-checkconf command line option, 492
named-checkzone command line option, 494
named-compilezone command line option, 497

-n

dnssec-signzone command line option, 478
named-checkconf command line option, 492
named-checkzone command line option, 493

dnssec-keyfromlabel command line option, 462
dnssec-keygen command line option, 467
dnssec-signzone command line option, 478
named command line option, 520
named-checkzone command line option, 495
named-compilezone command line option, 498
-o
dnssec-signzone command line option, 478
dnssec-verify command line option, 482
named-checkzone command line option, 495
named-compilezone command line option, 498
named-rrchecker command line option, 501
nsupdate command line option, 528
-p
delv command line option, 442
dig command line option, 447
dnssec-keyfromlabel command line option, 463
dnssec-keygen command line option, 467
dnssec-settime command line option, 475
dnstap-read command line option, 483
host command line option, 486
mdig command line option, 488
named command line option, 520
named-checkconf command line option, 492
named-rrchecker command line option, 501
nsupdate command line option, 528
rndc command line option, 536
rndc-confgen command line option, 532
-q
ddns-confgen command line option, 440
delv command line option, 442
dig command line option, 447
dnssec-keyfromlabel command line option, 467
dnssec-signzone command line option, 479
dnssec-verify command line option, 482
named-checkzone command line option, 493
named-compilezone command line option, 496
rndc command line option, 536
rndc-confgen command line option, 532
-r
dig command line option, 447
-dnssec-revoke command line option, 472
dnssec-settime command line option, 475
host command line option, 486
named-checkzone command line option, 495
named-compilezone command line option, 498
nsupdate command line option, 528
rndc command line option, 536
dnssec-cds command line option, 440
dnssec-dsfromkey command line option, 456
dnssec-keygen command line option, 459
dnssec-settime command line option, 475
dnssec-signzone command line option, 477
host command line option, 486
named command line option, 520
named-checkzone command line option, 495
named-compilezone command line option, 498
rndc command line option, 536
rndc-confgen command line option, 533
-r
delv command line option, 442
dig command line option, 447
dnssec-keygen command line option, 467
dnssec-signzone command line option, 478
host command line option, 486
mdig command line option, 488
named command line option, 520
named-checkconf command line option, 492
named-checkzone command line option, 495
named-compilezone command line option, 498
nsupdate command line option, 528
rndc command line option, 536
rndc-confgen command line option, 533
-u
dig command line option, 447
Index
delv
  manual page, 440
delv command line option
  +all, 444
  +cdflag, 443
  +class, 443
  +comments, 444
  +crypto, 444
  +dnssec, 444
  +hint, 443
  +mtrace, 443
  +multiline, 444
  +noall, 444
  +nocdflag, 443
  +noclass, 443
  +nocomments, 444
  +nocrypto, 444
  +nodnssec, 444
  +nohint, 443
  +nomtrace, 443
  +nomultiline, 444
  +nons, 443
  +noqmin, 443
  +noroot, 444
  +norrrcomments, 444
  +nortrace, 443
  +noqmin, 443
  +noqmin, 443
  +nomultiline, 444
  +nons, 443
  +noqmin, 443
  +norrrcomments, 444
  +nortrace, 443
  +noqmin, 443
  +nomultiline, 444
  +nons, 443
  +noqmin, 443
  +noroot, 444
  +norrrcomments, 444
  +nortrace, 443
  +noqmin, 443
  +nomultiline, 444
  +nons, 443
  +noqmin, 443
  +norrrcomments, 444
  +nortrace, 443
  +noqmin, 443
  +nomultiline, 444
  +nons, 443
  +noqmin, 443
  +noroot, 444
  +norecm, 444
  +nnounknownformat, 445
  +novtrace, 444
  +nnounknownformat, 445
  +novtrace, 444
