Knot DNS Documentation

Release 3.5.0

2025-09-18

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 What is Knot DNS

Knot DNS is a high-performance open-source DNS server. It implements only the authoritative domain name service. Knot DNS can reliably serve TLD domains as well as any other zones.

Knot DNS benefits from its multi-threaded and mostly lock-free implementation which allows it to scale well on SMP systems and operate non-stop even when adding or removing zones.

The server itself is accompanied by several utilities for general DNS operations or for maintaining the server.

For more info and downloads see www.knot-dns.cz.

1.2 Knot DNS features

DNS features:

- Primary and secondary server operation
- Internet (IN) and limited Chaos (CH) classes
- DNS extension (EDNS0, EDE, EXPIRE, PADDING, ZONEVERSION)
- UDP, TCP, TLS 1.3, and QUIC protocols
- Zone catalog generation and interpretation
- Minimal responses
- Dynamic zone updates
- DNSSEC with NSEC and NSEC3
- ZONEMD generation and validation
- Transaction signature using TSIG
- Full and incremental zone transfers (AXFR, IXFR)
- Name server identification using NSID or Chaos TXT records
- Resource record types A, NS, CNAME, SOA, PTR, HINFO, MINFO, MX, TXT, RP, AFSDB, RT, KEY, AAAA, LOC, SRV, NAPTR, KX, CERT, DNAME, APL, DS, SSHFP, IPSECKEY, RRSIG, NSEC, DNSKEY, DHCID, NSEC3, NSEC3PARAM, TLSA, SMIMEA, CDS, CDNSKEY, OPENPGPKEY, CSYNC, ZONEMD, SVCB, HTTPS, SPF, NID, L32, L64, LP, EUI48, EUI64, URI, CAA, WALLET, and Unknown

Server features:

- IPv4 and IPv6 support
- Semantic zone checks
- Server control interface
- Zone journal storage

- Persistent zone event timers
- YAML-based or database-based configuration
- Query processing modules with dynamic loading
- On-the-fly zone management and server reconfiguration
- Multithreaded DNSSEC zone signing and zone validation
- Automatic DNSSEC key management
- Zone data backup and restore
- Database zone backend
- Offline KSK operation
- PKCS #11 interface

Remarkable module extensions:

- Response rate limiting
- Forward and reverse records synthesis
- DNS request traffic statistics
- Efficient DNS traffic logging interface
- Dnstap traffic logging
- Online DNSSEC signing
- GeoIP response tailoring supporting ECS and DNSSEC

Remarkable supported networking features:

- TCP Fast Open (client and server)
- Opportunistic, strict, and mutual authentication profiles over TLS 1.3 or QUIC
- High-performance UDP, TCP, and QUIC through AF_XDP processing (on Linux 4.18+)
- SO_REUSEPORT (on Linux) or SO_REUSEPORT_LB (on FreeBSD 12.0+) on UDP and by choice on TCP
- Binding to non-local addresses (IP_FREEBIND on Linux, IP_BINDANY/IPV6_BINDANY on FreeBSD)
- Ignoring PMTU information for IPv4/UDP via IP_PMTUDISC_OMIT

1.3 License

Knot DNS is licensed under the GNU General Public License v2.0 or later. The full text of the license is available in the `COPYING` file distributed with the source code.

CHAPTER

TWO

REQUIREMENTS

2.1 Hardware

Knot DNS requirements are not very demanding for typical installations, and a commodity server or a virtual solution will be sufficient in most cases.

However, please note that there are some scenarios that will require administrator's attention and some testing of exact requirements before deploying Knot DNS to a production environment. These cases include deployment for a large number of zones (DNS hosting), large number of records in one or more zones (TLD), or large number of requests.

2.1.1 CPU requirements

The server scales with processing power and also with the number of available cores/CPUs. Enabling Hyper-threading is convenient if supported.

There is no lower bound on the CPU requirements, but it should support memory barriers and atomic instructions (i586 and newer).

2.1.2 Network card

The best results have been achieved with multi-queue network cards. The number of multi-queues should equal the total number of CPU cores (with Hyper-threading enabled).

2.1.3 Memory requirements

The server implementation focuses on performance and thus can be quite memory demanding. The rough estimate for memory requirements is 3 times the size of the zone in the plain-text format. Again this is only an estimate and you are advised to do your own measurements before deploying Knot DNS to production.

Note

To ensure uninterrupted serving of the zone, Knot DNS employs the Read-Copy-Update mechanism instead of locking and thus requires twice the amount of memory for the duration of incoming transfers.

2.2 Operating system

Knot DNS itself is written in a portable way and can be compiled and run on most UNIX-like systems, such as Linux, *BSD, and macOS.

2.3 Required libraries

Knot DNS requires a few libraries to be available:

- gnutls >= 3.6.10

- libedit
- liburcu
- lmdb >= 0.9.15

Note

The LMDB library is included with Knot DNS source code. However, linking with the system library is preferred.

2.4 Optional libraries

International Domain Names support (IDNA2008) in *kdig*:

- libidn2

Systemd's startup notification mechanism and journald logging:

- libsystemd

Dnstap support in *kdig* or module *dnstap*:

- fstrm (and protobuf-c if building from source code)

Linux `capabilities(7)` support, which allows the server to be started as a non-root user/group, binding to privileged ports (53), and giving up all its capabilities, resulting in a completely unprivileged process:

- libcap-ng >= 0.6.4

MaxMind database for **geodb** support in module *geoip*:

- libmaxminddb0

DNS-over-HTTPS (DoH) support in *kdig*:

- libnghttp2

The *XDP functionality* and *kxdpgun* tool. These are only supported on Linux operating systems. See the chapter *Mode XDP* for software and hardware recommendations.

- libbpf
- libxdp (if libbpf >= 1.0)
- libmnl (for kxdpgun)

DNS-over-QUIC (DoQ) support in *knotd*, *kxdpgun*, and *kdig*:

- libngtcp2 >= 0.17.0 (or embedded one via *--enable-quic*)
- gnutls >= 3.7.3
- *Mode XDP* (for knotd and kxdpgun)

Database zone backend support in *knotd*:

- libhiredis (for TLS support >= 1.1.0)

CHAPTER

THREE

INSTALLATION

3.1 Installation from a package

Knot DNS may already be included in your operating system distribution and therefore can be installed from packages (Linux), ports (BSD), or via Homebrew (macOS). This is always preferred unless you want to test the latest features, contribute to Knot development, or you know what you are doing.

See the project download page for the latest information.

3.2 Installation from source code

3.2.1 Required build environment

The build process relies on these standard tools:

- make
- libtool
- pkg-config
- autoconf >= 2.65
- python-sphinx (optional, for documentation building)

A GCC or LLVM Clang compiler with C11 support.

3.2.2 Getting the source code

You can find the source code for the latest release on www.knot-dns.cz. Alternatively, you can fetch the whole project from the git repository https://gitlab.nic.cz/knot/knot-dns.git.

After obtaining the source code, compilation and installation is quite a straightforward process using autotools.

3.2.3 Configuring and generating Makefiles

If compiling from git source, you need to bootstrap the `./configure` file first:

```
$ autoreconf -i -f
```

In most cases, you can just run configure without any options:

```
$ ./configure
```

For all available configure options run:

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

3.2.4 Compilation

After running `./configure` you can compile Knot DNS by running `make` command, which will produce binaries and other related files:

```
$ make
```

Note

The compilation with enabled optimizations may take a long time. In such a case the `--disable-fastparser` configure option can help.

3.2.5 Installation

When you have finished building Knot DNS, it's time to install the binaries and configuration files into the operation system hierarchy. You can do so by executing:

```
$ make install
```

When installing as a non-root user, you might have to gain elevated privileges by switching to root user, e.g. `sudo make install` or `su -c 'make install'`.

CHAPTER

FOUR

CONFIGURATION

4.1 Simple configuration

The following example presents a simple configuration file which can be used as a base for your Knot DNS setup:

```
# Example of a very simple Knot DNS configuration.

server:
    listen: 0.0.0.0@53
    listen: ::@53

zone:
  - domain: example.com
    storage: /var/lib/knot/zones/
    file: example.com.zone

log:
  - target: syslog
    any: info
```

Now let's walk through this configuration step by step:

- The *listen* statement in the *server section* defines where the server will listen for incoming connections. We have defined the server to listen on all available IPv4 and IPv6 addresses, all on port 53.
- The *zone section* defines the zones that the server will serve. In this case, we defined one zone named *example.com* which is stored in the zone file `/var/lib/knot/zones/example.com.zone`.
- The *log section* defines the log facilities for the server. In this example, we told Knot DNS to send its log messages with the severity `info` or more serious to the syslog (or systemd journal).

For detailed description of all configuration items see *Configuration Reference*.

4.2 Zone templates

A zone template allows a single zone configuration to be shared among several zones. There is no inheritance between templates; they are exclusive. The `default` template identifier is reserved for the default template:

```
template:
  - id: default
    storage: /var/lib/knot/master
    semantic-checks: on

  - id: signed
    storage: /var/lib/knot/signed
    dnssec-signing: on
    semantic-checks: on
```

(continues on next page)

(continued from previous page)

```
    master: [master1, master2]

  - id: slave
    storage: /var/lib/knot/slave

zone:
  - domain: example1.com     # Uses default template

  - domain: example2.com     # Uses default template
    semantic-checks: off     # Override default settings

  - domain: example.cz
    template: signed
    master: master3          # Override masters to just master3

  - domain: example1.eu
    template: slave
    master: master1

  - domain: example2.eu
    template: slave
    master: master2
```

Note

Each template option can be explicitly overridden in zone-specific configuration.

4.3 Access control list (ACL)

Normal DNS queries are always allowed. All other DNS requests must be authorized before they can be processed by the server. A zone can have configured *ACL* which is a sequence of rules describing what requests are authorized. An *automatic ACL* feature can be used to simplify ACL management.

Every ACL rule can allow or deny one or more request types (*actions*) based on the source IP address, network subnet, address range, protocol, remote certificate key PIN and/or if the request is secured by a given TSIG key. See *keymgr -t* on how to generate a TSIG key.

If there are multiple ACL rules assigned to a zone, they are applied in the specified order of the *acl* configuration. The first rule that matches the given request is applied and the remaining rules are ignored. Some examples:

```
acl:
  - id: address_rule
    address: [2001:db8::1, 192.168.2.0/24]
    action: transfer

  - id: deny_rule
    address: 192.168.2.100
    action: transfer
    deny: on

zone:
  - domain: acl1.example.com
    acl: [deny_rule, address_rule]     # Allow some addresses with an exception
```

```
key:
  - id: key1                           # The real TSIG key name
    algorithm: hmac-sha256
    secret: 4Tc0K1QkcMCs7cOW2LuSWnxQY0qysdvsZlSb4yTN9pA=

acl:
  - id: deny_all
    address: 192.168.3.0/24
    deny: on                           # No action specified and deny on implies␣
↪denial of all actions

  - id: key_rule
    key: key1                          # Access based just on TSIG key
    action: [transfer, notify]

zone:
  - domain: acl2.example.com
    acl: [deny_all, key_rule]          # Allow with the TSIG except for the subnet
```

In the case of dynamic DNS updates, some additional conditions may be specified for more granular filtering. See more in the section *Restricting dynamic updates*.

Note

If more conditions (address ranges and/or a key) are given in a single ACL rule, all of them have to be satisfied for the rule to match.

Tip

In order to restrict regular DNS queries, use module *queryacl*.

4.4 Secondary (slave) zone

Knot DNS doesn't strictly differ between primary (formerly known as master) and secondary (formerly known as slave) zones. The only requirement for a secondary zone is to have a *master* statement set. For effective zone synchronization, incoming zone change notifications (NOTIFY), which require authorization, can be enabled using *automatic ACL* or *explicit ACL* configuration. Optional transaction authentication (TSIG) is supported for both zone transfers and zone notifications:

```
server:
    automatic-acl: on                     # Enabled automatic ACL

key:
  - id: xfr_notify_key                    # Common TSIG key for XFR an NOTIFY
    algorithm: hmac-sha256
    secret: VFRejzw8h4M7mb0xZKRFiZAfhhd1eDGybjqHr2FV3vc=

remote:
  - id: primary
    address: [2001:DB8:1::1, 192.168.1.1] # Primary server IP addresses
    # via: [2001:DB8:2::1, 10.0.0.1]      # Local source addresses (optional)
    key: xfr_notify_key                   # TSIG key (optional)

zone:
```

(continues on next page)

(continued from previous page)

```
  - domain: example.com
    master: primary                        # Primary remote(s)
```

An example of explicit ACL with different TSIG keys for zone transfers and notifications:

```
key:
  - id: notify_key                         # TSIG key for NOTIFY
    algorithm: hmac-sha256
    secret: uBbhV4aeSS4fPd+wF2ZIn5pxOMF35xEtdq2ibi2hHEQ=

  - id: xfr_key                            # TSIG key for XFR
    algorithm: hmac-sha256
    secret: VFRejzw8h4M7mb0xZKRFiZAfhhd1eDGybjqHr2FV3vc=

remote:
  - id: primary
    address: [2001:DB8:1::1, 192.168.1.1] # Primary server IP addresses
    # via: [2001:DB8:2::1, 10.0.0.1]      # Local source addresses if needed
    key: xfr_key                           # Optional TSIG key

acl:
  - id: notify_from_primary                # ACL rule for NOTIFY from primary
    address: [2001:DB8:1::1, 192.168.1.1] # Primary addresses (optional)
    key: notify_key                        # TSIG key (optional)
    action: notify

zone:
  - domain: example.com
    master: primary                        # Primary remote(s)
    acl: notify_from_primary               # Explicit ACL(s)
```

Note that the *master* option accepts a list of remotes, which are queried for a zone refresh sequentially in the specified order. When the server receives a zone change notification from a listed remote, only that remote is used for a subsequent zone transfer.

Note

When transferring a lot of zones, the server may easily get into a state where all available ports are in the TIME_WAIT state, thus transfers cease until the operating system closes the ports for good. There are several ways to work around this:

- Allow reusing of ports in TIME_WAIT (sysctl -w net.ipv4.tcp_tw_reuse=1)
- Shorten TIME_WAIT timeout (tcp_fin_timeout)
- Increase available local port count

4.5 Primary (master) zone

A zone is considered primary if it doesn't have *master* set. As outgoing zone transfers (XFR) require authorization, it must be enabled using *automatic ACL* or *explicit ACL* configuration. Outgoing zone change notifications (NOTIFY) to remotes can be set by configuring *notify*. Transaction authentication (TSIG) is supported for both zone transfers and zone notifications:

```
server:
    automatic-acl: on                      # Enabled automatic ACL
```

(continues on next page)

(continued from previous page)

```
key:
  - id: xfr_notify_key                    # Common TSIG key for XFR an NOTIFY
    algorithm: hmac-sha256
    secret: VFRejzw8h4M7mb0xZKRFiZAfhhd1eDGybjqHr2FV3vc=

remote:
  - id: secondary
    address: [2001:DB8:1::1, 192.168.1.1] # Secondary server IP addresses
    # via: [2001:DB8:2::1, 10.0.0.1]      # Local source addresses (optional)
    key: xfr_notify_key                   # TSIG key (optional)

acl:
  - id: local_xfr                         # Allow XFR to localhost without TSIG
    address: [::1, 127.0.0.1]
    action: transfer

zone:
  - domain: example.com
    notify: secondary                     # Secondary remote(s)
    acl: local_xfr                        # Explicit ACL for local XFR
```

Note that the *notify* option accepts a list of remotes, which are all notified sequentially in the specified order.

A secondary zone may serve as a primary zone for a different set of remotes at the same time.

4.6 Dynamic updates

Dynamic updates for the zone are allowed via proper ACL rule with the `update` action. If the zone is configured as a secondary and a DNS update message is accepted, the server forwards the message to its first primary *master* or *ddns-master* if configured. The primary master's response is then forwarded back to the originator.

However, if the zone is configured as a primary, the update is accepted and processed:

```
acl:
  - id: update_acl
    address: 192.168.3.0/24
    action: update

zone:
  - domain: example.com.
    acl: update_acl
```

Note

To forward DDNS requests signed with a locally unknown key, an ACL rule for the action `update` without a key must be configured for the zone. E.g.:

```
acl:
  - id: fwd_foreign_key
    action: update
    # possible non-key options

zone:
 - domain: example.com.
   acl: fwd_foreign_key
```

4.6.1 Restricting dynamic updates

There are several additional ACL options for dynamic DNS updates which affect the request classification based on the update contents.

Updates can be restricted to specific resource record types:

```
acl:
  - id: type_rule
    action: update
    update-type: [A, AAAA, MX]    # Updated records must match one of the specified types
```

Another possibility is restriction on the owner name of updated records. The option *update-owner* is used to select the source of domain names which are used for the comparison. And the option *update-owner-match* specifies the required relation between the record owner and the reference domain names. Example:

```
acl:
  - id: owner_rule1
    action: update
    update-owner: name             # Updated record owners are restricted by the next conditions
    update-owner-match: equal      # The record owner must exactly match one name from the next list
    update-owner-name: [foo, bar.] # Reference domain names
```

Note

If the specified owner name is non-FQDN (e.g. `foo`), it's considered relatively to the effective zone name. So it can apply to more zones (e.g. `foo.example.com.` or `foo.example.net.`). Alternatively, if the name is FQDN (e.g. `bar.`), the rule only applies to this name.

If the reference domain name is the zone name, the following variant can be used:

```
acl:
  - id: owner_rule2
    action: update
    update-owner: zone            # The reference name is the zone name
    update-owner-match: sub       # Any record owner matches except for the zone name itself

template:
  - id: default
    acl: owner_rule2

zone:
  - domain: example.com.
  - domain: example.net.
```

The last variant is for the cases where the reference domain name is a TSIG key name, which must be used for the transaction security:

```
key:
  - id: example.com               # Key names are always considered FQDN
    ...
  - id: steve.example.net
    ...
  - id: jane.example.net
```

(continues on next page)

(continued from previous page)

```
    ...

acl:
  - id: owner_rule3_com
    action: update
    update-owner: key             # The reference name is the TSIG key name
    update-owner-match: sub       # The record owner must be a subdomain of the key␣
↪name
    key: [example.com]            # One common key for updating all non-apex records

  - id: owner_rule3_net
    action: update
    update-owner: key             # The reference name is the TSIG key name
    update-owner-match: equal     # The record owner must exactly match the used key␣
↪name
    key: [steve.example.net, jane.example.net] # Keys for updating specific zone nodes

zone:
 - domain: example.com.
   acl: owner_rule3_com
 - domain: example.net.
   acl: owner_rule3_net
```

4.6.2 Handling CNAME and DNAME-related updates

In general, no RR must exist beside a CNAME or below a DNAME. Whenever such a CNAME or DNAME-related semantic rule is vialoated by an RR addition in DDNS (this means addition of a CNAME beside an existing record, addition of another record beside a CNAME, addition of a DNAME above an existing record, addition of another record below a DNAME), such an RR addition is silently ignored. However, other RRs from the same DDNS update are processed normally. This is slightly non-compliant with RFC 6672 (in particular, no RR occlusion takes place).

4.7 Automatic DNSSEC signing

Knot DNS supports automatic DNSSEC signing of zones. The signing can operate in two modes:

1. *Manual key management*: In this mode, the server maintains zone signatures (RRSIGs) only. The signatures are kept up-to-date and signing keys are rolled according to the timing parameters assigned to the keys. The keys must be generated and timing parameters must be assigned by the zone operator.
2. *Automatic key management*: In this mode, the server maintains signing keys. New keys are generated according to the assigned policy and are rolled automatically in a safe manner. No intervention from the zone operator is necessary.

For automatic DNSSEC signing, a *policy* must be configured and assigned to the zone. The policy specifies how the zone is signed (i.e. signing algorithm, key size, key lifetime, signature lifetime, etc.). If no policy is specified, the default signing parameters are used.

The DNSSEC signing process maintains some metadata which is stored in the KASP (Key And Signature Policy) database. This database is backed by LMDB.

Warning

Make sure to set the KASP database permissions correctly. For manual key management, the database must be *readable* by the server process. For automatic key management, it must be *writeable*. If no HSM is used, the database also contains private key material – don't set the permissions too weak.

4.7.1 Manual key management

For automatic DNSSEC signing with manual key management, the *manual* has to be enabled in the policy:

```
policy:
  - id: manual
    manual: on

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: manual
```

To generate signing keys, use the *keymgr* utility. For example, we can use Single-Type Signing:

```
$ keymgr myzone.test. generate algorithm=ECDSAP256SHA256 ksk=yes zsk=yes
```

And reload the server. The zone will be signed.

To perform a manual rollover of a key, the timing parameters of the key need to be set. Let's roll the key. Generate a new key, but do not activate it yet:

```
$ keymgr myzone.test. generate algorithm=ECDSAP256SHA256 ksk=yes zsk=yes active=+1d
```

Take the key ID (or key tag) of the old key and disable it the same time the new key gets activated:

```
$ keymgr myzone.test. set <old_key_id> retire=+2d remove=+3d
```

Reload the server again. The new key will be published (i.e. the DNSKEY record will be added into the zone). Remember to update the DS record in the parent zone to include a reference to the new key. This must happen within one day (in this case) including a delay required to propagate the new DS to caches.

4.7.2 Automatic ZSK management

With *manual* disabled in the assigned policy (the default), the DNSSEC keys are generated automatically (if they do not already exist) and are also automatically rolled over according to their configured lifetimes. The default *zsk-lifetime* is finite, whereas *ksk-lifetime* infinite, meaning no KSK rollovers occur in the following example:

```
policy:
  - id: custom_policy
    signing-threads: 4
    algorithm: ECDSAP256SHA256
    zsk-lifetime: 60d

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: custom_policy
```

After configuring the server, reload the changes:

```
$ knotc reload
```

Check the server logs (regularly) to see whether everything went well.

Note

Enabling automatic key management with already existing keys requires attention:

- Any key timers set to future timestamps are automatically cleared, preventing interference with automatic operation procedures.
- If the keys are in an inconsistent state (e.g. an unexpected number of keys or active keys), it might lead to undefined behavior or, at the very least, a halt in key management.