  +ns, 443
  +qmin, 443
  +root, 444
  +rtrace, 443
  +short, 444
  +split, 444
  +strace, 443
  +tcp, 445
  +trust, 444
  +ttl, 443
  +unknownformat, 445
  +vtrace, 444
  +yaml, 445
  -4, 442
  -6, 442
  -a, 441
  -b, 442
  -c, 442
  -d, 442
+idn, 450
+ignore, 450
+keepalive, 450
+keepopen, 450
+multiline, 450
+ndots, 450
+noaaflag, 448
+noa第一名, 448
+noadditional, 448
+noadflag, 448
+noanswer, 448
+noauthority, 448
+nobadcookie, 448
+nobesteffort, 448
+nocdflag, 448
+noclass, 448
+nocmd, 448
+nocomments, 448
+nocookie, 449
+nocrypto, 449
+nodefname, 449
+nodns64prefix, 449
+nodnssec, 449
+nodo, 449
+noedns, 449
+noednsflags, 449
+noednsnegotiation, 449
+noednsopt, 449
+noexpire, 449
+nofail, 449
+nofuzztime, 449
+nohost-only, 450
+nohttp-plain, 450
+nohttp-plain-get, 450
+nohttp-plain-post, 450
+noltimes, 450
+nosplit, 450
+nosurround, 450
+noidentify, 450
+noidn, 450
+noidnignore, 450
+nookipalive, 450
+nookipopen, 450
+nomultiline, 450
+nonsid, 451
+nosregex, 451
+nossearch, 451
+nonesoa, 451
+nopcode, 451
+noproxy, 451
+noproxy-plain, 451
+nopr, 451
+noplication, 452
+nosqlflag, 452
+norecurse, 452
+norrcomments, 452
+nosearch, 452
+noshort, 452
+noshowbadcookie, 452
+noshowsearch, 452
+nosubnet, 452
+notcflag, 453
+notcp, 453
+notls, 453
+notls-ca, 453
+notls-certfile, 453
+notls-hostname, 453
+notls-keyfile, 453
+notrace, 453
+nottlid, 453
+nottlunits, 453
+nounknownformat, 453
+nvoc, 454
+noyaml, 454
+noflag, 454
+nssid, 451
+nsssearch, 451
+nonesoa, 451
+opcode, 451
+padding, 451
+proxy, 451
+proxy-plain, 451
+qid, 451
+qr, 451
+question, 452
+raflag, 452
+rflags, 452
+recurs, 452
+retry, 452
+rrcomments, 452
+search, 452
+short, 452
+showbadcookie, 452
+showsearch, 452
+split, 452
+stats, 452
+subnet, 452
+tcflag, 453
+tcp, 453
+timeout, 453
+tls, 453
+tls-ca, 453
+tls-certfile, 453
+tls-hostname, 453
+tls-keyfile, 453
+trace, 453
+tries, 453
+ttlid, 453
+ttlunits, 453
+unknownformat, 453
+vc, 454
+yaml, 454
+zflag, 454
-4, 446
-6, 446
-b, 446
-c, 446
-f, 446
-h, 446
-k, 446
-m, 446
-p, 447
-q, 447
-r, 447
-t, 447
-u, 447
-v, 447
-x, 447
-y, 447
name, 446
server, 446
type, 446
dnssec
    rndc command line option, 537
    manual page, 455
dnssec-cds
    command line option
    -D, 456
    -T, 456
    -V, 457
    -a, 456
    -c, 456
    -d, 456
    -f, 456
    -i, 456
    -s, 456
    -u, 456
    -v, 457
dnssec-dsfromkey
    manual page, 458
    command line option
    -1, 458
    -2, 458
    -A, 458
    -C, 459
    -K, 459
    -T, 459
    -V, 459
    -a, 458
    -c, 458
    -D, 456
    -K, 456
    -L, 460
    -P, 461
    -V, 461
    -f, 460
    -h, 460
    -v, 460
dnssec-importkey
    command line option
    -D, 461
    -K, 460
    -L, 460
    -P, 461
    -V, 461
    -f, 460
    -h, 460
    -v, 460
dnssec-importkey--D command line option
    sync, 461
dnssec-importkey--P command line option
    sync, 461
dnssec-keyfromlabel
    command line option
    -3, 462
    -A, 464
    -C, 462
    -D, 464
    -E, 462
    -I, 464
    -K, 463
    -L, 463
dnssec-keyfromlabel--D command line option
    sync, 464
dnssec-keyfromlabel--P command line option
    sync, 464
dnssec-keygen
manual page, 465
dnssec-keygen command line option
  -3, 465
  -A, 468
  -C, 466
  -D, 468
  -E, 466
  -F, 466
  -G, 466
  -I, 468
  -K, 466
  -L, 467
  -P, 468
  -R, 468
  -S, 467
  -T, 467
  -V, 467
  -a, 465
  -b, 466
  -c, 466
  -d, 466
  -f, 466
  -h, 466
  -i, 468
  -k, 467
  -l, 467
  -n, 467
  -p, 467
  -q, 467
  -s, 467
  -t, 467
  -v, 468
dnssec-keygen--D command line option
sync, 468
dnssec-keygen--P command line option
sync, 468
dnssec-keygen command line option
  -E, 472
  -K, 472
  -R, 472
  -V, 472
  -f, 472
  -h, 472
  -r, 472
  -v, 472
dnssec-revoke command line option