4.7.3 Automatic KSK management

For automatic KSK management, first configure ZSK management as described above, and use *submission section* along with several options in *policy section*, specifying the desired (finite) lifetime for KSK and semi-automatic DS submission (see also *DNSSEC key states* and *DNSSEC key rollovers*):

```
remote:
  - id: parent_zone_server
    address: 192.168.12.1@53

submission:
  - id: parent_zone_sbm
    parent: [parent_zone_server]

policy:
  - id: custom_policy
    signing-threads: 4
    algorithm: ECDSAP256SHA256
    zsk-lifetime: 60d
    ksk-lifetime: 365d
    ksk-submission: parent_zone_sbm

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: custom_policy
```

After the initially-generated KSK reaches its lifetime, new KSK is published and after convenience delay the submission is started. The server publishes CDS and CDNSKEY records and the user shall propagate them to the parent. The server periodically checks for DS at the parent zone and when positive, finishes the rollover.

Note

When the initial keys are automatically generated for the first time, the KSK is actually in the `ready` state, allowing the initial parent DS submission to take place automatically.

4.7.4 Zone signing

The signing process consists of the following steps:

1. Processing KASP database events. (e.g. performing a step of a rollover).
2. Updating the DNSKEY records. The whole DNSKEY set in zone apex is replaced by the keys from the KASP database. Note that keys added into the zone file manually will be removed. To add an extra DNSKEY record into the set, the key must be imported into the KASP database (possibly deactivated).
3. Fixing the NSEC or NSEC3 chain.
4. Removing expired signatures, invalid signatures, signatures expiring in a short time, and signatures issued by an unknown key.
5. Creating missing signatures. Unless the Single-Type Signing Scheme is used, DNSKEY records in a zone apex are signed by KSK keys and all other records are signed by ZSK keys.

6. Updating and re-signing SOA record.

The signing is initiated on the following occasions:

- Start of the server
- Zone reload
- Reaching the signature refresh period
- Key set changed due to rollover event
- NSEC3 salt is changed
- Received DDNS update
- Forced zone re-sign via server control interface

On a forced zone re-sign, all signatures in the zone are dropped and recreated.

The `knotc zone-status` command can be used to see when the next scheduled DNSSEC re-sign will happen.

4.7.5 On-secondary (on-slave) signing

It is possible to enable automatic DNSSEC zone signing even on a secondary server. If enabled, the zone is signed after every AXFR/IXFR transfer from primary, so that the secondary always serves a signed up-to-date version of the zone.

It is strongly recommended to block any outside access to the primary server, so that only the secondary server's signed version of the zone is served.

Enabled on-secondary signing introduces events when the secondary zone changes while the primary zone remains unchanged, such as a key rollover or refreshing of RRSIG records, which cause inequality of zone SOA serial between primary and secondary. The secondary server handles this by saving the primary's SOA serial in a special variable inside KASP DB and appropriately modifying AXFR/IXFR queries/answers to keep the communication with primary server consistent while applying the changes with a different serial.

4.8 Catalog zones

Catalog zones (**RFC 9432**) are a concept whereby a list of zones to be configured is maintained as contents of a separate, special zone. This approach has the benefit of simple propagation of a zone list to secondary servers, especially when the list is frequently updated.

Terminology first. *Catalog zone* is a meta-zone which shall not be a part of the DNS tree, but it contains information about the set of member zones and is transferable to secondary servers using common AXFR/IXFR techniques. A *catalog-member zone* (or just *member zone*) is a zone based on information from the catalog zone and not from configuration file/database. *Member properties* are some additional information related to each member zone, also distributed with the catalog zone.

A catalog zone is handled almost in the same way as a regular zone: It can be configured using all the standard options (but for example DNSSEC signing is useless as the zone won't be queried by clients), including primary/secondary configuration and ACLs. A catalog zone is indicated by setting the option *catalog-role*. Standard DNS queries to a catalog zone are answered with REFUSED as though the zone doesn't exist unless there is a matching ACL rule for action transfer configured. The name of the catalog zone is arbitrary. It's possible to configure multiple catalog zones.

> **Warning**
>
> Don't choose a name for a catalog zone below a name of any other existing zones configured on the server as it would effectively "shadow" part of your DNS subtree.

Upon catalog zone (re)load or change, all the PTR records in the format `unique-id.zones.catalog. 0 IN PTR member.com.` (but not `too.deep.zones.catalog.`!) are processed and member zones created, with zone names taken from the PTR records' RData, and zone settings taken from the configuration templates specified by *catalog-template*.

The owner names of the PTR records shall follow this scheme:

```
<unique-id>.zones.<catalog-zone>.
```

where the mentioned labels shall match:

- *<unique-id>* — Single label that is recommended to be unique among member zones.
- `zones` — Required label.
- *<catalog-zone>* — Name of the catalog zone.

Additionally, records in the format `group.unique-id.zones.catalog. 0 IN TXT "conf-template"` are processed as a definition of the member's *group* property. The `unique-id` must match the one of the PTR record defining the member. It's required that at most one group is defined for each member. If multiple groups are defined, one group is picked at random.

All other records and other member properties are ignored. They remain in the catalog zone, however, and might be for example transferred to a secondary server, which may interpret catalog zones differently. SOA still needs to be present in the catalog zone and its serial handled appropriately. An apex NS record must be present as for any other zone. The version record `version 0 IN TXT "2"` is required at the catalog zone apex.

A catalog zone may be modified using any standard means (e.g. AXFR/IXFR, DDNS, zone file reload). In the case of incremental change, only affected member zones are reloaded.

The catalog zone must have at least one *catalog-template* configured. The configuration for any defined member zone is taken from its *group* property value, which should match some catalog-template name. If the *group* property is not defined for a member, is empty, or doesn't match any of defined catalog-template names, the first catalog-template (in the order from configuration) is used. Nesting of catalog zones isn't supported.

Any de-cataloged member zone is purged immediately, including its zone file, journal, timers, and DNSSEC keys. The zone file is not deleted if *zonefile-sync* is set to *-1* for member zones. Any member zone, whose PTR record's owner has been changed, is purged immediately if and only if the *<unique-id>* has been changed.

When setting up catalog zones, it might be useful to set *catalog-db* and *catalog-db-max-size* to non-default values.

Note

Whenever a catalog zone is updated, the server reloads itself with all configured zones, including possibly existing other catalog zones. It's similar to calling *knotc zone-reload* (for all zones). The consequence is that new zone files might be discovered and reloaded, even for zones that do not relate to updated catalog zone.

Catalog zones never expire automatically, regardless of what is declared in the catalog zone SOA. However, a catalog zone can be expired manually at any time using *knotc -f zone-purge +expire*.

Currently, expiration of a catalog zone doesn't have any effect on its member zones.

Warning

The server does not work well if one member zone appears in two catalog zones concurrently. The user is encouraged to avoid this situation whatsoever. Thus, there is no way a member zone can be migrated from one catalog to another while preserving its metadata. Following steps may be used as a workaround:

- *Back up* the member zone's metadata (on each server separately).
- Remove the member zone from the catalog it's a member of.
- Wait for the catalog zone to be propagated to all servers.
- Add the member zone to the other catalog.

- Restore the backed up metadata (on each server separately).

4.8.1 Catalog zones configuration examples

Below are configuration snippets (e.g. *server* and *log* sections missing) of very simple catalog zone setups, in order to illustrate the relations between catalog-related configuration options.

First setup represents a very simple scenario where the primary is the catalog zone generator and the secondary is the catalog zone consumer.

Primary configuration:

```
acl:
  - id: slave_xfr
    address: ...
    action: transfer

template:
  - id: mmemb
    catalog-role: member
    catalog-zone: catz.
    acl: slave_xfr

zone:
  - domain: catz.
    catalog-role: generate
    acl: slave_xfr

  - domain: foo.com.
    template: mmemb

  - domain: bar.com.
    template: mmemb
```

Secondary configuration:

```
acl:
  - id: master_notify
    address: ...
    action: notify

template:
  - id: smemb
    master: master
    acl: master_notify

zone:
  - domain: catz.
    master: master
    acl: master_notify
    catalog-role: interpret
    catalog-template: smemb
```

When new zones are added (or removed) to the primary configuration with assigned *mmemb* template, they will automatically propagate to the secondary and have the *smemb* template assigned there.

Second example is with a hand-written (or script-generated) catalog zone, while employing configuration groups:

```
catz.                   0       SOA     invalid. invalid. 1625079950 3600 600 2147483646 0
catz.                   0       NS      invalid.
version.catz.           0       TXT     "2"
nj2xg5bnmz2w4ltd.zones.catz.       0       PTR     just-fun.com.
group.nj2xg5bnmz2w4ltd.zones.catz. 0       TXT     unsigned
nvxxezjnmz2w4ltd.zones.catz.       0       PTR     more-fun.com.
group.nvxxezjnmz2w4ltd.zones.catz. 0       TXT     unsigned
nfwxa33sorqw45bo.zones.catz.       0       PTR     important.com.
group.nfwxa33sorqw45bo.zones.catz. 0       TXT     signed
mjqw42zomnxw2lq0.zones.catz.       0       PTR     bank.com.
group.mjqw42zomnxw2lq0.zones.catz. 0       TXT     signed
```

And the server in this case is configured to distinguish the groups by applying different templates:

```
template:
  - id: unsigned
    ...

  - id: signed
    dnssec-signing: on
    dnssec-policy: ...
    ...

zone:
  - domain: catz.
    file: ...
    catalog-role: interpret
    catalog-template: [ unsigned, signed ]
```

4.9 Database zone backend

It is possible to use Redis 7.0+ or a Valkey database for storing zone contents. The database does not need to be located on the same machine as the Knot DNS servers. One or more servers can read from or write to the same database. However, if multiple writers are used, they must be configured carefully to avoid conflicts or unexpected interactions.

On the database side, the `knot` module must be available (e.g. by installing the `redis-knot` package from a repository) and loaded into the database. This is usually done by adding the following line to the database configuration:

```
loadmodule /usr/lib/redis/modules/knot.so [<parameter> <value>]...
```

where the optional module parameters are:

- `default-ttl` — The default record TTL value if not specified (default is 600 seconds).
- `max-event-age` — The maximum age of the stored zone events (default is 1200 seconds).
- `max-update-depth` — The maximum depth of the zone update history (default is 20).

On the Knot DNS side, at least an address or a UNIX socket path must be configured *zone-db-listen* in the *database section*. Optionally, TLS communication can be enabled using *zone-db-tls*, *zone-db-cert-key*, or *zone-db-cert-hostname*. For example:

```
database:
  zone-db-listen: /run/redis/redis-server.sock
```

or:

```
database:
  zone-db-listen: 192.168.1.2@6379
  zone-db-tls: on
  zone-db-cert-key: s3L4U7E41DnN2tHFT8bu9DQe6eo2ySehlltyzdLbWjg=
```

In the database, up to 8 independent versions of zone contents — called zone *instances* — can be stored per zone, numbered from 1 to 8. Whether a zone is loaded from or store to the database is configured independently using the *zone-db-input* and *zone-db-output* options, whose values correspond to zone instances. For example, if a zone should be read from database instance 1 and the DNSSEC-signed version stored in instance 2, the configuration may look like this:

```
zone:
  - domain: example.com.
    dnssec-signing: on
    zone-db-input: 1
    zone-db-output: 2
```

It is still possible to combine zone database backend with zone file or journal operations.

For manual operations on the database, see *Zone database*.

4.10 DNS over QUIC

QUIC is a low-latency, encrypted, internet transport protocol. Knot DNS supports DNS over QUIC (DoQ) (**RFC 9250**), including zone transfers (XoQ). By default, the UDP port *853* is used for DNS over QUIC.

To use QUIC, a server *private key* and a *certificate* must be available. If no key is configured, the server automatically generates one with a self-signed temporary certificate. The key is stored in the KASP database directory for persistence across restarts.

In order to listen for incoming requests over QUIC, at least one *interface* or *XDP interface* must be configured.

An example of configuration of listening for DNS over QUIC on the loopback interface:

```
server:
  listen-quic: ::1
```

When the server is started, it logs some interface details and public key pin of the used certificate:

```
... info: binding to QUIC interface ::1@853
... info: QUIC/TLS, certificate public key 0xtdayWpnJh4Py8goi8cei/
↪gXGD4kJQ+HEqcxS++DBw=
```

Tip

The public key pin, which isn't secret, can also be displayed via:

```
$ knotc status cert-key
0xtdayWpnJh4Py8goi8cei/gXGD4kJQ+HEqcxS++DBw=
```

Or from the keyfile via:

```
$ certtool --infile=quic_key.pem -k | grep pin-sha256
     pin-sha256:0xtdayWpnJh4Py8goi8cei/gXGD4kJQ+HEqcxS++DBw=
```

Using *kdig* we can verify that the server responds over QUIC:

```
$ kdig @::1 ch txt version.server +quic
;; QUIC session (QUICv1)-(TLS1.3)-(ECDHE-X25519)-(EdDSA-Ed25519)-(AES-256-GCM)
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 0
;; Flags: qr rd; QUERY: 1; ANSWER: 1; AUTHORITY: 0; ADDITIONAL: 1

;; EDNS PSEUDOSECTION:
;; Version: 0; flags: ; UDP size: 1232 B; ext-rcode: NOERROR
;; PADDING: 370 B

;; QUESTION SECTION:
;; version.server.                    CH      TXT

;; ANSWER SECTION:
version.server.       0       CH      TXT     "Knot DNS 3.4.0"

;; Received 468 B
;; Time 2024-06-21 08:30:12 CEST
;; From ::1@853(QUIC) in 1.1 ms
```

In this case, **opportunistic authentication** was used, which doesn't guarantee that the client communicates with the genuine server and vice versa. For **strict authentication** of the server, we can enforce certificate key pin check by specifying it (enabled debug mode for details):

```
$ kdig @::1 ch txt version.server +tls-pin=0xtdayWpnJh4Py8goi8cei/
↪gXGD4kJQ+HEqcxS++DBw= +quic -d
;; DEBUG: Querying for owner(version.server.), class(3), type(16), server(::1),␣
↪port(853), protocol(UDP)
;; DEBUG: TLS, received certificate hierarchy:
;; DEBUG:  #1, CN=tester
;; DEBUG:      SHA-256 PIN: 0xtdayWpnJh4Py8goi8cei/gXGD4kJQ+HEqcxS++DBw=, MATCH
;; DEBUG: TLS, skipping certificate verification
;; QUIC session (QUICv1)-(TLS1.3)-(ECDHE-X25519)-(EdDSA-Ed25519)-(AES-256-GCM)
...
```

We see that a server certificate key matches the specified pin. Another possibility is to use certificate chain validation if a suitable certificate is configured on the server.

4.10.1 Zone transfers

For outgoing requests (e.g. NOTIFY and refresh), Knot DNS utilizes **session resumption**, which speeds up QUIC connection establishment.

Here are a few examples of zone transfer configurations using various **authentication mechanisms**:

Opportunistic authentication:

Primary and secondary can authenticate using TSIG. Fallback to clear-text DNS isn't supported.

Primary:

```
server:
    listen-quic: ::1
    automatic-acl: on

key:
  - id: xfr_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=
```

(continues on next page)

(continued from previous page)

```
remote:
  - id: secondary
    address: ::2
    key: xfr_key  # TSIG for secondary authentication
    quic: on

zone:
  - domain: example.com
    notify: secondary
```

Secondary:

```
server:
    listen-quic: ::2
    automatic-acl: on

key:
  - id: xfr_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=

remote:
  - id: primary
    address: ::1
    key: xfr_key  # TSIG for primary authentication
    quic: on

zone:
  - domain: example.com
    master: primary
```

Strict authentication:

Note that the automatic ACL doesn't work in this case due to asymmetrical configuration. The secondary can authenticate using TSIG.

With PIN checks:

Primary:

```
server:
    listen-quic: ::1

key:
  - id: secondary_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=

remote:
  - id: secondary
    address: ::2
    quic: on

acl:
  - id: secondary_xfr
    address: ::2
    key: secondary_key  # TSIG for secondary authentication
```

(continues on next page)

(continued from previous page)

```
    action: transfer

zone:
  - domain: example.com
    notify: secondary
    acl: secondary_xfr
```

Secondary:

```
server:
    listen-quic: ::2

key:
  - id: secondary_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=

remote:
  - id: primary
    address: ::1
    key: secondary_key  # TSIG for secondary authentication
    quic: on

acl:
  - id: primary_notify
    address: ::1
    cert-key: 0xtdayWpnJh4Py8goi8cei/gXGD4kJQ+HEqcxS++DBw=
    action: notify

zone:
  - domain: example.com
    master: primary
    acl: primary_notify
```

With CA and hostname checks:

Primary

```
server:
    listen-quic: ::1
    cert-file: primary-cert.pem
    key-file: primary-key.pem

key:
  - id: secondary_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=

remote:
  - id: secondary
    address: ::2
    quic: on

acl:
  - id: secondary_xfr
    address: ::2
    key: secondary_key  # TSIG for secondary authentication
```

(continues on next page)

(continued from previous page)

```
    action: transfer

zone:
  - domain: example.com
    notify: secondary
    acl: secondary_xfr
```

Secondary:

```
server:
    listen-quic: ::2
    ca-file: ca-cert.pem

key:
  - id: secondary_key
    algorithm: hmac-sha256
    secret: S059OFJv1SCDdR2P6JKENgWaM409iq2X44igcJdERhc=

remote:
  - id: primary
    address: ::1
    key: secondary_key  # TSIG for secondary authentication
    quic: on

acl:
  - id: primary_notify
    address: ::1
    cert-hostname: "Primary Knot"
    action: notify

zone:
  - domain: example.com
    master: primary
    acl: primary_notify
```

Mutual authentication:

The **mutual authentication** guarantees authentication for both the primary and the secondary. In this case, TSIG would be redundant. This mode is recommended if possible.

With PIN checks:

Primary:

```
server:
    listen-quic: ::1
    automatic-acl: on

remote:
  - id: secondary
    address: ::2
    quic: on
    cert-key: PXqv7/lXn6N7scg/KJWvfU/TEPe5BoIUHQGRLMPr6YQ=

zone:
  - domain: example.com
    notify: secondary
```

Secondary:

```
server:
    listen-quic: ::2
    automatic-acl: on

remote:
  - id: primary
    address: ::1
    quic: on
    cert-key: 0xtdayWpnJh4Py8goi8cei/gXGD4kJQ+HEqcxS++DBw=

zone:
  - domain: example.com
    master: primary
```

With CA and hostname checks:

Primary:

```
server:
    listen-quic: ::1
    ca-file: ca-cert.pem
    cert-file: primary-cert.pem
    key-file: primary-key.pem
    automatic-acl: on

remote:
  - id: secondary
    address: ::2
    quic: on
    cert-hostname: "Secondary Knot"

zone:
  - domain: example.com
    notify: secondary
```

Secondary:

```
server:
    listen-quic: ::2
    ca-file: ca-cert.pem
    cert-file: secondary-cert.pem
    key-file: secondary-key.pem
    automatic-acl: on

remote:
  - id: primary
    address: ::1
    quic: on
    cert-hostname: "Primary Knot"

zone:
  - domain: example.com
    master: primary
```

Tip

Using GnuTLS certtool you can generate a CA certificate with its private key:

```
$ certtool --generate-privkey --key-type ed25519 --outfile ca-key.pem
$ echo -e "cn = \"My Example CA\"\nca\ncert_signing_key\nexpiration_days = 3650" >
↪ca-template.info
$ certtool --generate-self-signed --load-privkey ca-key.pem \
           --template ca-template.info --outfile ca-cert.pem
```

Then create certificates signed with this CA like so:

```
$ CERT_NAME="primary"
$ certtool --generate-privkey --key-type ed25519 --outfile ${CERT_NAME}-key.pem
$ echo -e "dns_name = \"${CERT_NAME} server\"\nexpiration_days = 365" >${CERT_NAME}
↪-template.info
$ certtool --generate-certificate --load-privkey ${CERT_NAME}-key.pem \
           --load-ca-certificate ca-cert.pem --load-ca-privkey ca-key.pem \
           --template ${CERT_NAME}-template.info --outfile ${CERT_NAME}-cert.pem
```

If you want to use a wildcard DNSName in your certificate, beware that GnuTLS, which is the TLS backend for Knot DNS, **will not verify** wildcard names directly under TLDs (like `*.example`).

To see a server's TLS hostnames:

```
$ kdig @1.1.1.1 +tls -dd
;; DEBUG: Querying for owner(.), class(1), type(2), server(1.1.1.1), port(853),␣
↪protocol(TCP)
;; DEBUG: TLS, received certificate hierarchy:
;; DEBUG:  #1, CN=cloudflare-dns.com,O=Cloudflare\, Inc.,L=San Francisco,
↪ST=California,C=US
;; DEBUG:      Subject Alternative Name:
;; DEBUG:        DNSName: cloudflare-dns.com
;; DEBUG:        DNSName: *.cloudflare-dns.com
;; DEBUG:        DNSName: one.one.one.one
[...]
```

Knot DNS will only verify hostnames under the *Subject Alternative Name* extension in compliance with **RFC 8310 Section 8.1**.

Note

Certificate validation by CA and hostname is more computationally expensive than by PIN, but PIN checking has the disadvantage of relying on constantness of the public key.

4.11 DNS over TLS

TLS is an encrypted internet transport protocol. Knot DNS supports DNS over TLS (DoT) (**RFC 7858**), including zone transfers (XoT). By default, the TCP port *853* is used for DNS over TLS.

There are the same requirements for TLS key and certificate as for *DNS over QUIC*.

In order to listen for incoming requests over TLS, *interface* must be configured.

An example of configuration of listening for DNS over TLS on the loopback interface:

```
server:
  listen-tls: ::1
```

When the server is started, it logs some interface details and public key pin of the used certificate:

```
... info: binding to TLS interface ::1@853
... info: QUIC/TLS, certificate public key 0xtdayWpnJh4Py8goi8cei/
↪gXGD4kJQ+HEqcxS++DBw=
```

Using *kdig* we can verify that the server responds over TLS:

```
$ kdig @::1 ch txt version.server +tls
;; TLS session (TLS1.3)-(ECDHE-X25519)-(EdDSA-Ed25519)-(AES-256-GCM)
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 0
;; Flags: qr rd; QUERY: 1; ANSWER: 1; AUTHORITY: 0; ADDITIONAL: 1

;; EDNS PSEUDOSECTION:
;; Version: 0; flags: ; UDP size: 1232 B; ext-rcode: NOERROR
;; PADDING: 370 B

;; QUESTION SECTION:
;; version.server.                    CH      TXT

;; ANSWER SECTION:
version.server.       0       CH      TXT     "Knot DNS 3.4.0"

;; Received 468 B
;; Time 2024-06-21 08:31:13 CEST
;; From ::1@853(TLS) in 9.1 ms
```

Zone transfer configuration and authentication profiles are almost identical to *DNS over QUIC*, with the only difference being the enabling of *tls* for the corresponding remotes.

4.12 Query modules

Knot DNS supports configurable query modules that can alter the way queries are processed. Each query requires a finite number of steps to be resolved. We call this set of steps a *query plan*, an abstraction that groups these steps into several stages.

- Before-query processing
- Answer, Authority, Additional records packet sections processing
- After-query processing

For example, processing an Internet-class query needs to find an answer. Then based on the previous state, it may also append an authority SOA or provide additional records. Each of these actions represents a 'processing step'. Now, if a query module is loaded for a zone, it is provided with an implicit query plan which can be extended by the module or even changed altogether.

A module is active if its name, which includes the `mod-` prefix, is assigned to the zone/template *module* option or to the `default` template *global-module* option if activating for all queries. If the module is configurable, a corresponding module section with an identifier must be created and then referenced in the form of `module_name/module_id`. See *Modules* for the list of available modules.

The same module can be specified multiple times, such as a global module and a per-zone module, or with different configurations. However, not all modules are intended for this, for example, mod-cookies! Global modules are executed before per-zone modules.

Note

Query modules are processed in the order they are specified in the zone/template configuration. In most cases, the recommended order is:

```
mod-synthrecord, mod-onlinesign, mod-cookies, mod-rrl, mod-dnstap, mod-stats
```

4.13 Performance Tuning

4.13.1 Numbers of Workers

There are three types of workers ready for parallel execution of performance-oriented tasks: UDP workers, TCP workers, and Background workers. The first two types handle all network requests via the UDP and TCP protocol (respectively) and do the response jobs for common queries. Background workers process changes to the zone.

By default, Knot determines a well-fitting number of workers based on the number of CPU cores. The user can specify the number of workers for each type with configuration/server section: *udp-workers*, *tcp-workers*, *background-workers*.

An indication of when to increase the number of workers is when the server is lagging behind expected performance, while CPU usage remains low. This is usually due to waiting for network or I/O response during the operation. It may be caused by Knot design not fitting the use-case well. The user should try increasing the number of workers (of the related type) slightly above 100 and if the performance improves, decide a further, exact setting.

4.13.2 Number of available file descriptors

A name server configured for a large number of zones (hundreds or more) needs enough file descriptors available for zone transfers and zone file updates, which default OS settings often don't provide. It's necessary to check with the OS configuration and documentation and ensure the number of file descriptors (sometimes called a number of concurrently open files) effective for the knotd process is set suitably high. The number of concurrently open incoming TCP connections must be taken into account too. In other words, the required setting is affected by the *tcp-max-clients* setting.

4.13.3 Sysctl and NIC optimizations

There are several recommendations based on Knot developers' experience with their specific HW and SW (mainstream Intel-based servers, Debian-based GNU/Linux distribution). They may improve or impact performance in common use cases.

If your NIC driver allows it (see /proc/interrupts for hint), set CPU affinity (/proc/irq/$IRQ/smp_affinity) manually so that each NIC channel is served by unique CPU core(s). You must turn off irqbalance service in advance to avoid configuration override.

Configure sysctl as follows:

```
socket_bufsize=1048576
busy_latency=0
backlog=40000
optmem_max=20480

net.core.wmem_max     = $socket_bufsize
net.core.wmem_default = $socket_bufsize
net.core.rmem_max     = $socket_bufsize
net.core.rmem_default = $socket_bufsize
net.core.busy_read = $busy_latency
net.core.busy_poll = $busy_latency
net.core.netdev_max_backlog = $backlog
net.core.optmem_max = $optmem_max
```

Disable huge pages.

Configure your CPU to "performance" mode. This can be achieved depending on architecture, e.g. in BIOS, or e.g. configuring /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor to "performance".

Tune your NIC device with ethtool:

```
ethtool -A $dev autoneg off rx off tx off
ethtool -K $dev tso off gro off ufo off
ethtool -G $dev rx 4096 tx 4096
ethtool -C $dev rx-usecs 75
ethtool -C $dev tx-usecs 75
ethtool -N $dev rx-flow-hash udp4 sdfn
ethtool -N $dev rx-flow-hash udp6 sdfn
```

On FreeBSD you can just:

```
ifconfig ${dev} -rxcsum -txcsum -lro -tso
```

Knot developers are open to hear about users' further suggestions about network devices tuning/optimization.

CHAPTER

FIVE

OPERATION

The Knot DNS server part *knotd* can run either in the foreground, or in the background using the `-d` option. When run in the foreground, it doesn't create a PID file. Other than that, there are no differences and you can control both the same way.

The tool *knotc* is designed as a user front-end, making it easier to control a running server daemon. If you want to control the daemon directly, use `SIGINT` to quit the process or `SIGHUP` to reload the configuration.

If you pass neither configuration file (`-c` parameter) nor configuration database (`-C` parameter), the server will first attempt to use the default configuration database stored in `/var/lib/knot/confdb` or the default configuration file stored in `/etc/knot/knot.conf`. Both the default paths can be reconfigured with `--with-storage=path` or `--with-configdir=path` respectively.

Example of server start as a daemon:

```
$ knotd -d -c knot.conf
```

Example of server shutdown:

```
$ knotc -c knot.conf stop
```

For a complete list of actions refer to the program help (`-h` parameter) or to the corresponding manual page.

Also, the server needs to create *rundir* and *storage* directories in order to run properly.

Note

Avoid editing of or other manipulation with configuration file during start or reload of *knotd* or start of *knotc* and other *utilities* which use it. There is a risk of malfunction or a *crash* otherwise.

5.1 Configuration database

In the case of a huge configuration file, the configuration can be stored in a binary database. Such a database can be simply initialized:

```
$ knotc conf-init
```

or preloaded from a file:

```
$ knotc conf-import input.conf
```

Also the configuration database can be exported into a textual file:

```
$ knotc conf-export output.conf
```

Warning

The import and export commands access the configuration database directly, without any interaction with the server. Therefore, any data not yet committed to the database won't be exported. And the server won't reflect imported configuration correctly. So it is strictly recommended to import new configuration when the server is not running.

5.2 Dynamic configuration

The configuration database can be accessed using the server control interface while the server is running. To get the full power of the dynamic configuration, the server must be started with a specified configuration database location or with the default database initialized. Otherwise all the changes to the configuration will be temporary (until the server is stopped).

Note

The database can be *imported* in advance.

Most of the commands get an item name and value parameters. The item name is in the form of `section[identifier].name`. If the item is multivalued, more values can be specified as individual (command line) arguments.

Caution

Beware of the possibility of pathname expansion by the shell. For this reason, it is advisable to escape (with backslash) square brackets or to quote command parameters if not executed in the interactive mode.

To get the list of configuration sections or to get the list of section items:

```
$ knotc conf-list
$ knotc conf-list 'server'
```

To get the whole configuration or to get the whole configuration section or to get all section identifiers or to get a specific configuration item:

```
$ knotc conf-read
$ knotc conf-read 'remote'
$ knotc conf-read 'zone.domain'
$ knotc conf-read 'zone[example.com].master'
```

Warning

The following operations don't work on OpenBSD!

Modifying operations require an active configuration database transaction. Just one transaction can be active at a time. Such a transaction then can be aborted or committed. A semantic check is executed automatically before every commit:

```
$ knotc conf-begin
$ knotc conf-abort
$ knotc conf-commit
```

To set a configuration item value or to add more values or to add a new section identifier or to add a value to all identified sections:

```
$ knotc conf-set 'server.identity' 'Knot DNS'
$ knotc conf-set 'server.listen' '0.0.0.0@53' '::@53'
$ knotc conf-set 'zone[example.com]'
$ knotc conf-set 'zone.slave' 'slave2'
```

Note

Also the include operation can be performed. A non-absolute file location is relative to the server binary path, not to the control binary path!

```
$ knotc conf-set 'include' '/tmp/new_zones.conf'
```

To unset the whole configuration or to unset the whole configuration section or to unset an identified section or to unset an item or to unset a specific item value:

```
$ knotc conf-unset
$ knotc conf-unset 'zone'
$ knotc conf-unset 'zone[example.com]'
$ knotc conf-unset 'zone[example.com].master'
$ knotc conf-unset 'zone[example.com].master' 'remote2' 'remote5'
```

To get the change between the current configuration and the active transaction for the whole configuration or for a specific section or for a specific identified section or for a specific item:

```
$ knotc conf-diff
$ knotc conf-diff 'zone'
$ knotc conf-diff 'zone[example.com]'
$ knotc conf-diff 'zone[example.com].master'
```

Caution

While it is possible to change most of the configuration parameters dynamically or via configuration file reload, a few of the parameters in the section `server` require restarting the server, such that the changes take effect. These parameters are: *rundir*, *user*, *pidfile*, *tcp-reuseport*, *udp-workers*, *tcp-workers*, *background-workers*, and *listen*.

An example of possible configuration initialization:

```
$ knotc conf-begin
$ knotc conf-set 'server.listen' '0.0.0.0@53' '::@53'
$ knotc conf-set 'remote[master_server]'
$ knotc conf-set 'remote[master_server].address' '192.168.1.1'
$ knotc conf-set 'template[default]'
$ knotc conf-set 'template[default].storage' '/var/lib/knot/zones/'
$ knotc conf-set 'template[default].master' 'master_server'
$ knotc conf-set 'zone[example.com]'
$ knotc conf-diff
$ knotc conf-commit
```

5.3 Secondary (slave) mode

Running the server as a secondary is very straightforward as the zone is transfered automatically from a remote server. The received zone is usually stored in a zone file after the *zonefile-sync* period elapses. Zone differences are stored in the *zone journal*.

5.4 Primary (master) mode

If you just want to check the zone files before starting, you can use:

```
$ knotc zone-check example.com
```

5.5 Reading and editing zones

Knot DNS allows you to read or change zone contents online using the server control interface.

To get contents of all configured zones, or a specific zone contents, or zone records with a specific owner, or even with a specific record type:

```
$ knotc zone-read --
$ knotc zone-read example.com
$ knotc zone-read example.com ns1
$ knotc zone-read example.com ns1 NS
```

Note

If the record owner is not a fully qualified domain name, then it is considered as a relative name to the zone name.

To start a writing transaction on all zones or on specific zones:

```
$ knotc zone-begin --
$ knotc zone-begin example.com example.net
```

Now you can list all nodes within the transaction using the `zone-get` command, which always returns current data with all changes included. The command has the same syntax as `zone-read`.

Warning

If a zone transaction is open, all zone events that modify the zone (e.g. DNSSEC signing) are blocked until the transaction is eithe committed or aborted. It is therefore advisable to finish the transaction quickly and commit without delay.

Within the transaction, you can add a record to a specific zone or to all zones with an open transaction:

```
$ knotc zone-set example.com ns1 3600 A 192.168.0.1
$ knotc zone-set -- ns1 3600 A 192.168.0.1
```

To remove all records with a specific owner, or a specific rrset, or specific record data:

```
$ knotc zone-unset example.com ns1
$ knotc zone-unset example.com ns1 A
$ knotc zone-unset example.com ns1 A 192.168.0.2
```

To see the difference between the original zone and the current version:

```
$ knotc zone-diff example.com
```

Finally, either commit or abort your transaction:

```
$ knotc zone-commit example.com
$ knotc zone-abort example.com
```

A full example of setting up a completely new zone from scratch:

```
$ knotc conf-begin
$ knotc conf-set zone.domain example.com
$ knotc conf-commit
$ knotc zone-begin example.com
$ knotc zone-set example.com  @ 3600 SOA  ns admin 1 86400 900 691200 3600
$ knotc zone-set example.com  @ 3600 NS   ns
$ knotc zone-set example.com ns 3600 A    192.168.0.1
$ knotc zone-set example.com ns 3600 AAAA 2001:DB8::1
$ knotc zone-commit example.com
```

Note

If quotes are necessary for record data specification, remember to escape them:

```
$ knotc zone-set example.com @ 3600 TXT \"v=spf1 a:mail.example.com -all\"
```

5.6 Reading and editing the zone file safely

It's always possible to read and edit zone contents via zone file manipulation. It may lead to confusion or even a *program crash*, however, if the zone contents are continuously being changed by DDNS, DNSSEC signing and the like. In such a case, the safe way to modify the zone file is to freeze zone events first:

```
$ knotc -b zone-freeze example.com.
$ knotc -b zone-flush example.com.
```

After calling freeze on the zone, there still may be running zone operations (e.g. signing), causing freeze pending. Because of this, the blocking mode is used to ensure the operation was finished. Then the zone can be flushed to a file.

Now the zone file can be safely modified (e.g. using a text editor). If *zonefile-load* is not set to *difference-no-serial*, it's also necessary to **increase SOA serial** in this step to keep consistency. Finally, we can load the modified zone file and if successful, thaw the zone:

```
$ knotc -b zone-reload example.com.
$ knotc zone-thaw example.com.
```

5.7 Zone loading

The process of how the server loads a zone is influenced by the configuration of the *zonefile-load* and *journal-content* parameters (also DNSSEC signing applies), the existence of a zone file and journal (and their relative out-of-dateness), and whether it is a cold start of the server or a zone reload (e.g. invoked by the *knotc* interface). Please note that zone transfers are not taken into account here – they are planned after the zone is loaded (including *zone bootstrap*).

If the zone file exists and is not excluded by the configuration, it is first loaded and according to its SOA serial number, relevant journal changesets are applied. If this is a zone reload and we have *zonefile-load* set to *difference*, the difference between old and new contents is computed and stored in the journal like an update. The zone file should be either unchanged since last load or changed with incremented SOA serial. In the case of a decreased SOA serial, the load is interrupted with an error; if unchanged, it is increased by the server.

If the procedure described above succeeds without errors, the resulting zone contents are (after potential DNSSEC signing) used as the new zone.

The option *journal-content* set to *all* lets the server, beside better performance, keep track of the zone contents also across server restarts. It makes the cold start effectively work like a zone reload with the old contents loaded from the journal (unless this is the very first start with the zone not yet saved into the journal).

5.8 Journal behaviour

The zone journal keeps some history of changes made to the zone. It is useful for responding to IXFR queries. Also if *zone file flush* is disabled, the journal keeps the difference between the zone file and the current zone in case of server shutdown. The history is stored in changesets – differences of zone contents between two (usually subsequent) zone versions (specified by SOA serials).

Journals of all zones are stored in a common LMDB database. Huge changesets are split into 15-70 KiB[1] blocks to prevent fragmentation of the DB. The journal does each operation in one transaction to keep consistency of the DB and performance.

Each zone journal has its own occupation limits *maximum usage* and *maximum depth*. Changesets are stored in the journal one by one. When hitting any of the limits, the zone is flushed into the zone file if there are no redundant changesets to delete, and the oldest changesets are deleted. In the case of the size limit, twice[1] the needed amount of space is purged to prevent overly frequent deletes.

If *zone file flush* is disabled, then instead of flushing the zone, the journal tries to save space by merging the changesets into a special one. This approach is effective if the changes rewrite each other, e.g. periodically changing the same zone records, re-signing the whole zone etc. Thus the difference between the zone file and the zone is still preserved even if the journal deletes some older changesets.

If the journal is used to store both zone history and contents, a special changeset is present with zone contents. When the journal gets full, the changes are merged into this special changeset.

There is also a *safety hard limit* for overall journal database size, but it's strongly recommended to set the per-zone limits in a way to prevent hitting this one. For LMDB, it's hard to recover from the database-full state. For wiping one zone's journal, see *knotc zone-purge +journal* command.

5.9 Handling zone file, journal, changes, serials

Some configuration options regarding the zone file and journal, together with operation procedures, might lead to unexpected results. This chapter points out potential interference and both recommends and warns before some combinations thereof. Unfortunately, there is no optimal combination of configuration options, every approach has some disadvantages.

5.9.1 Example 1

Keep the zone file updated:

```
zonefile-sync: 0
zonefile-load: whole
journal-content: changes
```

These are default values. The user can always check the current zone contents in the zone file, and also modify it (recommended with server turned-off or taking the *safe way*). The journal serves here just as a source of history for secondary servers' IXFR. Some users dislike that the server overwrites their prettily prepared zone file.

[1] This constant is hardcoded.

5.9.2 Example 2

Zonefileless setup:

```
zonefile-sync: -1
zonefile-load: none
journal-content: all
```

Zone contents are stored only in the journal. The zone is updated by DDNS, zone transfer, or via the control interface. The user might have filled the zone contents initially from a zone file by setting *zonefile-load* to *whole* temporarily. It's also a good setup for secondary servers. Anyway, it's recommended to carefully tune the journal-size-related options to avoid surprises like the journal getting full (see *Journal behaviour*).

5.9.3 Example 3

Input-only zone file:

```
zonefile-sync: -1
zonefile-load: difference
journal-content: changes
```

The user can make changes to the zone by editing the zone file, and his pretty zone file is never overwritten or filled with DNSSEC-related autogenerated records – they are only stored in the journal.

Warning

The zone file's SOA serial must be properly set to a number which is higher than the current SOA serial in the zone (not in the zone file) if manually updated! This is important to ensure consistency of the journal and outgoing IXFR.

Note

This mode is not suitable if the zone can be modified externally (e.g. DDNS, knotc).

5.9.4 Example 4

Auto-increment SOA serial:

```
zonefile-sync: -1
zonefile-load: difference-no-serial
journal-content: all
```

This is similar to the previous setup, but the SOA serial is handled by the server automatically. So the user no longer needs to care about it in the zone file.

However, this requires setting *journal-content* to *all* so that the information about the last real SOA serial is preserved in case of server re-start. The sizing of journal limits needs to be taken into consideration (see *Journal behaviour*).

Note

This mode is not suitable if the zone can be modified externally (e.g. DDNS, knotc).

5.9.5 Example 5

Managing specific record types externally:

```
zonefile-sync: -1
zonefile-load: difference-no-serial
journal-content: all
zonefile-skip: [ "DS" ]
```

Most of the zone contents are still managed by editing the input-only zone file, but some specific resource record types (e.g. DS) are managed externally (e.g. via DDNS or *knotc*). The *zonefile-skip* option ensures that both approaches do not interfere and that additions of these records are not reverted when reloading the zone file that does not contain them.

5.10 Zone database

The *zone database* can be accessed directly through the database CLI or API, without any interaction with a Knot DNS server. The `knot` module provides specific commands for zone data manipulation.

The `KNOT.ZONE.*` commands are used for manipulating entire zones:

- `KNOT.ZONE.BEGIN <zone> <instance>` — Initializes a zone transaction for the specified instance. Returns a transaction ID, which is then used as a parameter for subsequent commands.
- `KNOT.ZONE.STORE <zone> <txn> "data"` — Stores zone data in a zone transaction. The data must be provided in the form of a textual zone file and may contain one or more records (including a full zone file content).
- `KNOT.ZONE.COMMIT <txn>` — Commits the specified zone transaction.
- `KNOT.ZONE.ABORT <txn>` — Aborts the specified zone transaction.
- `KNOT.ZONE.LOAD [--compact] <zone> <instance|txn> [owner] [type]` — Returns the contents of the specified zone instance, optionally in compact format and filtered by owner and record type.
- `KNOT.ZONE.PURGE <zone>` — Purges all data for the specified zone.
- `KNOT.ZONE.LIST [--instances]` — Lists stored zones, optionally including available instances.

Example of a zone initialization:

```
$ redis-cli KNOT.ZONE.BEGIN example.com 1
(integer) 10

$ redis-cli KNOT.ZONE.STORE example.com 10 "@ SOA ns admin 1 86400 900 691200 3600"
OK
$ redis-cli KNOT.ZONE.STORE example.com 10 "@ NS ns"
OK
$ redis-cli KNOT.ZONE.STORE example.com 10 "ns 300 AAAA ::1"
OK

$ redis-cli KNOT.ZONE.LOAD example.com 10
1) 1) "example.com."
   2) "600"
   3) "SOA"
   4) "ns.example.com. admin.example.com. 1 86400 900 691200 3600"
2) 1) "example.com."
   2) "600"
   3) "NS"
   4) "ns.example.com."
3) 1) "ns.example.com."
   2) "300"
```

(continues on next page)

(continued from previous page)

```
   3) "AAAA"
   4) "::1"

$ redis-cli KNOT.ZONE.COMMIT example.com 10
OK

$ redis-cli KNOT.ZONE.LOAD --compact example.com 1
1) "example.com. 600 SOA ns.example.com. admin.example.com. 1 86400 900 691200 3600"
2) "example.com. 600 NS ns.example.com."
3) "ns.example.com. 300 AAAA ::1"

$ redis-cli KNOT.ZONE.LIST --instances
1) "example.com.: 1"
```

Tip

For storing the entire zone file, the following command works:

```
$ cat example.com.zone | redis-cli --raw -x KNOT.ZONE.STORE example.com 10
```

In the case of very large zone files, ensure that the database limits `proto-max-bulk-len` and `client-query-buffer-limit` are set high enough.