dnssec-revoke command line option
  -E, 472
  -K, 472
  -R, 472
  -V, 472
  -f, 472
  -h, 472
  -r, 472
  -v, 472
dnssec-settime--D command line option
ds, 474
dnssec-settime--P command line option
ds, 474
dnssec-settime command line option
  -A, 474
  -D, 474
  -E, 474
  -I, 474
  -K, 473
  -L, 473
  -P, 474
  -R, 474
  -S, 474
  -V, 473
  -u, 475
dnssec-settime--P command line option
ds, 474
sync, 474
dnssec-settime command line option
  -A, 474
  -D, 474
  -E, 474
  -I, 474
  -K, 473
  -L, 473
  -P, 474
  -R, 474
  -S, 474
  -V, 473
  -u, 475
dnssec-signzone command line option
request, 471
sign, 471
dnssec-signzone command line option
  -3, 480
  -A, 480
  -AA, 480
  -C, 476
  -D, 476
  -E, 476
  -F, 476
  -G, 477
  -H, 480
  -I, 478
  -J, 477
halt
  rndc command line option, 538
host
  manual page, 484
host command line option
    -4, 485
    -6, 485
    -A, 485
    -C, 485
    -N, 485
    -R, 486
    -T, 486
    -U, 486
    -V, 486
    -W, 486
    -a, 485
    -c, 485
    -d, 485
    -1, 485
    -m, 486
    -p, 486
    -r, 486
    -s, 486
    -t, 486
    -v, 486
    -w, 486
HS, 32

IN, 32
Indeterminate, 429
Insecure, 429
integer, 91
ip_address, 91
ipv4_address, 91
ipv6_address, 91
Iterations, 399
iterations
  nsec3hash command line option, 522
K
  key
dnssec-signzone command line option, 481
  keygen
dnssec-ksr command line option, 471
L
  level
  rndc-trace command line option, 542
loadkeys
  rndc command line option, 538
M
  managed-keys
  rndc command line option, 538
manual page
  arpaname, 439
ddns-config, 439
delv, 440
dig, 445
dnssec-cds, 455
dnssec-dsfromkey, 458
dnssec-importkey, 460
dnssec-keyfromlabel, 461
dnssec-keygen, 465
dnssec-ksr, 469
dnssec-revoke, 471
dnssec-settime, 472
dnssec-signzone, 476
dnssec-verify, 481
dnstap-read, 483
filter-aaaa, 483
host, 484
mdig, 487
named, 518
named.conf, 501
named-checkconf, 491
named-checkzone, 493
named-compilezone, 496
named-journalprint, 499
named-nzd2nzf, 500
named-rrchecker, 500
nsec3hash, 522
nslookup, 523
nsupdate, 525
rndc, 535
rndc.conf, 533
rndc-config, 532
  tsig-keygen, 543
mdig
  manual page, 487
  mdig command line option
    +aaflag, 490
    +aaonly, 490
    +additional, 488
    +adflag, 490
    +all, 488
    +answer, 488
    +authority, 488
    +besteffort, 488
    +bufsize, 490
    +burst, 488
    +cdflag, 490
    +cl, 489
    +comments, 489
    +continue, 489
    +cookie, 490
    +crypto, 489
    +dnssec, 490
    +edns, 490
    +ednsflags, 490
    +ednssopt, 490
    +expire, 491
    +multiline, 489
    +noaaflag, 490
    +noaaonly, 490
    +noadditional, 488
    +nodash, 490
    +noall, 488
    +noanswer, 488
    +noauthority, 488
    +nobreastime, 488
    +nodedflag, 490
    +nocl, 489
    +nocontinue, 489
    +nocookie, 490
    +nocrypto, 489
    +nodnssec, 490
    +noedns, 490
    +noednsflags, 490
    +noednssopt, 490
    +noexpire, 491
    +nomultiline, 489
    +nocomments, 489
    +nocomments, 489
    +nosubnet, 491
    +notcp, 489
    +notlid, 489
+nottlunits, 489
+nounknownformat, 491
+nocv, 489
+noyaml, 491
+nozflag, 491
+nsid, 491
+question, 489
+recurse, 491
+retry, 491
+rnccomments, 489
+short, 489
+split, 489
+subnet, 491
+tcp, 489
+timeout, 491
+tries, 491
+ttlid, 489
+tttlnits, 489
+udptimeout, 491
+unknownformat, 491
+vc, 489
+yaml, 491
+zflag, 491
-4, 488
-6, 488
-b, 488
-c, 490
-f, 488
-h, 488
-m, 488
-p, 488
-t, 490
-v, 488
-x, 490