The `KNOT.UPD.*` commands are used to manipulate zone updates:

- `KNOT.UPD.BEGIN <zone> <instance>` — Initializes a zone update transaction for the specified instance. Returns a transaction ID, which is then used as a parameter for subsequent commands.
- `KNOT.UPD.ADD <zone> <txn> "data"` — Adds zone data to a zone update transaction. The data must be provided in the form of a textual zone file and may contain one or more records (including a full zone file content).
- `KNOT.UPD.REMOVE <zone> <txn> "data"` — Removes zone data from a zone update transaction. The data must be provided in the form of a textual zone file and may contain one or more records (including a full zone file content).
- `KNOT.UPD.DIFF [--compact] <zone> <txn> [owner] [type]` — Returns the contents of the specified zone update transaction, optionally in compact format and filtered by owner and record type.
- `KNOT.UPD.COMMIT <zone> <txn>` — Commits the specified zone update transaction. If the SOA serial is not incremented explicitly, it will be incremented automatically.
- `KNOT.UPD.ABORT <zone> <txn>` — Aborts the specified zone update transaction.
- `KNOT.UPD.LOAD [--compact] <zone> <instance> <serial>` — Returns the contents of the zone updates since the specified SOA serial, optionally in compact format and filtered by owner and record type.

Example of a zone update:

```
$ redis-cli KNOT.UPD.BEGIN example.com 1
(integer) 10

$ redis-cli KNOT.UPD.ADD example.com 10 "test TXT \"Knot DNS\""
OK

$ redis-cli KNOT.UPD.DIFF example.com 10
1) 1) (empty array)
   2) 1) 1) "test.example.com."
         2) "600"
         3) "TXT"
```

(continues on next page)

(continued from previous page)

```
         4) "\"Knot DNS\""

$ redis-cli KNOT.UPD.COMMIT example.com 10
OK

$ redis-cli KNOT.UPD.LOAD --compact example.com 1 1
1) 1) 1) 1) "example.com. 600 SOA ns.example.com. admin.example.com. 1 86400 900␣
↪691200 3600"
      2) 1) "example.com. 600 SOA ns.example.com. admin.example.com. 2 86400 900␣
↪691200 3600"
   2) 1) (empty array)
      2) 1) "test.example.com. 600 TXT \"Knot DNS\""
```

Warning

Do not modify the zone data using native commands such as SET or ZADD, as this may cause data corruption!

5.11 Multi-primary

In some setups, it is desirable for a secondary server to have multiple primaries configured. An example is *DNSSEC multi-signer* or any other kind of fail-proof redundancy. If the zone contents may differ among the primaries, it is necessary to avoid interchanged IXFR zone transfers at the secondary. Applying mismatched incremental changes to different zone versions can create inconsistencies that may be difficult to detect.

Knot DNS provides the following options, which can also be combined.

5.11.1 Master pinning

The *master-pin-tolerance* option, configured on the secondary server, ensures that the same primary is used unless it becomes unresponsive or remains outdated for a configured time period. Additionally, if a fallback to a different primary occurs, it enforces a full zone transfer (AXFR).

5.11.2 Serial modulo

The *serial-modulo* option, configured on the primary servers, prevents multiple primaries from using the same zone serial numbers. This ensures that even secondaries without the master pinning support will use AXFR.

5.11.3 Serial shift

Another use case for this configuration option is mostly useful with the `unixtime` *serial-policy*. It specifies that one primary is slightly "ahead" or "behind" another in zone serial numbers. This naturally enforces master pinning, even for secondaries that do not support it.

5.11.4 Disabling IXFR

As a last resort, IXFR can be disabled on the primaries using *provide-ixfr*, with clear performance drawbacks.

5.12 Zone bootstrapping on secondary

When zone refresh from the primary fails, the `retry` value from SOA is used as the interval between refresh attempts. In a case that SOA isn't known to the secondary (either because the zone hasn't been retrieved from the primary yet, or the zone has expired), a backoff is used for repeated retry attempts.

With every retry, the delay rises as a quadratic polynomial (5 * n^2, where n represents the sequence number of the retry attempt) up to two hours, each time with a random delay of 0 to 30 seconds added to spread the load on the primary. In each attempt, the retry interval is subject to *retry-min-interval* and *retry-max-interval*.

Until the refresh has been successfully completed, the backoff is restarted from the beginning by every `zone-refresh` or `zone-retransfer` of the zone triggered manually via *knotc*, by `zone-purge` or `zone-restore` of the zone's timers, or by a restart of *knotd*.

5.13 Zone expiration

On a primary, zone normally never expires. On a secondary, zone expiration results in removal of the current zone contents and a trigger of immediate zone refresh. The zone file and zone's journal are kept, but not used for answering requests until the refresh is successfully completed.

The zone expire timer is set according to the zone's SOA expire field. In addition to it, Knot DNS also supports EDNS EXPIRE extension of the expire timer in both primary and secondary roles as described in **RFC 7314**.

When Knot DNS is configured as a secondary, EDNS EXPIRE option present in a SOA, IXFR, or AFXR response from the primary is processed and used to update the zone timer when necessary. This functionality (together with requests of any other EDNS options) for a specified primary may be disabled using the *no-edns* configuration parameter.

If it's necessary, any zone may be expired manually using the `zone-purge` command of the *knotc* utility. Manual expiration is applicable to any zone, including a catalog zone or a zone on a primary. Beware, a manually expired zone on a primary or a manually expired catalog zone becomes valid again after a server configuration is reloaded or the *knotd* process is restarted, provided that the zone data hasn't been removed.

5.14 DNSSEC key states

During its lifetime, a DNSSEC key finds itself in different states. Most of the time it is used for signing the zone and published in the zone. In order to exchange the key, one type of a key rollover is necessary, and during this rollover, the key goes through various states with respect to the rollover type and also the state of the other key being rolled-over.

First, let's list the states of the key being rolled-in.

Standard states:

- `active` — The key is used for signing.
- `published` — The key is published in the zone, but not used for signing. If the key is a KSK or CSK, it is used for signing the DNSKEY RRSet.
- `ready` (only for KSK) — The key is published in the zone and used for signing. The old key is still active, since we are waiting for the DS records in the parent zone to be updated (i.e. "KSK submission").

Special states for algorithm rollover:

- `pre-active` — The key is not yet published in the zone, but it's used for signing the zone.
- `published` — The key is published in the zone, and it's still used for signing since the pre-active state.

Second, we list the states of the key being rolled-out.

Standard states:

- `retire-active` — The key is still used for signing, and is published in the zone, waiting for the updated DS records in parent zone to be acked by resolvers (KSK case) or synchronizing with KSK during algorithm rollover (ZSK case).
- `retired` — The key is no longer used for signing. If ZSK, the key is still published in the zone.
- `removed` — The key is not used in any way (in most cases such keys are deleted immediately).

Special states for algorithm rollover:

- `post-active` — The key is no longer published in the zone, but still used for signing.

Special states for **RFC 5011** trust anchor roll-over

- `revoke` (only for KSK) — The key is published and used for signing, and the Revoked flag is set.

Note

Trust anchor roll-over is not implemented with automatic key management.

The `revoke` state can only be established using *keymgr* when using *Manual key management*.

The states listed above are relevant for *keymgr* operations like generating a key, setting its timers and listing KASP database.

Note that the key "states" displayed in the server log lines while zone signing are not according to those listed above, but just a hint as to what the key is currently used for (e.g. "public, active" = key is published in the zone and used for signing).

5.15 DNSSEC key rollovers

This section describes the process of DNSSEC key rollover and its implementation in Knot DNS, and how the operator might watch and check that it's working correctly. The prerequisite is automatic zone signing with enabled *automatic key management*.

The KSK and ZSK rollovers are triggered by the respective zone key getting old according to the settings (see *KSK* and *ZSK* lifetimes).

The algorithm rollover starts when the policy *algorithm* field is updated to a different value.

The signing scheme rollover happens when the policy *signing scheme* field is changed.

It's also possible to change the algorithm and signing scheme in one rollover.

The operator may check the next rollover phase time by watching the next zone signing time, either in the log or via `knotc zone-status`. There is no special log for finishing a rollover.

Note

There are never two key rollovers running in parallel for one zone. If a rollover is triggered while another is in progress, it waits until the first one is finished. Note that a rollover might be considered finished when the old key is retired or waiting to be deleted.

The ZSK rollover is performed with Pre-publish method, KSK rollover uses Double-Signature scheme, as described in **RFC 6781**.

5.15.1 Automatic KSK and ZSK rollovers example

Let's start with the following set of keys:

```
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, key, tag 53594, algorithm ECDSAP256SHA256, KSK, public, active
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm ECDSAP256SHA256, public, active
```

The last fields hint the key state: `public` denotes a key that will be presented as the DNSKEY record, `ready` means that CDS/CDNSKEY records were created, `active` tells us that the key is used for signing, while `active+` is an active key undergoing a roll-over or roll-in.

For demonstration purposes, the following configuration is used:

```
submission:
 - id: test_submission
   check-interval: 2s
   parent: dnssec_validating_resolver
```

(continues on next page)

(continued from previous page)

```
policy:
 - id: test_policy
   ksk-lifetime: 5m
   zsk-lifetime: 2m
   propagation-delay: 2s
   dnskey-ttl: 10s
   zone-max-ttl: 15s
   ksk-submission: test_submission
```

Upon the zone's KSK lifetime expiration, a new KSK is generated and the rollover continues along the lines of **RFC 6781 Section 4.1.2**:

```
# KSK Rollover (53594 -> 3375)

2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, KSK rollover started
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, next key action, KSK tag 3375, submit at 2024-02-14T15:20:12+0100
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, key, tag 53594, algorithm ECDSAP256SHA256, KSK, public, active
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm ECDSAP256SHA256, public, active
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm ECDSAP256SHA256, KSK, public, active+
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, successfully signed, serial 2010111204, new RRSIGs 3
2024-02-14T15:20:00+0100 info: [example.com.] DNSSEC, next signing at 2024-02-14T15:20:12+0100

... (propagation-delay + dnskey-ttl) ...

2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:20:12+0100 notice: [example.com.] DNSSEC, KSK submission, waiting for confirmation
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, key, tag 53594, algorithm ECDSAP256SHA256, KSK, public, active
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm ECDSAP256SHA256, public, active
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm ECDSAP256SHA256, KSK, public, ready, active+
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, successfully signed, serial 2010111205, new RRSIGs 6
2024-02-14T15:20:12+0100 info: [example.com.] DNSSEC, next signing at 2024-02-28T15:19:37+0100
```

At this point the new KSK has to be submitted to the parent zone. Knot detects the updated parent's DS record automatically (and waits for additional period of the DS's TTL before retiring the old key) if *parent DS check* is configured, otherwise the operator must confirm it manually (using `knotc zone-ksk-submitted`)

Note

A DS record for the new KSK can be generated using:

```
$ keymgr example.com ds 3375
```

```
2024-02-14T15:20:12+0100 info: [example.com.] DS check, outgoing, remote 127.0.0.
↪1@5300 TCP, KSK submission check: negative
2024-02-14T15:20:14+0100 info: [example.com.] DS check, outgoing, remote 127.0.0.
↪1@5300 TCP/pool, KSK submission check: negative
2024-02-14T15:20:16+0100 info: [example.com.] DS check, outgoing, remote 127.0.0.
↪1@5300 TCP/pool, KSK submission check: positive
2024-02-14T15:20:16+0100 notice: [example.com.] DNSSEC, KSK submission, confirmed
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, next key action, KSK tag 53594,␣
↪remove at 2024-02-14T15:20:23+0100
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, key, tag 53594, algorithm␣
↪ECDSAP256SHA256, KSK, public, active+
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111206, new RRSIGs 2
2024-02-14T15:20:16+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:20:23+0100

... (parent's DS TTL is 7 seconds) ...

2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, next key action, ZSK, generate␣
↪at 2024-02-14T15:21:54+0100
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111207, new RRSIGs 2
2024-02-14T15:20:23+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:21:54+0100
```

Upon the zone's ZSK lifetime expiration, a new ZSK is generated and the rollover continues along the lines of **RFC 6781 Section 4.1.1**:

```
# ZSK Rollover (36185 -> 38559)

2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, ZSK rollover started
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, next key action, ZSK tag 38559,␣
↪replace at 2024-02-14T15:22:06+0100
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, key, tag 38559, algorithm␣
↪ECDSAP256SHA256, public
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111208, new RRSIGs 2
2024-02-14T15:21:54+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:22:06+0100
```

(continues on next page)

(continued from previous page)

```
... (propagation-delay + dnskey-ttl) ...

2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, next key action, ZSK tag 36185,␣
↪remove at 2024-02-14T15:22:23+0100
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, key, tag 36185, algorithm␣
↪ECDSAP256SHA256, public
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, key, tag 38559, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111209, new RRSIGs 14
2024-02-14T15:22:06+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:22:23+0100

... (propagation-delay + zone-max-ttl) ...

2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, next key action, ZSK, generate␣
↪at 2024-02-14T15:24:06+0100
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, key, tag 38559, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111210, new RRSIGs 2
2024-02-14T15:22:23+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:24:06+0100
```

Further rollovers:

```
... (zsk-lifetime - propagation-delay - zone-max-ttl) ...

# Another ZSK Rollover (38559 -> 59825)

2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, ZSK rollover started
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, next key action, ZSK tag 59825,␣
↪replace at 2024-02-14T15:24:18+0100
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, key, tag 38559, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, key, tag 59825, algorithm␣
↪ECDSAP256SHA256, public
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111211, new RRSIGs 2
2024-02-14T15:24:06+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:24:18+0100

...
```

(continues on next page)

(continued from previous page)

```
# Another KSK Rollover (3375 -> 50822)

2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, signing zone
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, KSK rollover started
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, next key action, KSK tag 50822,␣
↪submit at 2024-02-14T15:25:12+0100
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, key, tag  3375, algorithm␣
↪ECDSAP256SHA256, KSK, public, active
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, key, tag 59825, algorithm␣
↪ECDSAP256SHA256, public, active
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, key, tag 50822, algorithm␣
↪ECDSAP256SHA256, KSK, public, active+
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, signing started
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, successfully signed, serial␣
↪2010111214, new RRSIGs 3
2024-02-14T15:25:00+0100 info: [example.com.] DNSSEC, next signing at 2024-02-
↪14T15:25:12+0100

...
```

Tip

If systemd is available, the KSK submission event is logged into journald in a structured way. The intended use case is to trigger a user-created script. Example:

```
journalctl -f -t knotd -o json | python3 -c '
import json, sys
for line in sys.stdin:
  k = json.loads(line);
  if "KEY_SUBMISSION" in k:
    print("%s, zone=%s, keytag=%s" % (k["__REALTIME_TIMESTAMP"], k["ZONE"], k["KEY_
↪SUBMISSION"]))
'
```

Alternatively, the *D-Bus signaling* can be utilized for the same use.

5.16 DNSSEC shared KSK

Knot DNS allows, with automatic DNSSEC key management, to configure a shared KSK for multiple zones. By enabling *ksk-shared*, we tell Knot to share all newly-created KSKs among all the zones with the same *DNSSEC signing policy* assigned.

The feature works as follows. Each zone still manages its keys separately. If a new KSK shall be generated for the zone, it first checks if it can grab another zone's shared KSK instead - that is the last generated KSK in any of the zones with the same policy assigned. Anyway, only the cryptographic material is shared, the key may have different timers in each zone.

Consequences:

If we have an initial setting with brand new zones without any DNSSEC keys, the initial keys for all zones are generated. With shared KSK, they will all have the same KSK, but different ZSKs. The KSK rollovers may take place at slightly different times for each of the zones, but the resulting new KSK will be shared again among all of them.

If we have zones which already have their keys, turning on the shared KSK feature triggers no action. But when a KSK rollover takes place, they will use the same new key afterwards.

Warning

Changing the policy *id* must be done carefully if shared KSK is in use.

5.17 DNSSEC delete algorithm

This is how to "disconnect" a signed zone from a DNSSEC-aware parent zone. More precisely, we tell the parent zone to remove our zone's DS record by publishing a special formatted CDNSKEY and CDS record. This is mostly useful if we want to turn off DNSSEC on our zone so it becomes insecure, but not bogus.

With automatic DNSSEC signing and key management by Knot, this is as easy as configuring *cds-cdnskey-publish* option and reloading the configuration. We check if the special CDNSKEY and CDS records with the rdata "0 3 0 AA==" and "0 0 0 00", respectively, appeared in the zone.

After the parent zone notices and reflects the change, we wait for TTL expire (so all resolvers' caches get updated), and finally we may do anything with the zone, e.g. turning off DNSSEC, removing all the keys and signatures as desired.

5.18 DNSSEC Offline KSK

Knot DNS allows a special mode of operation where the private part of the Key Signing Key is not available to the daemon, but it is rather stored securely in an offline storage. This requires that the KSK/ZSK signing scheme is used (i.e. *single-type-signing* is off). The Zone Signing Key is always fully available to the daemon in order to sign common changes to the zone contents.

The server (or the "ZSK side") only uses ZSK to sign zone contents and its changes. Before performing a ZSK rollover, the DNSKEY records will be pre-generated and signed by the signer (the "KSK side"). Both sides exchange keys in the form of human-readable messages with the help of the *keymgr* utility.

5.18.1 Prerequisites

For the ZSK side (i.e. the operator of the DNS server), the zone has to be configured with:

- Enabled *DNSSEC signing*
- Properly configured and assigned *DNSSEC policy*:
  - Enabled *manual*
  - Enabled *offline-ksk*
  - Explicit *dnskey-ttl*
  - Explicit *zone-max-ttl*
  - Recommended *keytag-modulo* setting to `0/2` to prevent keytag conflicts
  - Other options are optional
- KASP DB may contain a ZSK (the present or some previous one(s))

For the KSK side (i.e. the operator of the KSK signer), the zone has to be configured with:

- Properly configured and assigned *DNSSEC policy*:
  - Enabled *manual*
  - Enabled *offline-ksk*
  - Explicit *rrsig-refresh*
  - Recommended *keytag-modulo* setting to `1/2` to prevent keytag conflicts
  - Optional *rrsig-lifetime*, *rrsig-pre-refresh*, *algorithm*, *reproducible-signing*, and *cds-cdnskey-publish*
  - Other options are ignored

- KASP DB contains a KSK (the present or a newly generated one)

5.18.2 Generating and signing future ZSKs

1. Use the `keymgr pregenerate` command on the ZSK side to prepare the ZSKs for a specified period of time in the future. The following example generates ZSKs for the *example.com* zone for 6 months ahead starting from now:

   ```
   $ keymgr -c /path/to/ZSK/side.conf example.com. pregenerate +6mo
   ```

   If the time period is selected as e.g. *2 x zsk-lifetime + 4 x propagation-delay*, it will prepare roughly two complete future key rollovers. The newly-generated ZSKs remain in non-published state until their rollover starts, i.e. the time they would be generated in case of automatic key management.

2. Use the `keymgr generate-ksr` command on the ZSK side to export the public parts of the future ZSKs in a form similar to DNSKEY records. You might use the same time period as in the first step:

   ```
   $ keymgr -c /path/to/ZSK/side.conf example.com. generate-ksr +0 +6mo > /path/to/
   ↪ksr/file
   ```

   Save the output of the command (called the Key Signing Request or KSR) to a file and transfer it to the KSK side e.g. via e-mail.

3. Use the `keymgr sign-ksr` command on the KSK side with the KSR file from the previous step as a parameter:

   ```
   $ keymgr -c /path/to/KSK/side.conf example.com. sign-ksr /path/to/ksr/file > /
   ↪path/to/skr/file
   ```

   This creates all the future forms of the DNSKEY, CDNSKEY and CSK records and all the respective RRSIGs and prints them on output. Save the output of the command (called the Signed Key Response or SKR) to a file and transfer it back to the ZSK side.

4. Use the `keymgr import-skr` command to import the records and signatures from the SKR file generated in the last step into the KASP DB on the ZSK side:

   ```
   $ keymgr -c /path/to/ZSK/side.conf example.com. import-skr /path/to/skr/file
   ```

5. Use the `knotc zone-keys-load` command to trigger a zone re-sign on the ZSK-side and set up the future re-signing events correctly.:

   ```
   $ knotc -c /path/to/ZSK/side.conf zone-keys-load example.com.
   ```

6. Now the future ZSKs and DNSKEY records with signatures are ready in KASP DB for later usage. Knot automatically uses them at the correct time intervals. The entire procedure must be repeated before the time period selected at the beginning passes, or whenever a configuration is changed significantly. Importing new SKR over some previously-imported one leads to deleting the old offline records.

5.18.3 Offline KSK and manual ZSK management

If the automatically preplanned ZSK roll-overs (first step) are not desired, just set the *zsk-lifetime* to zero, and manually pregenerate ZSK keys and set their timers. Then follow the steps `generate-ksr - sign-ksr - import-skr - zone-keys-load` and repeat the ceremony when necessary.

5.18.4 Offline KSK roll-over

The KSKs (on the KSK side) must be managed manually, but manual KSK roll-over is possible. Just plan the steps of the KSK roll-over in advance, and whenever the KSK set or timers are changed, re-perform the relevant rest of the ceremony `sign-ksr - import-skr - zone-keys-load`.

5.18.5 Emergency SKR

A general recommendation for large deployments is to have some backup pre-published keys, so that if the current ones are compromised, they can be rolled-over to the backup ones without any delay. But in the case of Offline KSK, according to the procedures above, both ZSK and KSK immediate rollovers require the KSR-SKR ceremony.

However, a trick can be done to achieve really immediate key substitution. This is no longer about Knot DNS functionality, just a hint for the operator.

The idea is to perform every KSR-SKR ceremony twice: once with normal state of the keys (the backup key is only published), and once with the keys already exchanged (the backup key is temporarily marked as active and the standard key temporarily as public only). The second (backup) SKR should be saved for emergency key replacement.

Summary of the steps:

- Prepare KSK and ZSK side as usual, including public-only emergency key
- Perform normal Offline KSK ceremony:
  - Pre-generate ZSKs (only in the case of automatic ZSK management)
  - Generate KSR
  - Sign KSR on the KSK side
  - Import SKR
  - Re-sign the zone
- Freeze the zone on the ZSK side
- Temporarily set the backup key as active and the normal key as publish-only
- Perform backup Offline KSK ceremony:
  - Generate KSR (only if the backup key is a replacement for ZSK)
  - Sign the KSR on the KSK side
  - Save the SKR to a backup storage, don't import it yet
- Return the keys to the previous state
- Thaw the zone on the ZSK side

Emergency key replacement:

- Import the backup SKR
- Align the keys with the new states (backup key as active, compromised key as public)
- Re-sign the zone

5.19 DNSSEC multi-signer

Multi-signer is a general term that refers to any mode of operation in which a DNS zone is signed by multiple servers (usually two) in parallel. Knot DNS offers various multi-signer modes, which are recommended for redundancy within an organization. For multi-signer operations involving multiple "DNSSEC providers" and the ability to switch between them, you can also refer to MUSIC.

Regardless of the chosen mode from the following options, any secondary that has multiple signers configured as primaries must prevent interchanged IXFR from them. See the relevant *Multi-primary* capter on how to achieve it.

In order to prevent keytag conflicts, it is recommended that the keytags of keys generated by each signer are from distinct subset of possible values. With Knot DNS, this can be achieved using *keytag-modulo* option (e.g. for three signers, setting `0/3` on the first one, `1/3` on the second, and `2/3` on the third of them).

5.19.1 Sharing private keys, manual policy

When DNSSEC keys are shared among zone signing servers (signers), one challenge is automatic key management (roll-overs) and synchronization among the signers. In this example mode of operation, it is expected that key management is maintained outside of Knot, and the generated keys, including private keys and metadata (timers), are available in Bind9 format.

Every new key is then imported into each Knot using the *keymgr* `import-bind` command, after which *knotc* `zone-keys-load` is invoked. With *manual* policy configured, the signers simply follow prescribed key timers, maintaining the same key set at each signer. For more useful commands like `list`, `set`, and `delete`, refer to *keymgr*.

5.19.2 Sharing private keys, automatic policy

Knot handles automatic key management very well, but enabling it on multiple instances would lead to redundant key generation. However, it's possible to enable it on one signer and keep synchronizing the keys to all others. The managing signer shall be configured with *automatic ZSK/KSK management*, all others with *manual* policy.

The key set changes on the managing signer can be monitored by periodic queries with *keymgr* `list`, or by listening to *D-Bus* interface and watching for the `keys_updated` event.

Whenever the key set is changed, key synchronization can be safely performed with *Data and metadata backup* feature. Dump the KASP database on the managing signer with *knotc* `zone-backup +kaspdb`, transfer the backup directory to each other signer, and import the keys by *knotc* `zone-restore +kaspdb`, followed by `zone-keys-load` on them.

This way, the full key set, including private keys and all metadata, is always synchronized between signers. The method of transporting the backup directory is beyond the scope of Knot and this documentation. An eventual loss of the managing signer results in the automatic key management being halted, but DNSSEC signing continues to function. The synchronization delay for keys between the managing signer and other signers must be accounted for in *propagation-delay*.

5.19.3 Distinct keys, DNSKEY record synchronization

When the DNSSEC keys are not shared among signers, each server can manage its own keys separately. However, the DNSKEY (including CDNSKEY and CDS) records (with public keys) must be synchronized for full validity of the signed zone. *Dynamic updates* can be used to achieve this sharing.

The following configuration options should be used:

- Set *dnskey-management* to `incremental` on each signer to ensure it retains the other's DNSKEY records in the zone during signing.
- Set *delete-delay* to a reasonable time interval, which ensures that all signers get synchronized during this period.
- Set *cds-cdnskey-publish* to either `none` or `always`, otherwise the parent DS record might configure itself to point only to one signer's KSK.
- Configure *dnskey-sync* to all other signers so that this signer's public keys appear in each other's DNSKEY (also applies to CDNSKEY and CDS) RRSets.
- Configure *Access control list (ACL)* so that DDNS from all other signers is allowed.
- Set *ddns-master* to empty value ("") so that DDNS from other signers is not forwarded to the primary master if any.
- Additionally, the synchronization delay between all signers must be accounted for in *propagation-delay*.

With careful configuration, all signers automatically synchronize their DNSKEY (and eventually CDNSKEY and CDS) RRSets, keeping them synchronized during roll-overs. Nevertheless, it is recommended to monitor their logs.

An illustrative example of the second of three signers:

```
remote:
    - id: signer1
      address: 10.20.30.1
    - id: signer3
      address: 10.20.30.3

 acl:
    - id: signers
      remote: [ signer1, signer3 ]
      action: [ query, update ]
      # TODO configure TSIGs!

dnskey-sync:
    - id: sync
      remote: [ signer3, signer1 ]

policy:
    - id: multisigner
      single-type-signing: on
      ksk-lifetime: 60d
      ksk-submission: ... # TODO see Automatic KSK management
      propagation-delay: 14h
      delete-delay: 2h
      cds-cdnskey-publish: always
      dnskey-management: incremental
      dnskey-sync: sync

zone:
    - domain: example.com.
      # TODO configure zonefile and journal
      # TODO configure transfers in/out: master, NOTIFY, ACLs...
      dnssec-signing: on
      dnssec-policy: multisigner
      ddns-master: ""
      serial-modulo: 1/3
      acl: signers
```

5.19.4 Distinct keys, DNSKEY at common unsigned primary

The same approach and configuration can be used, with the difference that the signers do not send updated DNSKEYs (along with CDNSKEYs and CDSs) to each other. Instead, they send the updates to their common primary, which holds the unsigned version of zone. The only configuration change involves redirecting *dnskey-sync* to the common primary and adjusting its ACL to allow DDNS from the signers.

It is also necessary to configure *ixfr-benevolent* on each signer so that they accept incremental zone transfers from the primary with additions (or removals) of their own's DNSKEYs.

This setup should work nicely with any number of signers, however, due to the size of DNSKEY RRSet, at most three are advisable.

5.20 DNSSEC keys import to HSM

Knot DNS stores DNSSEC keys in textual PEM format (**RFC 7468**), while many HSM management software require the keys for import to be in binary DER format (Rec. ITU-T X.690). Keys can be converted from one format to another by software tools such as `certtool` from GnuTLS suite or `openssl` from OpenSSL suite.

In the examples below, `c4eae5dea3ee8c15395680085c515f2ad41941b6` is used as the key ID, `c4eae5dea3ee8c15395680085c515f2ad41941b6.pem` represents the filename of the key in PEM format as copied from the Knot DNS zone's *KASP database directory*, `c4eae5dea3ee8c15395680085c515f2ad41941b6.priv.der` represents the file containing the private key in DER format as generated by the conversion tool, and `c4eae5dea3ee8c15395680085c515f2ad41941b6.pub.der` represents the file containing the public key in DER format as generated by the conversion tool.

```
$ certtool -V -k --outder --infile c4eae5dea3ee8c15395680085c515f2ad41941b6.pem \
  --outfile c4eae5dea3ee8c15395680085c515f2ad41941b6.priv.der

$ certtool -V --pubkey-info --outder --load-privkey␣
↪c4eae5dea3ee8c15395680085c515f2ad41941b6.pem \
  --outfile c4eae5dea3ee8c15395680085c515f2ad41941b6.pub.der
```

As an alternative, `openssl` can be used instead. It is necessary to specify either `rsa` or `ec` command according to the algorithm used by the key.

```
$ openssl rsa -outform DER -in c4eae5dea3ee8c15395680085c515f2ad41941b6.pem \
  -out c4eae5dea3ee8c15395680085c515f2ad41941b6.priv.der

$ openssl rsa -outform DER -in c4eae5dea3ee8c15395680085c515f2ad41941b6.pem \
  -out c4eae5dea3ee8c15395680085c515f2ad41941b6.pub.der -pubout
```

Actual import of keys (both public and private keys from the same key pair) to an HSM can be done via PKCS #11 interface, by `pkcs11-tool` from OpenSC toolkit for example. In the example below, `/usr/local/lib/pkcs11.so` is used as a name of the PKCS #11 library or module used for communication with the HSM.

```
$ pkcs11-tool --module /usr/local/lib/pkcs11.so --login \
  --write-object c4eae5dea3ee8c15395680085c515f2ad41941b6.priv.der --type privkey \
  --usage-sign --id c4eae5dea3ee8c15395680085c515f2ad41941b6

$ pkcs11-tool --module /usr/local/lib/pkcs11.so -login \
  --write-object c4eae5dea3ee8c15395680085c515f2ad41941b6.pub.der --type pubkey \
  --usage-sign --id c4eae5dea3ee8c15395680085c515f2ad41941b6
```

5.21 DNSSEC multiple keystores

Private keys for signing a single zone can be stored in more than one keystore with two main use-cases:

5.21.1 Keystore migration

Some keystores (HSMs) do not allow exporting private key material, so it is not possible to migrate away from them by copying the private keys to a new keystore (another HSM or a directory with PEM files). In such cases, the migration must be performed by key rollover(s), where new keys are generated in the target keystore, while the old keys remain in the original one and are eventually retired and removed.

During the transition period (i.e. rollovers), both keystores must be configured simultaneously. This is achieved by configuring them both in the *keystore section* and referencing them in the *keystore* option in *policy section*. The order matters: new keys are generated in the first referenced keystore, so in this case, the target keystore must be listed first.

5.21.2 KSK more secured than ZSK

When desirable, the Key Signing Key (KSK) can be stored in an HSM, while the Zone Signing Key (ZSK) is kept in a simple PEM file. This enhances the security of the KSK while preserving zone signing performance (as HSM signing operations are usually much slower).

To achieve this, both keystores must be configured in the *keystore section*, while the PKCS#11 keystore (the HSM) has the *ksk-only* option enabled and being listed first in the *keystore* option in *policy section*. With this configuration, new KSKs are generated in the HSM, while ZSKs are generated in the second keystore.

5.22 Daemon controls

Knot DNS was designed to allow server reconfiguration on-the-fly without interrupting its operation. Thus it is possible to change both configuration and zone files and also add or remove zones without restarting the server. This can be done with:

```
$ knotc reload
```

If you want to refresh the secondary zones, you can do this with:

```
$ knotc zone-refresh
```

5.23 Logging

Knot DNS supports *logging* to *syslog* or *systemd-journald* facility, to a specified file, to standard output, or to standard error output. Several different logging targets may be used in parallel.

If *syslog* or *systemd-journald* is used for logging, log rotation is handled by that logging facility. When logging to a specified file, log rotation should be done by moving the current log file followed by reopening of the log file with either `knotc -b reload` or by sending `SIGHUP` to the `knotd` process (see the *pidfile*).

5.24 External validation

This feature enables automatic validation of every zone update using third-party zone-checking tools, as well as the enforcement of user-defined policies on zone contents and updates (for example, ensuring that an update does not modify too many records of specific RRtypes).

External validation can be combined with, but does not depend on, *dnssec-validation* and *zonemd-verify*. It behaves similarly in that, if validation fails, the update is discarded and the server continues responding (including outgoing zone transfers) from the last known good version of the zone.

Specifically, once *external-validation* is configured, any time a zone is being modified, the update process is paused just before publishing the new version. At this point, it waits for verification and either confirmation or rejection. The user or script can retrieve the status via `knotc zone-status` or will be notified via *dbus-event*. A response should be issued by calling `knotc zone-commit` to approve the update or `knotc zone-abort` to reject it. As an alternative to using *knotc*, the command can be sent directly to the server's control socket.

Note that the background worker processing the update is blocked during the external validation and is not able to handle updates to other zones at that time. Therefore, it is recommended to finish the validation as fast as possible and/or configure a sufficient number of *background-workers* relative to the number of zones.

Additional options can be configured in the *external section*. This includes settings to dump the full new zone and/or the zone differences to specified files in textual (zone file) format before external validation begins.

Also note that external validation may be rejected automatically (without an explicit call to `knotc zone-abort`) due to *timeout*. This mechanism prevents the server from becoming locked.

See a simple demonstration script. showing how to implement external validation using the combined D-Bus and knotc interface.

5.25 Data and metadata backup

Some of the zone-related data, such as zone contents or DNSSEC signing keys, and metadata, like zone timers, might be worth backing up. For the sake of consistency, it's usually necessary to shut down the server, or at least freeze all the zones, before copying the data like zone files, KASP database, etc, to a backup location. To avoid this necessity, Knot DNS provides a feature to back up some or all of the zones seamlessly.

5.25.1 Online backup

While the server is running and the zones normally loaded (even when they are constantly/frequently being updated), the user can manually trigger the backup by calling:

```
$ knotc zone-backup +backupdir /path/of/backup
```

To back up just some of the zones (instead of all), the user might provide their list:

```
$ knotc zone-backup +backupdir /path/to/backup zone1.com. zone2.com. ...
```

The backup directory should be empty or non-existing and it must be accessible and writable for the *user* account under which knotd is running. The backup procedure will begin soon and will happen zone-by-zone (partially in parallel if more *background-workers* are configured). **The user shall check the logs for the outcome of each zone's backup attempt.** The knotc's `-b` parameter might be used if the user desires to wait until the backup work is done and a simple result status is printed out.

Tip

There is a plain ASCII text file in the backup directory, `knot_backup.label`, that contains some useful information about the backup, such as the backup creation date & time, the server identity, etc. Care must always be taken **not to remove this file** from the backup nor to damage it.

If a backup fails, the backup directory containing incomplete backup is retained. For repeated backup attempts to the same directory, it must be removed or renamed manually first, or the force option may be used in a repeated backup.

Note

When backing up or restoring a catalog zone, it's necessary to make sure that the contents of the catalog doesn't change during the backup or restore. An easy solution is to use `knotc zone-freeze` on the catalog zone for the time of backup and restore.

5.25.2 Offline restore

If the Online backup was performed for all zones, it's possible to restore the backed up data by simply copying them to their normal locations, since they're simply copies. For example, the user can copy (overwrite) the backed up KASP database files to their configured location.

This restore of course must be done when the server is stopped. After starting up the server, it should run in the same state as at the time of backup.

This method is recommended in the case of complete data loss, for example physical server failure.

Note

The online backup procedure stores all zone files in a single directory using their default file names. If the original directory layout was different, then the required directory structure must be created manually for offline restore and zone files must be placed individually to their respective directories. If the zone file names don't follow the default pattern, they must be renamed manually to match the configuration. These limitations don't apply to the online restore procedure.

5.25.3 Online restore

This procedure is symmetrical to Online backup. By calling:

```
$ knotc zone-restore +backupdir /path/of/backup
```

the user triggers a one-by-one zone restore from the backup on a running server. Again, a subset of zones might be specified. It must be specified if the backup was created for only a subset of zones.

Note

For restore of backups that have been created by Knot DNS releases prior to 3.1, it's necessary to use the `-f` option. Since this option also turns off some verification checks, it shouldn't be used in other cases.

Note

For QUIC/TLS, only the auto-generated key is restored. The `zone-restore` command doesn't restore a user-defined QUIC/TLS key and certificate so as to avoid possible configuration management conflicts and they must be restored from the backup (its subdirectory `quic`) manually. In all cases, restart of the Knot server after the restore is necessary for the restored QUIC/TLS key/certificate to take effect.

5.25.4 Limitations

Neither configuration file nor *Configuration database* is backed up by zone backup. The configuration has to be synchronized before zone restore is performed!

If the private keys are stored in a HSM (anything using a PKCS#11 interface), they are not backed up. This includes internal metadata of the PKCS#11 provider software, such as key mappings, authentication information, and the configuration of the provider. Details are vendor-specific.

The restore procedure does not care for keys deleted after taking the snapshot. Thus, after restore, there might remain some redundant `.pem` files of obsolete signing keys.

Tip

In order to seamlessly deploy a restored backup of KASP DB with respect to a possibly ongoing DNSSEC key rollover, it's recommended to set *propagation-delay* to the sum of:

- The maximum delay between beginning of the zone signing and publishing re-signed zone on all public secondary servers.
- How long it takes for the backup server to start up with the restored data.
- The period between taking backup snapshots of the live environment.

5.26 Statistics

The server provides some general statistics and optional query module statistics (see *mod-stats*).

Server statistics or global module statistics can be shown by:

```
$ knotc stats
$ knotc stats server              # Show all server counters
$ knotc stats mod-stats           # Show all mod-stats counters
$ knotc stats server.zone-count  # Show specific server counter
```

Per zone statistics can be shown by:

```
$ knotc zone-stats example.com.                     # Show all zone counters
$ knotc zone-stats example.com. mod-stats           # Show all zone mod-stats␣
```

(continues on next page)

(continued from previous page)

```
↪counters
$ knotc zone-stats example.com. mod-stats.query-type  # Show specific zone counter
$ knotc zone-stats --                                 # Show all zone counters for␣
↪all zones
$ knotc zone-stats -- mod-stats.request-protocol      # Show specific zone counter␣
↪for all zones
```

To show all supported counters even with 0 value, use the force option.

A simple periodic statistic dump to a YAML file can also be enabled. See *statistics section* for the configuration details.

As the statistics data can be accessed over the server control socket, it is possible to create an arbitrary script (Python is supported at the moment) which could, for example, publish the data in JSON format via HTTP(S) or upload the data to a more efficient time series database. Take a look into the python folder of the project for these scripts.

5.27 Mode XDP

Thanks to recent Linux kernel capabilities, namely eXpress Data Path and AF_XDP address family, Knot DNS offers a high-performance packet processing mode for UDP, TCP, and QUIC protocols. The basic idea is to filter DNS messages close to the network device and effectively forward them to the nameserver without touching the network stack of the operating system. Messages in protocols, for which XDP hasn't been enbled, are processed as usual.

If *listen* is configured, the server creates additional XDP workers, listening on specified interface(s) and port(s) for DNS queries. Each XDP worker handles one RX and TX network queue pair.

5.27.1 Pre-requisites

- Linux kernel 4.18+ (5.x+ is recommended for optimal performance) compiled with the *CONFIG_XDP_SOCKETS=y* option. The XDP mode isn't supported in other operating systems.
- A multiqueue network card, which offers enough Combined RX/TX channels, with native XDP support is highly recommended. Successfully tested cards:
  - NVIDIA (Mellanox) ConnectX-6 Dx (driver *mlx5_core*), the maximum number of channels per interface is 127.
  - Intel series E810 (driver *ice*), the maximum number of channels per interface must be limited to 127 or fewer. Linux kernel 6.10.4 or later, along with its driver, is required for stability.
  - Intel series 700 (driver *i40e*), the maximum number of channels per interface is 64. Linux kernel drivers are recommended.

  Cards with known instability issues:

  - Intel series 500 (driver *ixgbe*).
- If the *knotd* service is not directly executed in the privileged mode, some additional Linux capabilities have to be set:

  Execute command:

  ```
  systemctl edit knot
  ```

  And insert these lines:

  ```
  [Service]
  CapabilityBoundingSet=CAP_NET_RAW CAP_NET_ADMIN CAP_SYS_ADMIN CAP_IPC_LOCK CAP_
  ↪SYS_RESOURCE
  ```

(continues on next page)

(continued from previous page)

```
AmbientCapabilities=CAP_NET_RAW CAP_NET_ADMIN CAP_SYS_ADMIN CAP_IPC_LOCK CAP_SYS_
↪RESOURCE
```

The *CAP_SYS_RESOURCE* is needed on Linux < 5.11.

All the capabilities are dropped upon the service is started.

- For proper processing of VLAN traffic, VLAN offloading should be disabled. E.g.:

```
ethtool -K <interface> tx-vlan-offload off rx-vlan-offload off
```

5.27.2 Optimizations

Some helpful commands:

```
ethtool -N <interface> rx-flow-hash udp4 sdfn
ethtool -N <interface> rx-flow-hash udp6 sdfn
ethtool -L <interface> combined <?>
ethtool -G <interface> rx <?> tx <?>
renice -n 19 -p $(pgrep '^ksoftirqd/[0-9]*$')
```

5.27.3 Limitations

- Request and its response must go through the same physical network device.
- Dynamic DNS over XDP is not supported.
- MTU higher than 1790 bytes is not supported.
- Multiple BPF filters per one network device are not supported.
- Systems with big-endian byte ordering require special recompilation of the nameserver.
- IPv4 header and UDP checksums are not verified on received DNS messages.
- DNS over XDP traffic is not visible to common system tools (e.g. firewall, tcpdump etc.).
- BPF filter is not automatically unloaded from the network device. Manual filter unload:

```
ip link set dev <interface> xdp off
```

CHAPTER

SIX

TROUBLESHOOTING

First of all, check the logs. Enabling at least the `warning` message severity may help you to identify some problems. See the *log section* for details.

6.1 Reporting bugs

If you are unable to solve the problem by yourself, you can submit a bugreport to the Knot DNS developers. For security or sensitive issues contact the developers directly on knot-dns@labs.nic.cz. All other bugs and questions may be directed to the public Knot DNS users mailing list (knot-dns-users@lists.nic.cz) or may be entered into the issue tracking system.

Before anything else, please try to answer the following questions:

- Has it been working?
- What has changed? System configuration, software updates, network configuration, firewall rules modification, hardware replacement, etc.

The bugreport should contain the answers for the previous questions and in addition at least the following information:

- Knot DNS version and type of installation (distribution package, from source, etc.)
- Operating system, platform, kernel version
- Relevant basic hardware information (processor, amount of memory, available network devices, etc.)
- Description of the bug
- Log output with the highest verbosity (category `any`, severity `debug`)
- Steps to reproduce the bug (if known)
- Backtrace (if the bug caused a crash or a hang; see the next section)

If possible, please provide a minimal configuration file and zone files which can be used to reproduce the bug.

6.2 Generating backtrace

Backtrace carries basic information about the state of the program and how the program got where it is. It helps determining the location of the bug in the source code.

If you run Knot DNS from distribution packages, make sure the debugging symbols for the package are installed. The symbols are usually distributed in a separate package.