modzone
  rndc command line option, 539

N
name
delv command line option, 441
dig command line option, 446
named
  manual page, 518
named command line option
-4, 519
-6, 519
-c, 519
-d, 519
-f, 519
-g, 520
-m, 520
-n, 520
-p, 520
-s, 520
-t, 520
-u, 520
-v, 521
-nottlunits, 489
-nounknownformat, 491
-nocv, 489
-noyaml, 491
-nozflag, 491
-nsid, 491
-question, 489
-recurse, 491
-retry, 491
-rrcomments, 489
-short, 489
-split, 489
-subnet, 491
-tcp, 489
-timeout, 491
-tries, 491
-ttlid, 489
-tttlnits, 489
-udptimeout, 491
-unknownformat, 491
-vc, 489
-yaml, 491
-zflag, 491
-4, 521
-6, 520
-b, 488
-c, 490
-f, 488
-h, 488
-m, 488
-p, 488
-t, 490
-v, 488
-x, 490

named.conf
  manual page, 501
named-checkconf
  manual page, 491
named-checkconf command line option
-a, 492
-c, 492
-h, 492
-i, 492
-j, 492
-l, 492
-p, 492
-t, 492
-v, 492
-x, 492
-z, 493
filename, 493
named-checkzone
  manual page, 493
named-checkzone command line option
-c, 494
-d, 495
-F, 494
-J, 494
-L, 494
-M, 495
-S, 495
-T, 495
-W, 495
-h, 493
-N, 493
-o, 495
-q, 493
-r, 495

Index 563
-s, 495
-t, 495
-v, 493
-w, 495
filename, 495
zonename, 495
named-compilezone
manual page, 496
named-compilezone command line option
-C, 497
-D, 498
-F, 497
-J, 496
-L, 497
-M, 498
-S, 498
-T, 498
-W, 498
-c, 497
-d, 496
-f, 497
-h, 496
-i, 497
-j, 496
-k, 497
-l, 497
-n, 498
-o, 498
-q, 496
-r, 498
-s, 498
-t, 498
-v, 496
-w, 498
filename, 498
zonename, 498
named-journalprint
manual page, 499
named-nzd2nzf
manual page, 500
named-nzd2nzf command line option
filename, 500
named-rrchecker
manual page, 500
named-rrchecker command line option
-C, 501
-P, 501
-T, 501
-h, 501
-o, 501
-p, 501
-u, 501
netprefix, 92
notify
rndc command line option, 539
notrace
rndc command line option, 539
NSEC3 Hashing Algorithm, 399
nsec3hash
manual page, 522
nsec3hash command line option
algorithm, 522
domain, 522
flags, 522
iterations, 522
salt, 522
nslookup
manual page, 523
nsupdate
manual page, 525
nsupdate command line option
-4, 526
-6, 526
-A, 526
-C, 526
-d, 527
-E, 527
-H, 527
-K, 527
-L, 527
-O, 527
-P, 527
-S, 528
-T, 528
-V, 528
-d, 526
-g, 527
-l, 527
-k, 527
-1, 527
-n, 527
-ta
nta
rndc command line option, 539
O
Opt-out, 399
owner name, 31
P
percentage, 92
port, 92
portrange, 92