There are several ways to get the backtrace. One possible way is to extract the backtrace from a core dump file. Core dump is a memory snapshot generated by the operating system when a process crashes. The generating of core dumps must be usually enabled:

```
$ ulimit -c unlimited                  # Enable unlimited core dump size
$ knotd ...                            # Reproduce the crash
...
$ gdb knotd <core-dump-file>           # Start gdb on the core dump
(gdb) info threads                     # Get a summary of all threads
(gdb) thread apply all bt full         # Extract backtrace from all threads
(gdb) quit
```

To save the backtrace into a file, the following GDB commands can be used:

```
(gdb) set pagination off
(gdb) set logging file backtrace.txt
(gdb) set logging on
(gdb) info threads
(gdb) thread apply all bt full
(gdb) set logging off
```

To generate a core dump of a running process, the *gcore* utility can be used:

```
$ gcore -o <output-file> $(pidof knotd)
```

Please note that core dumps can be intercepted by an error-collecting system service (systemd-coredump, ABRT, Apport, etc.). If you are using such a service, consult its documentation about core dump retrieval.

If the error is reproducible, it is also possible to start and inspect the server directly in the debugger:

```
$ gdb --args knotd -c /etc/knot.conf
(gdb) run
...
```

Alternatively, the debugger can be attached to a running server process. This is generally useful when troubleshooting a stuck process:

```
$ knotd ...
$ gdb --pid $(pidof knotd)
(gdb) continue
...
```

If you fail to get a backtrace of a running process using the previous method, you may try the single-purpose `pstack` utility:

```
$ pstack $(pidof knotd) > backtrace.txt
```

6.3 Crash caused by a Bus error

Zone files and a configuration file are usually accessed as mmaped files. If such files are changed or truncated at the same time when those files are being loaded/reloaded by the program, it may result in Bus error (SIGBUS) and a program crash. If you encounter a Bus error, first check that there isn't a concurrent write access from an external program to the respective files.

CHAPTER

SEVEN

CONFIGURATION REFERENCE

7.1 Description

Configuration files for Knot DNS use simplified YAML format. Simplified means that not all of the features are supported.

For the description of configuration items, we have to declare a meaning of the following symbols:

- `INT` – Integer
- `STR` – Textual string
- `HEXSTR` – Hexadecimal string (with `0x` prefix)
- `BOOL` – Boolean value (`on`/`off` or `true`/`false`)
- `TIME` – Number of seconds, an integer with a possible time multiplier suffix (`s` ~ 1, `m` ~ 60, `h` ~ 3600, `d` ~ 24 * 3600, `w` ~ 7 * 24 * 3600, `M` ~ 30 * 24 * 3600, `y` ~ 365 * 24 * 3600)
- `SIZE` – Number of bytes, an integer with a possible size multiplier suffix (`B` ~ 1, `K` ~ 1024, `M` ~ 1024^2 or `G` ~ 1024^3)
- `BASE64` – Base64 encoded string
- `ADDR` – IPv4 or IPv6 address
- `DNAME` – Domain name
- `...` – Multi-valued item, order of the values is preserved
- `[ ]` – Optional value
- `|` – Choice

The configuration consists of several fixed sections and optional module sections. There are 18 fixed sections (`module`, `server`, `xdp`, `control`, `log`, `statistics`, `database`, `keystore`, `key`, `remote`, `remotes`, `acl`, `submission`, `dnskey-sync`, `policy`, `external``, `template`, `zone`). Module sections are prefixed with the `mod-` prefix (e.g. `mod-stats`).

Most of the sections (e.g. `zone`) are sequences of settings blocks. Each settings block begins with a unique identifier, which can be used as a reference from other sections (such an identifier must be defined in advance).

A multi-valued item can be specified either as a YAML sequence:

```
address: [10.0.0.1, 10.0.0.2]
```

or as more single-valued items each on an extra line:

```
address: 10.0.0.1
address: 10.0.0.2
```

If an item value contains spaces or other special characters, it is necessary to enclose such a value within double quotes " ".

If not specified otherwise, an item representing a file or a directory path may be defined either as an absolute path (starting with `/`), or a path relative to the same directory as the default value of the item.

7.2 Comments

A comment begins with a # character and is ignored during processing. Also each configuration section or sequence block allows a permanent comment using the `comment` item which is stored in the server beside the configuration.

7.3 Including configuration

Another configuration file or files, matching a pattern, can be included at the top level in the current file.

```
include: STR
```

7.3.1 include

A path or a matching pattern specifying one or more files that are included at the place of the include option position in the configuration. If the path is not absolute, then it is considered to be relative to the current file. The pattern can be an arbitrary string meeting POSIX *glob* requirements, e.g. dir/*.conf. Matching files are processed in sorted order.

Default: not set

7.4 Clearing configuration sections

It's possible to clear specified configuration sections at given phases of the configuration parsing.

```
clear: STR
```

7.4.1 clear

A matching pattern specifying configuration sections that are cleared when this item is parsed. This allows overriding of existing configuration in the configuration database when including a configuration file or ensures that some configuration wasn't specified in previous includes.

Note

For the pattern matching the POSIX function fnmatch() is used. On Linux, the GNU extension FNM_EXTMATCH is enabled, which allows extended pattern matching. Examples:

- `clear: zone` – Clears the `zone` section.
- `clear: mod-*` – Clears all module sections.
- `clear: "[!z]*"` – Clears all sections not beginning with letter `z`.
- `clear: !(zone)` – (GNU only) Clears all sections except the `zone` one.
- `clear: @(zone|template)` – (GNU only) Clears the `zone` and `template` sections.

Default: not set

7.5 `module` section

Dynamic modules loading configuration.

Note

If configured with non-empty `--with-moduledir=path` parameter, all shared modules in this directory will be automatically loaded.

```
module:
  - id: STR
    file: STR
```

7.5.1 id

A module identifier in the form of the `mod-` prefix and module name suffix.

7.5.2 file

A path to a shared library file with the module implementation.

Warning

If the path is not absolute, the library is searched in the set of system directories. See `man dlopen` for more details.

Default: `${libdir}/knot/modules-${version}`/module_name.so (or `${path}`/module_name.so if configured with `--with-moduledir=path`)

7.6 server section

General options related to the server.

```
server:
    identity: [STR]
    version: [STR]
    nsid: [STR|HEXSTR]
    rundir: STR
    user: STR[:STR]
    pidfile: STR
    udp-workers: INT
    tcp-workers: INT
    background-workers: INT
    async-start: BOOL
    tcp-idle-timeout: TIME
    tcp-io-timeout: INT
    tcp-remote-io-timeout: INT
    tcp-max-clients: INT
    tcp-reuseport: BOOL
    tcp-fastopen: BOOL
    quic-max-clients: INT
    quic-outbuf-max-size: SIZE
    quic-idle-close-timeout: TIME
    remote-pool-limit: INT
    remote-pool-timeout: TIME
    remote-retry-delay: INT
    socket-affinity: BOOL
    udp-max-payload: SIZE
    udp-max-payload-ipv4: SIZE
```

(continues on next page)

(continued from previous page)

```
    udp-max-payload-ipv6: SIZE
    key-file: STR
    cert-file: STR
    ca-file: STR ...
    edns-client-subnet: BOOL
    answer-rotation: BOOL
    automatic-acl: BOOL
    proxy-allowlist: ADDR[/INT] | ADDR-ADDR ...
    dbus-event: none | running | zone-updated | external-verify | ksk-submission |␣
↪dnssec-invalid ...
    dbus-init-delay: TIME
    listen: ADDR[@INT] | STR ...
    listen-quic: ADDR[@INT] ...
    listen-tls: ADDR[@INT] ...
```

Caution

When you change configuration parameters dynamically or via configuration file reload, some parameters in the Server section require restarting the Knot server so that the changes take effect. See below for the details.

7.6.1 identity

An identity of the server returned in the response to the query for TXT record `id.server.` or `hostname.bind.` in the CHAOS class (**RFC 4892**). Set to an empty value to disable.

Default: FQDN hostname

7.6.2 version

A version of the server software returned in the response to the query for TXT record `version.server.` or `version.bind.` in the CHAOS class (**RFC 4892**). Set to an empty value to disable.

Default: server version

7.6.3 nsid

A DNS name server identifier (**RFC 5001**). Set to an empty value to disable.

Default: FQDN hostname at the moment of the daemon start

7.6.4 rundir

A path for storing run-time data (PID file, unix sockets, etc.). A non-absolute path is relative to the *knotd* startup directory.

Depending on the usage of this parameter, its change may require restart of the Knot server to take effect.

Default: `${localstatedir}/run/knot` (configured with `--with-rundir=path`)

7.6.5 user

A system user with an optional system group (`user:group`) under which the server is run after starting and binding to interfaces. Linux capabilities are employed if supported.

Change of this parameter requires restart of the Knot server to take effect.

Default: `root:root`

7.6.6 pidfile

A PID file *location*.

Change of this parameter requires restart of the Knot server to take effect.

Default: *rundir*/`knot.pid`

7.6.7 udp-workers

A number of UDP workers (threads) used to process incoming queries over UDP.

Change of this parameter requires restart of the Knot server to take effect.

Default: equal to the number of online CPUs

7.6.8 tcp-workers

A number of TCP workers (threads) used to process incoming queries over TCP.

Change of this parameter requires restart of the Knot server to take effect.

Default: equal to the number of online CPUs, default value is at least 10

7.6.9 background-workers

A number of workers (threads) used to execute background operations (zone loading, zone updates, etc.).

Change of this parameter requires restart of the Knot server to take effect.

Default: equal to the number of online CPUs, default value is at most 10

7.6.10 async-start

If enabled, server doesn't wait for the zones to be loaded and starts responding immediately with SERVFAIL answers until the zone loads.

Default: `off`

7.6.11 tcp-idle-timeout

Maximum idle time (in seconds) between requests on an inbound TCP connection. It means if there is no activity on an inbound TCP connection during this limit, the connection is closed by the server.

Minimum: `1`

Default: `10`

7.6.12 tcp-io-timeout

Maximum time (in milliseconds) to receive or send one DNS message over an inbound TCP connection. It means this limit applies to normal DNS queries and replies, incoming DDNS, and **outgoing zone transfers**. The timeout is measured since some data is already available for processing. Set to 0 for infinity.

Default: `500` (milliseconds)

Caution

In order to reduce the risk of Slow Loris attacks, it's recommended setting this limit as low as possible on public servers.

7.6.13 tcp-remote-io-timeout

Maximum time (in milliseconds) to receive or send one DNS message over an outbound TCP/QUIC/TLS connection which has already been established to a configured remote server. It means this limit applies to incoming zone transfers, sending NOTIFY, DDNS forwarding, and DS check or push. This timeout includes the time needed for a network round-trip and for a query processing by the remote. Set to 0 for infinity.

Default: `5000` (milliseconds)

7.6.14 tcp-reuseport

If enabled, each TCP worker listens on its own socket and the OS kernel socket load balancing is employed using SO_REUSEPORT (or SO_REUSEPORT_LB on FreeBSD). Due to the lack of one shared socket, the server can offer higher response rate processing over TCP. However, in the case of time-consuming requests (e.g. zone transfers of a TLD zone), enabled reuseport may result in delayed or not being responded client requests. So it is advisable to use this option on secondary servers.

Note

This option is ignored for UNIX sockets.

Change of this parameter requires restart of the Knot server to take effect.

Default: `off`

7.6.15 tcp-fastopen

If enabled, use TCP Fast Open for outbound TCP communication (client side): incoming zone transfers, sending NOTIFY, and DDNS forwarding. This mode simplifies TCP handshake and can result in better networking performance. TCP Fast Open for inbound TCP communication (server side) isn't affected by this configuration as it's enabled automatically if supported by OS.

Note

The TCP Fast Open support must also be enabled on the OS level:

- Linux/macOS: ensure kernel parameter `net.ipv4.tcp_fastopen` is `2` or `3` for server side, and `1` or `3` for client side.
- FreeBSD: ensure kernel parameter `net.inet.tcp.fastopen.server_enable` is `1` for server side, and `net.inet.tcp.fastopen.client_enable` is `1` for client side.

Default: `off`

7.6.16 quic-max-clients

A maximum number of QUIC clients connected in parallel.

See also *quic*.

Change of this parameter requires restart of the Knot server to take effect.

Minimum: `128`

Default: `10000` (ten thousand)

7.6.17 quic-outbuf-max-size

Maximum cumulative size of memory used for buffers of unACKed sent messages. This limit is per one UDP worker.

Note

Set low if little memory is available (together with *quic-max-clients* since QUIC connections are memory-heavy). Set to high value if outgoing zone transfers of big zone over QUIC are expected.

Change of this parameter requires restart of the Knot server to take effect.

Minimum: `1M` (1 MiB)

Default: `100M` (100 MiB)

7.6.18 quic-idle-close-timeout

Time in seconds, after which any idle QUIC connection is gracefully closed.

Change of this parameter requires restart of the Knot server to take effect.

Minimum: `1`

Default: `4`

7.6.19 remote-pool-limit

If nonzero, the server will keep up to this number of outgoing TCP connections open for later use. This is an optimization to avoid frequent opening of TCP connections to the same remote.

Change of this parameter requires restart of the Knot server to take effect.

Default: `0`

7.6.20 remote-pool-timeout

The timeout in seconds after which the unused kept-open outgoing TCP connections to remote servers are closed.

Default: `5`

7.6.21 remote-retry-delay

When a connection attempt times out to some remote address, this information will be kept for this specified time (in milliseconds) and other connections to the same address won't be attempted. This prevents repetitive waiting for timeout on an unreachable remote.

Default: `0`

7.6.22 socket-affinity

If enabled and if SO_REUSEPORT is available on Linux, all configured network sockets are bound to UDP and TCP workers in order to increase the networking performance. This mode isn't recommended for setups where the number of network card queues is lower than the number of UDP or TCP workers.

Change of this parameter requires restart of the Knot server to take effect.

Default: `off`

7.6.23 tcp-max-clients

A maximum number of TCP clients connected in parallel, set this below the file descriptor limit to avoid resource exhaustion.

> **Note**
>
> It is advisable to adjust the maximum number of open files per process in your operating system configuration.

Default: one half of the file descriptor limit for the server process

7.6.24 udp-max-payload

Maximum EDNS0 UDP payload size default for both IPv4 and IPv6.

Default: `1232`

7.6.25 udp-max-payload-ipv4

Maximum EDNS0 UDP payload size for IPv4.

Default: `1232`

7.6.26 udp-max-payload-ipv6

Maximum EDNS0 UDP payload size for IPv6.

Default: `1232`

7.6.27 key-file

Path to a server key PEM file which is used for DNS over QUIC/TLS communication. A non-absolute path of a user specified key file is relative to the `@config_dir@` directory.

Default: auto-generated key

7.6.28 cert-file

Path to a server certificate PEM file which is used for DNS over QUIC/TLS communication. A non-absolute path is relative to the `@config_dir@` directory.

Default: one-time in-memory certificate

7.6.29 ca-file

Specifies one or more paths to load trusted Certificate Authorities (CAs) from. An empty string ("") means the system's default trusted CAs. The loaded CAs are used for remote certificate validation (*cert-hostname*, *cert-hostname*, and *zone-db-cert-hostname*).

Default: not set

7.6.30 edns-client-subnet

Enable or disable EDNS Client Subnet support. If enabled, responses to queries containing the EDNS Client Subnet option always contain a valid EDNS Client Subnet option according to **RFC 7871**.

Default: `off`

7.6.31 answer-rotation

Enable or disable sorted-rrset rotation in the answer section of normal replies. The rotation shift is simply determined by a query ID.

Default: `off`

7.6.32 automatic-acl

If enabled, *automatic ACL* setting of configured remotes is considered when evaluating authorized operations.

Default: `off`

7.6.33 proxy-allowlist

An ordered list of IP addresses, network subnets, or network ranges which are allowed as a source address of proxied DNS traffic over UDP. The supported proxy protocol is haproxy PROXY v2.

Note

TCP is not supported.

Default: not set

7.6.34 dbus-event

Specification of server or zone states which emit a D-Bus signal on the system bus. The bus name is `cz.nic.knotd`, the object path is `/cz/nic/knotd`, and the interface name is `cz.nic.knotd.events`.

Possible values:

- `none` – No signal is emitted.
- `running` – There are two possible signals emitted:
  - `started` when the server is started and all configured zones (including catalog zones and their members) are loaded or successfully bootstrapped.
  - `stopped` when the server shutdown sequence is initiated.
- `zone-updated` – The signal `zone_updated` is emitted when a zone has been updated; the signal parameters are *zone name* and *zone SOA serial*.
- `external-verify` - The signal `external_verify` is emitted when a zone is awaiting external validation before applying the changes; the signal parameter is *zone name*.
- `keys-updated` - The signal `keys_updated` is emitted when a DNSSEC key set is updated; the signal parameter is *zone name*.
- `ksk-submission` – The signal `zone_ksk_submission` is emitted if there is a ready KSK present when the zone is signed; the signal parameters are *zone name*, *KSK keytag*, and *KSK KASP id*.
- `dnssec-invalid` – The signal `zone_dnssec_invalid` is emitted when DNSSEC validation fails, or when ZONEMD verification fails; the signal parameters are *zone name*, and *remaining seconds* until an RRSIG expires.

Note

This function requires systemd version at least 221 or libdbus.

Tip

A few sample script templates can be found in the project repository.

Change of this parameter requires restart of the Knot server to take effect.

Default: none

7.6.35 dbus-init-delay

Time in seconds which the server waits upon D-Bus initialization to ensure the D-Bus client is ready to receive signals.

Change of this parameter requires restart of the Knot server to take effect.

Minimum: 0

Default: 1

7.6.36 listen

One or more IP addresses where the server listens for incoming queries. Optional port specification (default is 53) can be appended to each address using @ separator. Use 0.0.0.0 for all configured IPv4 addresses or :: for all configured IPv6 addresses. Filesystem path can be specified for listening on local unix SOCK_STREAM socket. Non-absolute path (i.e. not starting with /) is relative to *rundir*. Non-local address binding is automatically enabled if supported by the operating system.

Change of this parameter requires restart of the Knot server to take effect.

Default: not set

7.6.37 listen-quic

One or more IP addresses (and optionally ports) where the server listens for incoming queries over QUIC protocol.

Change of this parameter requires restart of the Knot server to take effect.

Default: not set

7.6.38 listen-tls

One or more IP addresses (and optionally ports) where the server listens for incoming queries over TLS protocol (DoT).

Change of this parameter requires restart of the Knot server to take effect.

Default: not set

7.7 xdp section

Various options related to XDP listening, especially TCP.

```
xdp:
    listen: STR[@INT] | ADDR[@INT] ...
    udp: BOOL
    tcp: BOOL
    quic: BOOL
    quic-port: INT
    tcp-max-clients: INT
    tcp-inbuf-max-size: SIZE
    tcp-outbuf-max-size: SIZE
```

(continues on next page)

(continued from previous page)

```
    tcp-idle-close-timeout: TIME
    tcp-idle-reset-timeout: TIME
    tcp-resend-timeout: TIME
    route-check: BOOL
    ring-size: INT
    busypoll-budget: INT
    busypoll-timeout: INT
```

Caution

When you change configuration parameters dynamically or via configuration file reload, some parameters in the XDP section require restarting the Knot server so that the changes take effect.

7.7.1 listen

One or more network device names (e.g. `ens786f0`) on which the *Mode XDP* is enabled. Alternatively, an IP address can be used instead of a device name, but the server will still listen on all addresses belonging to the same interface! Optional port specification (default is 53) can be appended to each device name or address using `@` separator.

Change of this parameter requires restart of the Knot server to take effect.

Caution

If XDP workers only process regular DNS traffic over UDP, it is strongly recommended to also *listen* on the addresses which are intended to offer the DNS service, at least to fulfil the DNS requirement for working TCP.

Note

Incoming *DDNS* over XDP isn't supported. The server always responds with SERVFAIL.

Default: not set

7.7.2 udp

If enabled, DNS over UDP is processed with XDP workers.

Change of this parameter requires restart of the Knot server to take effect.

Default: `on`

7.7.3 tcp

If enabled, DNS over TCP traffic is processed with XDP workers.

The TCP stack limitations:

- Congestion control is not implemented.
- Lost packets that do not contain TCP payload may not be resend.
- Not optimized for transfers of non-trivial zones.

Change of this parameter requires restart of the Knot server to take effect.

Default: `off`

7.7.4 quic

If enabled, DNS over QUIC is processed with XDP workers.

Change of this parameter requires restart of the Knot server to take effect.

Default: `off`

7.7.5 quic-port

DNS over QUIC will listen on the interfaces configured by *listen*, but on different port, configured by this option.

Change of this parameter requires restart of the Knot server to take effect.

Default: `853`

7.7.6 tcp-max-clients

A maximum number of TCP clients connected in parallel.

Minimum: `1024`

Default: `1000000` (one million)

7.7.7 tcp-inbuf-max-size

Maximum cumulative size of memory used for buffers of incompletely received messages.

Minimum: `1M` (1 MiB)

Default: `100M` (100 MiB)

7.7.8 tcp-outbuf-max-size

Maximum cumulative size of memory used for buffers of unACKed sent messages.

Minimum: `1M` (1 MiB)

Default: `100M` (100 MiB)

7.7.9 tcp-idle-close-timeout

Time in seconds, after which any idle connection is gracefully closed.

Minimum: `1`

Default: `10`

7.7.10 tcp-idle-reset-timeout

Time in seconds, after which any idle connection is forcibly closed.

Minimum: `1`

Default: `20`

7.7.11 tcp-resend-timeout

Resend outgoing data packets (with DNS response payload) if not ACKed before this timeout (in seconds).

Minimum: `1`

Default: `5`

7.7.12 route-check

If enabled, routing information from the operating system is considered when processing every incoming DNS packet received over the XDP interface:

- If the outgoing interface of the corresponding DNS response differs from the incoming one, the packet is processed normally by UDP/TCP workers (XDP isn't used).
- If the destination address is blackholed, unreachable, or prohibited, the DNS packet is dropped without any response.
- The destination MAC address and possible VLAN tag for the response are taken from the routing system.

If disabled, symmetrical routing is applied. It means that the query source MAC address is used as a response destination MAC address. Possible VLAN tag is preserved.

Change of this parameter requires restart of the Knot server to take effect.