Q

querylog
  rndc command line option, 540

R

RDATA, 31
reconfig
  rndc command line option, 540
recurring
  rndc command line option, 540
refresh
  rndc command line option, 540
reload
  rndc command line option, 540
remote-servers, 92
request
dnssec-ksr command line option, 471
retransfer
  rndc command line option, 540
RFC
  RFC 1033, 519, 522
  RFC 1034, 7, 31, 32, 104, 146, 154, 431, 433, 501, 519, 522, 531
  RFC 1035, 7, 34, 35, 431, 433, 455, 491, 496, 499, 501, 519, 522
  RFC 1101, 437
  RFC 1123, 135, 145, 437
  RFC 1183, 433
  RFC 1535, 437
  RFC 1536, 437
  RFC 1706, 433
  RFC 1712, 433
  RFC 1796, 433
  RFC 1876, 433
  RFC 1912, 437
  RFC 1918, 24, 58, 186, 194, 241
  RFC 1982, 433, 478
  RFC 1995, 58, 73, 433
  RFC 1996, 18, 58, 73, 74, 433
  RFC 2104, 531
  RFC 2136, 57, 433, 526, 531
  RFC 2163, 433
  RFC 2181, 433
  RFC 2193, 24
  RFC 2219, 437
  RFC 2230, 434
  RFC 2308, 33, 35, 434
  RFC 2317, 36, 437
  RFC 2535, 86, 463, 465, 467, 526, 531
  RFC 2539, 434, 469
  RFC 2606, 437
  RFC 2671, 204, 262, 305
  RFC 2782, 434
  RFC 2845, 84, 469, 526, 531, 543
  RFC 2874, 437
  RFC 2930, 434
  RFC 2931, 86, 434, 526, 531
  RFC 3007, 57, 434, 531
  RFC 3056, 252
  RFC 3110, 434
  RFC 3123, 68, 434
  RFC 3225, 434
  RFC 3226, 434
  RFC 3339, 91
  RFC 3363, 61, 62, 434, 437
  RFC 3403, 434
  RFC 3492, 434
  RFC 3493, 434
  RFC 3496, 434
  RFC 3542, 13
  RFC 3548, 41
  RFC 3596, 434
  RFC 3597, 434, 445, 447, 454, 491
  RFC 3645, 434, 526
  RFC 3658, 460
  RFC 3833, 437
  RFC 3901, 437
  RFC 3986, 213
  RFC 4025, 434
  RFC 4033, 434, 481, 483
  RFC 4034, 434, 445, 465, 469
  RFC 4035, 429, 434, 445
  RFC 4074, 437
  RFC 4193, 186
  RFC 4255, 434
  RFC 4343, 434
  RFC 4398, 434
  RFC 4431, 437, 445
  RFC 4470, 399, 435
  RFC 4509, 435, 460
  RFC 4592, 71, 435, 495, 498
  RFC 4635, 435
  RFC 4701, 435
  RFC 4892, 437
  RFC 4955, 435
  RFC 5001, 435
  RFC 5011, 50, 93, 215, 386, 435, 441, 461, 463, 472, 476, 538
  RFC 5074, 445
  RFC 5077, 211
  RFC 5155, 421, 435, 445, 523
  RFC 5205, 435
  RFC 5452, 435
  RFC 5625, 437
  RFC 5702, 435
  RFC 5737, 24, 186
tcp-timeouts, 542
thaw, 542
trace, 542
validation, 543
zonestatus, 543
rndc.conf
manual page, 533
rndc-confgen
manual page, 532
rndc-confgen command line option
-A, 532
-a, 532
-b, 532
-c, 532
-h, 532
-k, 532
-p, 532
-q, 532
-s, 533
-t, 533
-u, 533
rndc-reload command line option
zone, 540
rndc-trace command line option
level, 542
RR TTLs, 33
RR type, 31

S
Salt, 399
salt
nsec3hash command line option, 522
scan
rndc command line option, 540
secroots
rndc command line option, 540
Secure, 429
serve-stale
rndc command line option, 541
server
delv command line option, 441
dig command line option, 446
server_key, 92
showzone
rndc command line option, 541
sign
dnssec-ksr command line option, 471
rndc command line option, 541
signing
rndc command line option, 541
size, 92
sizeval, 92
SOA minimum, 33
Statement, 87
stats
rndc command line option, 542
status
rndc command line option, 542
stop
rndc command line option, 542
sync
dnssec-importkey--D command line option, 461
dnssec-importkey--P command line option, 461
dnssec-keyfromlabel--D command line option, 464
dnssec-keyfromlabel--P command line option, 464
dnssec-keygen--D command line option, 468
dnssec-keygen--P command line option, 468
dnssec-settime--D command line option, 474
dnssec-settime--P command line option, 474
rndc command line option, 542

T
tcp-timeouts
rndc command line option, 542
thaw
rndc command line option, 542
tls_id, 92
trace
rndc command line option, 542
tsig-keygen
manual page, 543
tsig-keygen command line option
-a, 544
-h, 544
TTL, 31
type
delv command line option, 441
dig command line option, 446

V
validation
rndc command line option, 543

Z
zone
rndc-reload command line option, 540
zonefile
dnssec-signzone command line option, 481
dnssec-verify command line option, 482
zonename
   named-checkzone command line option, 495
   named-compilezone command line option, 498
zonestatus
   rndc command line option, 543