Note

This mode requires forwarding enabled on the loopback interface (`sysctl -w net.ipv4.conf.lo.forwarding=1` and `sysctl -w net.ipv6.conf.lo.forwarding=1`). If forwarding is disabled, all incoming DNS packets are dropped!

Only VLAN 802.1Q is supported.

Default: `off`

7.7.13 ring-size

Size of RX, FQ, TX, and CQ rings.

Change of this parameter requires restart of the Knot server to take effect.

Note

This value should be at least as high as the configured RX size of the network device in the XDP mode.

Default: `2048`

7.7.14 busypoll-budget

If set to a positive value, preferred busy polling is enabled with the specified budget.

Change of this parameter requires restart of the Knot server to take effect.

Note

Preferred busy polling also requires setting `napi_defer_hard_irqs` and `gro_flush_timeout` for the appropriate network interface. E.g.:

```
echo 2 | sudo tee /sys/class/net/<interface>/napi_defer_hard_irqs
echo 200000 | sudo tee /sys/class/net/<interface>/gro_flush_timeout
```

Note

A recommended value is between 8 and 64.

Default: `0` (disabled)

7.7.15 busypoll-timeout

Timeout in microseconds of preferrred busy polling if enabled by *busypoll-budget*.

Change of this parameter requires restart of the Knot server to take effect.

Default: `20` (20 microseconds)

7.8 `control` section

Configuration of the server control interface.

```
control:
    listen: STR ...
    backlog: INT
    timeout: TIME
```

7.8.1 listen

A UNIX socket *path* where the server listens for control commands.

Multiple sockets can be configured for parallel independent use, but their number is limited (currently to 4), and some operations might be delayed due to mutexes.

Warning

Transaction-like operations, such as conf-begin/set/commit/abort or zone-begin/set/commit/abort, must be performed using the same socket.

Change of this parameter requires restart of the Knot server to take effect.

Default: *rundir*`/knot.sock`

7.8.2 backlog

The control UNIX socket listen backlog size.

Change of this parameter requires restart of the Knot server to take effect.

Default: `5`

7.8.3 timeout

Maximum time (in seconds) the control socket operations can take. Set to 0 for infinity.

Default: `5`

7.9 `log` section

Server can be configured to log to the standard output, standard error output, syslog (or systemd journal if systemd is enabled) or into an arbitrary file.

There are 6 logging severity levels:

- `critical` – Non-recoverable error resulting in server shutdown.
- `error` – Recoverable error, action should be taken.
- `warning` – Warning that might require user action.
- `notice` – Server notice or hint.
- `info` – Informational message.

- debug – Debug or detailed message.

In the case of a missing log section, warning or more serious messages will be logged to both standard error output and syslog. The info and notice messages will be logged to standard output.

```
log:
  - target: stdout | stderr | syslog | STR
    server: critical | error | warning | notice | info | debug
    control: critical | error | warning | notice | info | debug
    zone: critical | error | warning | notice | info | debug
    quic: critical | error | warning | notice | info | debug
    any: critical | error | warning | notice | info | debug
```

7.9.1 target

A logging output.

Possible values:

- stdout – Standard output.
- stderr – Standard error output.
- syslog – Syslog or systemd journal.
- *file_name* – A specific file.

With syslog target, syslog service is used. However, if Knot DNS has been compiled with systemd support and operating system has been booted with systemd, systemd journal is used for logging instead of syslog.

A *file_name* may be specified as an absolute path or a path relative to the *knotd* startup directory.

7.9.2 server

Minimum severity level for messages related to general operation of the server to be logged.

Default: not set

7.9.3 control

Minimum severity level for messages related to server control to be logged.

Default: not set

7.9.4 zone

Minimum severity level for messages related to zones to be logged.

Default: not set

7.9.5 quic

Minimum severity level for messages related to QUIC to be logged.

Default: not set

7.9.6 any

Minimum severity level for all message types, except quic, to be logged.

Default: not set

7.10 statistics section

Periodic server statistics dumping.

```
statistics:
    timer: TIME
    file: STR
    append: BOOL
```

7.10.1 timer

A period (in seconds) after which all available statistics metrics will by written to the *file*.

Default: not set

7.10.2 file

A file *path* of statistics output in the YAML format.

Default: *rundir*/stats.yaml

7.10.3 append

If enabled, the output will be appended to the *file* instead of file replacement.

Default: off

7.11 database section

Configuration of databases for zone contents, DNSSEC metadata, or event timers.

```
database:
    storage: STR
    journal-db: STR
    journal-db-mode: robust | asynchronous
    journal-db-max-size: SIZE
    kasp-db: STR
    kasp-db-max-size: SIZE
    timer-db: STR
    timer-db-max-size: SIZE
    catalog-db: str
    catalog-db-max-size: SIZE
    zone-db-listen: ADDR[@INT] | STR
    zone-db-tls: BOOL
    zone-db-cert-key: BASE64 ...
    zone-db-cert-hostname: STR ...
```

7.11.1 storage

A data directory for storing journal, KASP, and timer databases. A non-absolute path is relative to the *knotd* startup directory.

Default: ${localstatedir}/lib/knot (configured with --with-storage=path)

7.11.2 journal-db

An explicit *specification* of the persistent journal database directory.

Default: *storage*/journal

7.11.3 journal-db-mode

Specifies journal LMDB backend configuration, which influences performance and durability.

Possible values:

- `robust` – The journal database disk synchronization ensures database durability but is generally slower.
- `asynchronous` – The journal database disk synchronization is optimized for better performance at the expense of lower database durability in the case of a crash. This mode is recommended on secondary servers with many zones.

Default: `robust`

7.11.4 journal-db-max-size

The hard limit for the journal database maximum size. There is no cleanup logic in journal to recover from reaching this limit. Journal simply starts refusing changes across all zones. Decreasing this value has no effect if it is lower than the actual database file size.

It is recommended to limit *journal-max-usage* per-zone instead of *journal-db-max-size* in most cases. Please keep this value larger than the sum of all zones' journal usage limits. See more details regarding *journal behaviour*.

Note

This value also influences server's usage of virtual memory.

Default: `20G` (20 GiB), or `512M` (512 MiB) for 32-bit

7.11.5 kasp-db

An explicit *specification* of the KASP database directory.

Default: *storage*`/keys`

7.11.6 kasp-db-max-size

The hard limit for the KASP database maximum size.

Note

This value also influences server's usage of virtual memory.

Default: `500M` (500 MiB)

7.11.7 timer-db

An explicit *specification* of the persistent timer database directory.

Default: *storage*`/timers`

7.11.8 timer-db-max-size

The hard limit for the timer database maximum size.

Note

This value also influences server's usage of virtual memory.

Default: `100M` (100 MiB)

7.11.9 catalog-db

An explicit *specification* of the zone catalog database directory. Only useful if *Catalog zones* are enabled.

Default: *storage*`/catalog`

7.11.10 catalog-db-max-size

The hard limit for the catalog database maximum size.

> **Note**
>
> This value also influences server's usage of virtual memory.

Default: `20G` (20 GiB), or `512M` (512 MiB) for 32-bit

7.11.11 zone-db-listen

An IP address (and optionally a port) or a UNIX socket path of a running instance of a Redis (or compatible) database to be used for reading and/or writing zone contents. See *zone-db-input* and *zone-db-output*.

Default: not set

7.11.12 zone-db-tls

If enabled, TLS 1.3 will be used for communication with the zone database.

Default: `off`

7.11.13 zone-db-cert-key

An ordered list of up to 4 public key PINs of the zone database's certificate. If the list is non-empty, communication with the zone database is only possible over TLS, and a peer certificate is required. The peer certificate's public key must match one of the specified PINs.

Default: not set

7.11.14 zone-db-cert-hostname

An ordered list of up to 4 hostnames to be matched against the zone database's certificate. At least one hostname must match for the certificate to be considered valid (see *ca-file*). If the list is non-empty, communication with the zone database is only possible over TLS, and a peer certificate is required.

Default: not set

7.12 keystore section

DNSSEC keystore configuration.

```
keystore:
  - id: STR
    backend: pem | pkcs11
    config: STR
    ksk-only: BOOL
    key-label: BOOL
```

7.12.1 id

A keystore identifier.

7.12.2 backend

A key storage backend type.

Possible values:

- `pem` – PEM files.
- `pkcs11` – PKCS #11 storage.

Default: `pem`

7.12.3 config

A backend specific configuration. A directory with PEM files (the path can be specified as a relative path to *kasp-db*) or a configuration string for PKCS #11 storage (*<pkcs11-uri> <module-path>*). The PKCS #11 URI Scheme is defined in **RFC 7512**.

> **Note**
>
> Example configuration string for PKCS #11:
>
> ```
> "pkcs11:token=knot;pin-value=1234 /usr/lib64/pkcs11/libsofthsm2.so"
> ```

Default: *kasp-db*`/keys`

7.12.4 ksk-only

Newly generated keys sre stored in this keystore only if they are KSKs or CSKs. Zone signing keys will be stored in subsequent keystore without this option enabled.

Default: `off`

7.12.5 key-label

If enabled in combination with the PKCS #11 *backend*, generated keys are labeled in the form `<zone_name> KSK|ZSK`.

Default: `off`

7.13 key section

Shared TSIG keys used to authenticate communication with the server.

```
key:
  - id: DNAME
    algorithm: hmac-md5 | hmac-sha1 | hmac-sha224 | hmac-sha256 | hmac-sha384 | hmac-
↪sha512
    secret: BASE64
```

7.13.1 id

A key name identifier.

Note

This value MUST be exactly the same as the name of the TSIG key on the opposite primary/secondary server(s).

7.13.2 algorithm

A TSIG key algorithm. See TSIG Algorithm Numbers.

Possible values:

- `hmac-md5`
- `hmac-sha1`
- `hmac-sha224`
- `hmac-sha256`
- `hmac-sha384`
- `hmac-sha512`

Default: `hmac-sha256`

7.13.3 secret

Shared key secret.

Default: not set

7.14 `remote` section

Definitions of remote servers for outgoing connections (source of a zone transfer, target for a notification, etc.).

```
remote:
  - id: STR
    address: ADDR[@INT] | STR ...
    via: ADDR[@INT] ...
    quic: BOOL
    tls: BOOL
    key: key_id
    cert-key: BASE64 ...
    cert-hostname: STR ...
    block-notify-after-transfer: BOOL
    no-edns: BOOL
    automatic-acl: BOOL
```

7.14.1 id

A remote identifier.

7.14.2 address

An ordered list of destination IP addresses or UNIX socket paths which are used for communication with the remote server. Non-absolute path (i.e. not starting with `/`) is relative to *rundir*. Optional destination port (default is 53 for UDP/TCP and 853 for QUIC) can be appended to the address using `@` separator. The addresses are tried in sequence until the remote is reached.

Default: not set

Note

If the remote is contacted and it refuses to perform requested action, no more addresses will be tried for this remote.

7.14.3 via

An ordered list of source IP addresses which are used as source addresses for communication with the remote. For the N-th *remote address*, the last, but at most N-th, specified *via address* of the same family is used. This option can help if the server listens on more addresses. Optional source port (default is random) can be appended to the address using `@` separator.

Default: not set

Note

For the following configuration:

```
remote:
  - id: example
    address: [198.51.100.10, 2001:db8::10, 198.51.100.20, 2001:db8::20]
    via: [198.51.100.1, 198.51.100.2, 2001:db8::1]
```

the (`via` -> `address`) mapping is:

- `198.51.100.1` -> `198.51.100.10`
- `2001:db8::1` -> `2001:db8::10`
- `198.51.100.2` -> `198.51.100.20`
- `2001:db8::1` -> `2001:db8::20`

7.14.4 quic

If this option is set, the QUIC protocol will be used for outgoing communication with this remote.

Note

One connection per each remote is opened; *remote-pool-limit* does not take effect for QUIC. However, fast QUIC handshakes utilizing obtained session tickets are used for reopening connections to recently (up to 1 day) queried remotes.

Default: `off`

7.14.5 tls

If this option is set, the TLS (DoT) protocol will be used for outgoing communication with this remote.

Default: `off`

7.14.6 key

A *reference* to the TSIG key which is used to authenticate the communication with the remote server.

Default: not set

7.14.7 cert-key

An ordered list of up to 4 remote certificate public key PINs. If the list is non-empty, communication with the remote is only possible via QUIC or TLS protocols, and a peer certificate is required. The peer certificate key must match one of the specified PINs.

A PIN is a unique identifier that represents the public key of the peer certificate. It's a base64-encoded SHA-256 hash of the public key. This identifier usually remains the same on a certificate renewal.

Default: not set

7.14.8 cert-hostname

An ordered list of up to 4 hostnames to match against peer's certificate. At least one must match for successful certificate validation (see *ca-file*). If the list is non-empty, communication with the remote is only possible via QUIC or TLS protocols, and a peer certificate is required.

Default: not set

7.14.9 block-notify-after-transfer

When incoming AXFR/IXFR from this remote (as a primary server), suppress sending NOTIFY messages to all configured secondary servers.

Default: `off`

7.14.10 no-edns

If enabled, no OPT record (EDNS) is inserted to outgoing requests to this remote server. This mode is necessary for communication with some broken DNS implementations (e.g. Windows Server 2016).

Additionally, if TCP is used for zone refresh, the SOA query and the subsequent AXFR/IXFR query do not share the same TCP connection. This mode allows transfers from some broken DNS implementations (e.g. ixfrdist).

> **Note**
>
> This option effectively disables *zone expire* timer updates via EDNS EXPIRE option specified in **RFC 7314**.

Default: `off`

7.14.11 automatic-acl

If enabled, some authorized operations for the remote are automatically allowed based on the context:

- Incoming NOTIFY is allowed from the remote if it's configured as a *primary server* for the zone.
- Outgoing zone transfer is allowed to the remote if it's configured as a *NOTIFY target* for the zone.

Automatic ACL rules are evaluated before explicit *zone ACL* configuration.

> **Note**
>
> This functionality requires global activation via *automatic-acl* in the server section.

Default: `on`

7.15 `remotes` section

Definitions of groups of remote servers. Remote grouping can simplify the configuration.

```
remotes:
  - id: STR
    remote: remote_id ...
```

7.15.1 id

A remote group identifier.

7.15.2 remote

An ordered list of *references* to remote server definitions.

Default: not set

7.16 `acl` section

Access control list rule definitions. An ACL rule is a description of one or more authorized actions (zone transfer request, zone change notification, and dynamic DNS update) which are allowed to be processed or denied. Queries which don't require authorization are always allowed.

```
acl:
  - id: STR
    address: ADDR[/INT] | ADDR-ADDR | STR ...
    key: key_id ...
    cert-key: BASE64 ...
    cert-hostname: STR ...
    remote: remote_id | remotes_id ...
    action: query | notify | transfer | update ...
    protocol: udp | tcp | tls | quic ...
    deny: BOOL
    update-type: STR ...
    update-owner: key | zone | name
    update-owner-match: sub-or-equal | equal | sub | pattern
    update-owner-name: STR ...
```

7.16.1 id

An ACL rule identifier.

7.16.2 address

An ordered list of IP addresses, absolute UNIX socket paths, network subnets, or network ranges. The query's source address must match one of them. If this item is not set, address match is not required.

Default: not set

7.16.3 key

An ordered list of *reference*s to TSIG keys. The query must match one of them. If this item is not set, transaction authentication is not used.

Default: not set

7.16.4 cert-key

An ordered list of remote certificate public key PINs. If the list is non-empty, communication with the remote is only possible via QUIC or TLS protocols, and a peer certificate is required. The peer certificate key must match one of the specified PINs.

A PIN is a unique identifier that represents the public key of the peer certificate. It's a base64-encoded SHA-256 hash of the public key. This identifier usually remains the same on a certificate renewal.

Default: not set

7.16.5 cert-hostname

An ordered list of hostnames to match against peer's certificate. At least one must match for successful certificate validation (see *ca-file*). If the list is non-empty, communication with the remote is only possible via QUIC or TLS protocols, and a peer certificate is required.

Default: not set

7.16.6 remote

An ordered list of references *remote* and *remotes*. The query must match one of the remotes. Specifically, one of the remote's addresses and remote's TSIG key if configured must match.

> **Note**
>
> This option cannot be specified along with the *address*, *key*, or *protocol* option at one ACL item.

Default: not set

7.16.7 action

An ordered list of allowed, or denied, actions (request types).

Possible values:

- `query` – Allow regular DNS query. As normal queries are always allowed, this action is only useful in combination with *TSIG key*.
- `notify` – Allow incoming notify (NOTIFY).
- `transfer` – Allow zone transfer (AXFR, IXFR).
- `update` – Allow zone updates (DDNS).

Default: `query`

7.16.8 protocol

List of allowed protocols.

Possible values:

- `udp` – UDP protocol.
- `tcp` – TCP protocol.
- `tls` – TLS protocol.
- `quic` – QUIC protocol.

Default: not set (any)

7.16.9 deny

If enabled, instead of allowing, deny the matching combination of the specified items.

Default: `off`

7.16.10 update-type

A list of allowed types of Resource Records in a zone update. Every record in an update must match one of the specified types.

Default: not set

7.16.11 update-owner

This option restricts possible owners of Resource Records in a zone update by comparing them to either the *TSIG key* identity, the current zone name, or to a list of domain names given by the *update-owner-name* option. The comparison method is given by the *update-owner-match* option.

Possible values:

- `key` — The owner of each updated RR must match the identity of the TSIG key if used.
- `name` — The owner of each updated RR must match at least one name in the *update-owner-name* list.
- `zone` — The owner of each updated RR must match the current zone name.

Default: not set

7.16.12 update-owner-match

This option defines how the owners of Resource Records in an update are matched to the domain name(s) set by the *update-owner* option.

Possible values:

- `sub-or-equal` — The owner of each RR in an update must either be equal to or be a subdomain of at least one domain name set by *update-owner*.
- `equal` — The owner of each updated RR must be equal to at least one domain name set by *update-owner*.
- `sub` — The owner of each updated RR must be a subdomain of, but MUST NOT be equal to at least one domain name set by *update-owner*.
- `pattern` — The owner of each updated RR must match a pattern specified by *update-owner*. The pattern can be an arbitrary FQDN or non-FQDN domain name. If a label consists of one `*` (asterisk) character, it matches any label. More asterisk labels can be specified.

Default: `sub-or-equal`

7.16.13 update-owner-name

A list of allowed owners of RRs in a zone update used with *update-owner* set to `name`. Every listed owner name which is not FQDN (i.e. it doesn't end in a dot) is considered as if it was appended with the target zone name. Such a relative owner name specification allows better ACL rule reusability across multiple zones.

Default: not set

7.17 `submission` section

Parameters of KSK submission checks.

```
submission:
  - id: STR
    parent: remote_id | remotes_id ...
    check-interval: TIME
    timeout: TIME
    parent-delay: TIME
```

7.17.1 id

A submission identifier.

7.17.2 parent

A list of references *remote* and *remotes* to parent's DNS servers to be checked for presence of corresponding DS records in the case of KSK submission. All of them must have a corresponding DS for the rollover to continue. If none is specified, the rollover must be pushed forward manually.

Default: not set

Tip

A DNSSEC-validating resolver can be set as a parent.

7.17.3 check-interval

Interval (in seconds) for periodic checks of DS presence on parent's DNS servers, in the case of the KSK submission.

Default: `1h` (1 hour)

7.17.4 timeout

After this time period (in seconds) the KSK submission is automatically considered successful, even if all the checks were negative or no parents are configured. Set to 0 for infinity.

Default: `0`

7.17.5 parent-delay

After successful parent DS check, wait for this period (in seconds) before continuing the next key roll-over step. This delay shall cover the propagation delay of update in the parent zone.

Default: `0`

7.18 dnskey-sync section

Parameters of DNSKEY dynamic-update synchronization.

```
dnskey-sync:
  - id: STR
    remote: remote_id | remotes_id ...
    check-interval: TIME
```

7.18.1 id

A dnskey-sync identifier.

7.18.2 remote

A list of references *remote* and *remotes* to other signers or common master, which the DDNS updates with DNSKEY/CDNSKEY/CDS records shall be sent to.

Default: not set

7.18.3 check-interval

If the last DNSKEY sync failed or resulted in any change, re-check the consistence after this interval (in seconds) and re-try if needed.

Default: `60` (1 minute)

7.19 `policy` section

DNSSEC policy configuration.

```
policy:
  - id: STR
    keystore: keystore_id ...
    manual: BOOL
    single-type-signing: BOOL
    algorithm: rsasha1 | rsasha1-nsec3-sha1 | rsasha256 | rsasha512 | ecdsap256sha256␣
↪| ecdsap384sha384 | ed25519 | ed448
    ksk-size: SIZE
    zsk-size: SIZE
    ksk-shared: BOOL
    dnskey-ttl: TIME
    zone-max-ttl: TIME
    keytag-modulo: INT/INT
    ksk-lifetime: TIME
    zsk-lifetime: TIME
    delete-delay: TIME
    propagation-delay: TIME
    rrsig-lifetime: TIME
    rrsig-refresh: TIME
    rrsig-pre-refresh: TIME
    reproducible-signing: BOOL
    nsec3: BOOL
    nsec3-iterations: INT
    nsec3-opt-out: BOOL
    nsec3-salt-length: INT
    nsec3-salt-lifetime: TIME
    signing-threads: INT
    ksk-submission: submission_id
    ds-push: remote_id | remotes_id ...
    cds-cdnskey-publish: none | delete-dnssec | rollover | always | double-ds
    cds-digest-type: sha256 | sha384
    dnskey-management: full | incremental
    offline-ksk: BOOL
    unsafe-operation: none | no-check-keyset | no-update-dnskey | no-update-nsec | no-
↪update-expired ...
```

7.19.1 id

A policy identifier.

7.19.2 keystore

A *reference* to a keystore holding private key material for zones.

If multiple keystores are specified, private keys for signing are looked up in all of them. But newly generated keys are stored in the first one (or in the first one without enabled *ksk-only* in the case of a new ZSK) in the specified order.

> **Note**
>
> If multiple keystores are configured and a zone is being restored with the *back up* feature, all restored private keys are stored into the first referenced keystore.

> **Note**
>
> A configured keystore called "default" won't be used unless explicitly referenced.

Default: an imaginary keystore with all default values

7.19.3 manual

If enabled, automatic key management is not used.

Default: `off`

7.19.4 single-type-signing

If enabled, Single-Type Signing Scheme is used in the automatic key management mode.

Default: `off` (*module onlinesign* has default `on`)

7.19.5 algorithm

An algorithm of signing keys and issued signatures. See DNSSEC Algorithm Numbers.

Possible values:

- `rsasha1`
- `rsasha1-nsec3-sha1`
- `rsasha256`
- `rsasha512`
- `ecdsap256sha256`
- `ecdsap384sha384`
- `ed25519`
- `ed448`

> **Note**
>
> Ed448 algorithm is only available if compiled with GnuTLS 3.6.12+ and Nettle 3.6+.

Default: `ecdsap256sha256`

7.19.6 ksk-size

A length of newly generated KSK (Key Signing Key) or CSK (Combined Signing Key) keys.

Default: `2048` (rsa*), `256` (ecdsap256), `384` (ecdsap384), `256` (ed25519), `456` (ed448)

7.19.7 zsk-size

A length of newly generated ZSK (Zone Signing Key) keys.

Default: see default for *ksk-size*

7.19.8 ksk-shared

If enabled, all zones with this policy assigned will share one or more KSKs. More KSKs can be shared during a KSK rollover.

Warning

As the shared KSK set is bound to the policy *id*, renaming the policy breaks this connection and new shared KSK set is initiated when a new KSK is needed.

Default: `off`

7.19.9 dnskey-ttl

A TTL value for DNSKEY records added into zone apex.

Note

Has influence over ZSK key lifetime.

Warning

Ensure all DNSKEYs with updated TTL are propagated before any subsequent DNSKEY rollover starts.

Default: zone SOA TTL

7.19.10 zone-max-ttl

Declare (override) maximal TTL value among all the records in zone.

Note

It's generally recommended to override the maximal TTL computation by setting this explicitly whenever possible. It's required for *DNSSEC Offline KSK* and really reasonable when records are generated dynamically (e.g. by a *module*).

Default: computed after zone is loaded

7.19.11 keytag-modulo

Specifies that the keytags of any generated keys shall be congruent by specified modulo. The option value must be a string in the format `R/M`, where `R < M <= 256` are positive integers. Whenever a DNSSEC key is generated, it is ensured that `keytag % M == R`. This prevents keytag conflict in *DNSSEC Offline KSK* or *DNSSEC multi-signer* (and possibly other) setups.

Note

This only applies to newly generated keys when they are generated. Keys from before this option and keys imported from elsewhere might not fulfill the policy.

Default: `0/1`

7.19.12 ksk-lifetime

A period (in seconds) between KSK generation and the next rollover initiation.

Note

KSK key lifetime is also influenced by propagation-delay, dnskey-ttl, and KSK submission delay.

Zero (aka infinity) value causes no KSK rollover as a result.

This applies for CSK lifetime if single-type-signing is enabled.

Default: `0` (infinity)

7.19.13 zsk-lifetime

A period (in seconds) between ZSK activation and the next rollover initiation.

Note

More exactly, this period is measured since a ZSK is activated, and after this, a new ZSK is generated to replace it within following roll-over.

As a consequence, in normal operation, this results in the period of ZSK generation being *zsk-lifetime* + *propagation-delay* + *dnskey_ttl*.

Zero (aka infinity) value causes no ZSK rollover as a result.

Default: `30d` (30 days)

7.19.14 delete-delay

Once a key (KSK or ZSK) is rolled-over and removed from the zone, keep it in the KASP database for at least this period (in seconds) before deleting it completely. This might be useful in some troubleshooting cases when resurrection is needed.

Default: `0`

7.19.15 propagation-delay

An extra delay added for each key rollover step. This value (in seconds) should be high enough to cover propagation of data from the primary server to all secondary servers, as well as the duration of signing routine itself and possible outages in signing and propagation infrastructure. In other words, this delay should ensure that within this period of time after planned change of the key set, all public-facing secondaries will already serve new DNSKEY RRSet for sure.

Note

Has influence over ZSK key lifetime.

Default: `1h` (1 hour)

7.19.16 rrsig-lifetime

A validity period (in seconds) of newly issued signatures.

> **Note**
>
> The RRSIG's signature inception time is set to 90 minutes in the past. This time period is not counted to the signature lifetime.

Default: `14d` (14 days)

7.19.17 rrsig-refresh

A period (in seconds) how long at least before a signature expiration the signature will be refreshed, in order to prevent expired RRSIGs on secondary servers or resolvers' caches.

Default: 0.1 * *rrsig-lifetime* + *propagation-delay* + *zone-max-ttl*

If *dnssec-validation* is enabled:

Default: `1d` (1 day)

7.19.18 rrsig-pre-refresh

A period (in seconds) how long at most before a signature refresh time the signature might be refreshed, in order to refresh RRSIGs in bigger batches on a frequently updated zone (avoid re-sign event too often).

Default: `1h` (1 hour)

7.19.19 reproducible-signing

For ECDSA algorithms, generate RRSIG signatures deterministically (**RFC 6979**). Besides better theoretical cryptographic security, this mode allows significant speed-up of loading signed (by the same method) zones. However, the zone signing is a bit slower.

Default: `off`

7.19.20 nsec3

Specifies if NSEC3 will be used instead of NSEC.

Default: `off`

7.19.21 nsec3-iterations

A number of additional times the hashing is performed.

Default: `0`

7.19.22 nsec3-opt-out

If set, NSEC3 records won't be created for insecure delegations. This speeds up the zone signing and reduces overall zone size.

> **Warning**
>
> NSEC3 with the Opt-Out bit set no longer works as a proof of non-existence in this zone.

Default: `off`

7.19.23 nsec3-salt-length

A length of a salt field in octets, which is appended to the original owner name before hashing.

Default: `0`

7.19.24 nsec3-salt-lifetime

A validity period (in seconds) of newly issued salt field.

Zero value means infinity.

Special value *-1* triggers re-salt every time when active ZSK changes. This optimizes the number of big changes to the zone.

Default: `30d` (30 days)

7.19.25 signing-threads

When signing zone or update, use this number of threads for parallel signing.

Those are extra threads independent of *Background workers*.

> **Note**
>
> Some steps of the DNSSEC signing operation are not parallelized.

Default: `1` (no extra threads)

7.19.26 ksk-submission

A reference to *submission* section holding parameters of KSK submission checks.

Default: not set

7.19.27 ds-push

Optional references *remote* and *remotes* to authoritative DNS server of the parent's zone. The remote server must be configured to accept DS record updates via DDNS. Whenever a CDS record in the local zone is changed, the corresponding DS record is sent as a dynamic update (DDNS) to the parent DNS server. All previous DS records are deleted within the DDNS message. It's possible to manage both child and parent zones by the same Knot DNS server.

> **Note**
>
> This feature requires *cds-cdnskey-publish* not to be set to `none`.

> **Note**
>
> The mentioned change to CDS record usually means that a KSK roll-over is running and the new key being rolled-in is in "ready" state already for the period of *propagation-delay*.

> **Note**
>
> Module *Onlinesign* doesn't support DS push.

Note

When turning this feature on while a KSK roll-over is already running, it might not take effect for the already-running roll-over.

Default: not set

7.19.28 dnskey-sync

A reference to *dnskey-sync* section holding parameters of DNSKEY synchronization.

Default: not set

7.19.29 cds-cdnskey-publish

Controls if and how shall the CDS and CDNSKEY be published in the zone.

Possible values:

- `none` – Never publish any CDS or CDNSKEY records in the zone.
- `delete-dnssec` – Publish special CDS and CDNSKEY records indicating turning off DNSSEC.
- `rollover` – Publish CDS and CDNSKEY records for ready and not yet active KSK (submission phase of KSK rollover).
- `always` – Always publish one CDS and one CDNSKEY records for the current KSK.
- `double-ds` – Always publish up to two CDS and two CDNSKEY records for ready and/or active KSKs.

Note

If the zone keys are managed manually, the CDS and CDNSKEY rrsets may contain more records depending on the keys available.

Warning

The `double-ds` value does not trigger double-DS roll-over method. That method is only supported when performed manually, with unset *ksk-submission*.

Default: `rollover`

7.19.30 cds-digest-type

Specify digest type for published CDS records.

Possible values:

- `sha256`
- `sha384`

Default: `sha256`

7.19.31 dnskey-management

Specify how the DNSKEY, CDNSKEY, and CDS RRSets at the zone apex are handled when (re-)signing the zone.

Possible values:

- `full` – Upon every zone (re-)sign, delete all unknown DNSKEY, CDNSKEY, and CDS records and keep just those that are related to the zone keys stored in the KASP database.

- `incremental` – Keep unknown DNSKEY, CDNSKEY, and CDS records in the zone, and modify server-managed records incrementally by employing changes in the KASP database.

> **Note**
>
> Prerequisites for *incremental*:
>
> - The *Offline KSK* isn't supported.
> - The *delete-delay* is long enough to cover possible daemon shutdown (e.g. due to server maintenance).
> - Avoided manual deletion of keys with *keymgr*.
>
> Otherwise there might remain some DNSKEY records in the zone, belonging to deleted keys.

Default: `full`

7.19.32 offline-ksk

Specifies if *Offline KSK* feature is enabled.

Default: `off`

7.19.33 unsafe-operation

Turn off some DNSSEC safety features.

Possible values:

- `none` – Nothing disabled.
- `no-check-keyset` – Don't check active keys in present algorithms. This may lead to violation of **RFC 4035 Section 2.2**.
- `no-update-dnskey` – Don't maintain/update DNSKEY, CDNSKEY, and CDS records in the zone apex according to KASP database. Juste leave them as they are in the zone.
- `no-update-nsec` – Don't maintain/update NSEC/NSEC3 chain. Leave all the records as they are in the zone.
- `no-update-expired` – Don't update expired RRSIGs.

Multiple values may be specified.

> **Warning**
>
> This mode is intended for DNSSEC experts who understand the corresponding consequences.

Default: `none`

7.20 `external` section

External zone validation configuration.

```
external:
  - id: STR
    timeout: TIME
    dump-new-zone: STR
    dump-removals: STR
    dump-additions: STR
```

7.20.1 id

An external section identifier.

7.20.2 timeout

If the validation is not confirmed within this time interval in seconds, it is considered failed.

Default: `300`

7.20.3 dump-new-zone

A path to file where the new zone contents will be written before waiting for external validation.

Default: none

7.20.4 dump-removals

A path to file where the records being removed will be written before waiting for external validation.

Default: none

7.20.5 dump-additions

A path to file where the records being added will be written before waiting for external validation.

Default: none

7.21 `template` section

A template is shareable zone settings, which can simplify configuration by reducing duplicates. A special default template (with the *default* identifier) can be used for global zone configuration or as an implicit configuration if a zone doesn't have another template specified.

```
template:
  - id: STR
    global-module: STR/STR ...
    # All zone options (excluding 'template' item)
```

Note

If an item is explicitly specified both in the referenced template and the zone, the template item value is overridden by the zone item value.

7.21.1 id

A template identifier.

7.21.2 global-module

An ordered list of references to query modules in the form of *module_name* or *module_name/module_id*. These modules apply to all queries.

Note

This option is only available in the *default* template.

Default: not set

7.22 zone section

Definition of zones served by the server.

```
zone:
  - domain: DNAME
    template: template_id
    storage: STR
    file: STR
    zone-db-input: INT
    zone-db-output: INT
    master: remote_id | remotes_id ...
    ddns-master: remote_id
    notify: remote_id | remotes_id ...
    notify-delay: TIME
    update-delay: TIME
    acl: acl_id ...
    master-pin-tolerance: TIME
    provide-ixfr: BOOL
    semantic-checks: BOOL | soft
    default-ttl: TIME
    zonefile-sync: TIME
    zonefile-load: none | difference | difference-no-serial | whole
    zonefile-skip: STR ...
    journal-content: none | changes | all
    journal-max-usage: SIZE
    journal-max-depth: INT
    ixfr-benevolent: BOOL
    ixfr-by-one: BOOL
    ixfr-from-axfr: BOOL
    zone-max-size : SIZE
    adjust-threads: INT
    external-validation: external_id
    dnssec-signing: BOOL
    dnssec-validation: BOOL
    dnssec-policy: policy_id
    ds-push: remote_id | remotes_id ...
    zonemd-verify: BOOL
    zonemd-generate: none | zonemd-sha384 | zonemd-sha512 | remove
    serial-policy: increment | unixtime | dateserial
    serial-modulo: INT/INT | +INT | -INT | INT/INT+INT | INT/INT-INT
    reverse-generate: DNAME ...
    include-from: DNAME ...
    refresh-min-interval: TIME
    refresh-max-interval: TIME
    retry-min-interval: TIME
    retry-max-interval: TIME
    expire-min-interval: TIME
    expire-max-interval: TIME
    catalog-role: none | interpret | generate | member
    catalog-template: template_id ...
    catalog-zone: DNAME
    catalog-group: STR
    module: STR/STR ...
```

7.22.1 domain

A zone name identifier.

7.22.2 template

A *reference* to a configuration template.

Default: not set or `default` (if the template exists)

7.22.3 storage

A data directory for storing zone files. A non-absolute path is relative to the *knotd* startup directory.

Default: `${localstatedir}/lib/knot` (configured with `--with-storage=path`)

7.22.4 file

A *path* to the zone file. It is also possible to use the following formatters:

- `%c[`*N*`]` or `%c[`*N*`-`*M*`]` – Means the *N*th character or a sequence of characters beginning from the *N*th and ending with the *M*th character of the textual zone name (see `%s`). The indexes are counted from 0 from the left. All dots (including the terminal one) are considered. If the character is not available, the formatter has no effect.
- `%l[`*N*`]` – Means the *N*th label of the textual zone name (see `%s`). The index is counted from 0 from the right (0 ~ TLD). If the label is not available, the formatter has no effect.
- `%s` – Means the current zone name in the textual representation. The zone name doesn't include the terminating dot (the result for the root zone is the empty string!).
- `%%` – Means the `%` character.

Warning

Beware of special characters which are escaped or encoded in the \DDD form where DDD is corresponding decimal ASCII code.

Default: *storage*`/%s.zone`

7.22.5 zone-db-input

If set, the zone is loaded from the zone database configured at *zone-db-listen*. The value of this option specifies the zone instance number (from 1 to 8 inclusive) within the database to read from.

Note

With this option enabled, the textual zone file is never loaded. The *zonefile-load* setting still applies to handling of the zone contents loaded from the database, however, `zonefile-load: whole` and `journal-content: changes` are recommended (both being the defaults).

Default: `-1` (disabled)

7.22.6 zone-db-output

If set, the zone is stored to the zone database configured at *zone-db-listen* and updated there with every change to the zone contents. The value of this option specifies the zone instance number (from 1 to 8 inclusive) within the database to write to.

Default: `-1` (disabled)

7.22.7 master

An ordered list of references *remote* and *remotes* to zone primary servers (formerly known as master servers). Empty value is allowed for template value overriding.

Default: not set

7.22.8 ddns-master

A *reference* to a zone primary master where DDNS messages should be forwarded to. If not specified, the first *master* server is used.

If set to the empty value (""), incoming DDNS messages aren't forwarded but are applied to the local zone instead, no matter if it is a secondary server. This is only allowed in combination with *dnssec-signing* enabled.

Default: not set

7.22.9 notify

An ordered list of references *remote* and *remotes* to secondary servers to which NOTIFY message is sent if the zone changes. Empty value is allowed for template value overriding.

Default: not set

7.22.10 notify-delay

A time delay in seconds before an outgoing NOTIFY message is sent. This delay also defines the time granularity at which NOTIFY messages are sent per zone.

Default: `0`

7.22.11 update-delay

A time delay in seconds before a change to zone contents is made after an external trigger such as incoming NOTIFY or DDNS, or an internal trigger from different zone such as change to zone to be *reversed*, *included from* or a member of generated catalog zone.

Exception: zone changing events triggered by control socket (knotc zone-* commands) or by interpreted catalog are performed immediately, without configured delay.

Default: `0`

7.22.12 acl

An ordered list of *references* to ACL rules which can allow or disallow zone transfers, updates or incoming notifies.

Default: not set

7.22.13 master-pin-tolerance

If set to a nonzero value on a secondary, always request AXFR/IXFR from the same primary as the last time, effectively pinning one primary. Only when another primary is updated and the current one lags behind for the specified amount of time (defined by this option in seconds), change to the updated primary and force AXFR.

This option is useful when multiple primaries may have different zone history in their journals, making it unsafe to combine interchanged IXFR from different primaries.

Default: `0` (disabled)

7.22.14 provide-ixfr

If disabled, the server is forced to respond with AXFR to IXFR queries. If enabled, IXFR requests are responded normally.

Default: on

7.22.15 semantic-checks

Selects if extra zone semantic checks are used or impacts of the mandatory checks.

There are several mandatory checks which are always enabled and cannot be turned off. An error in a mandatory check causes the zone not to be loaded. Most of the mandatory checks can be weakened by setting `soft`, which allows the zone to be loaded even if the check fails.

If enabled, extra checks are used. These checks don't prevent the zone from loading.

The mandatory checks are applied to zone files, zone transfers, and updates via control interface. The extra checks are applied to zone files only!

Mandatory checks:

- Missing SOA record at the zone apex (**RFC 1034**) (*)
- An extra record exists together with a CNAME record except for RRSIG and NSEC (**RFC 1034**)
- Multiple CNAME records with the same owner exist (**RFC 1034**)
- DNAME record having a record under it (**RFC 6672**)
- Multiple DNAME records with the same owner exist (**RFC 6672**)
- NS record exists together with a DNAME record (**RFC 6672**)
- DS record exists at the zone apex (**RFC 3658**)

(*) The marked check can't be weakened by the soft mode. All other mandatory checks are subject to the optional soft mode.

Extra checks:

- Missing NS record at the zone apex
- Missing glue A or AAAA record
- Invalid DS or NSEC3PARAM record
- CDS or CDNSKEY inconsistency
- All other DNSSEC checks executed during *dnssec-validation*

Note

The soft mode allows the refresh event to ignore a CNAME response to a SOA query (malformed message) and triggers a zone bootstrap instead.

Default: `off`

7.22.16 default-ttl

The default TTL value if none is specified in a zone file or zone insertion using the dynamic configuration.

Warning

As changing this value can result in differently parsed zone file(s), the corresponding zone SOA serial(s) should be incremented before reloading or committing the configuration. Alternatively, setting *zonefile-load* to `difference-no-serial` ensures the resulting zone(s) update is correct.

Default: `3600`

7.22.17 zonefile-sync

The time in seconds after which the current zone in memory will be synced with a zone file on the disk (see *file*). The server will serve the latest zone even after a restart using zone journal, but the zone file on the disk will only be synced after `zonefile-sync` time has expired (or after manual zone flush). This is applicable when the zone is updated via IXFR, DDNS or automatic DNSSEC signing. In order to completely disable automatic zone file synchronization, set the value to `-1`. In that case, it is still possible to force a manual zone flush using the `-f` option.

Note

If you are serving large zones with frequent updates where the immediate sync with a zone file is not desirable, increase the value.

Default: `0` (immediate)

7.22.18 zonefile-load

Selects how the zone file contents are applied during zone load.

Possible values:

- `none` – The zone file is not used at all.
- `difference` – If the zone contents are already available during server start or reload, the difference is computed between them and the contents of the zone file. This difference is then checked for semantic errors and applied to the current zone contents.
- `difference-no-serial` – Same as `difference`, but the SOA serial in the zone file is ignored, the server takes care of incrementing the serial automatically.
- `whole` – Zone contents are loaded from the zone file.

When `difference` is configured and there are no zone contents yet (cold start and no zone contents in the journal), it behaves the same way as `whole`.

Default: `whole`

Note

See *Handling zone file, journal, changes, serials* for guidance on configuring these and related options to ensure reliable operation.

Warning

If *zone-db-input* is configured, the textual zone file is never loaded. However, this option still effects on how the zone contents loaded from the database are applied and handled.

7.22.19 zonefile-skip

Specifies resource record types to be omitted when loading and syncing zone files.

Resource record types are represented as strings (e.g. "DS") and multiple types may be specified. The special string `dnssec` represents all types usually created by DNSSEC signing routines (DNSKEY, RRSIG, NSEC, NSEC3, NSEC3PARAM, CDNSKEY, CDS — but not DS).

Note

This option takes effect while a zone file is being read or written, but it does not directly trigger any zone file operation. Therefore, the configured record types cannot be expected to disappear from the zone file or running zone immediately after setting the option.

Default: not set

7.22.20 journal-content

Selects how the journal shall be used to store zone and its changes.

Possible values:

- `none` – The journal is not used at all.
- `changes` – Zone changes history is stored in journal.
- `all` – Zone contents and history is stored in journal.

Default: `changes`

Warning

When this option is changed, the journal still contains data respective to the previous setting. For example, changing it to `none` does not purge the journal. Also, changing it from `all` to `changes` does not cause the deletion of the zone-in-journal and the behaviour of the zone loading procedure might be different than expected. It is recommended to consider purging the journal when this option is changed.

7.22.21 journal-max-usage

Policy how much space in journal DB will the zone's journal occupy.

Note

Journal DB may grow far above the sum of journal-max-usage across all zones, because of DB free space fragmentation.

Default: `100M` (100 MiB)

7.22.22 journal-max-depth

Maximum history length of the journal.

Note

Zone-in-journal changeset isn't counted to the limit.

Minimum: `2`

Default: `20`

7.22.23 ixfr-benevolent

If enabled, incoming IXFR is applied even when it contains removals of non-existing or additions of existing records.

Default: `off`

7.22.24 ixfr-by-one

Within incoming IXFR, process only one changeset at a time, not multiple together. This preserves the complete history in the journal and prevents the merging of changesets when multiple changesets are IXFRed simultaneously. However, this does not prevent the merging (or deletion) of old changesets in the journal to save space, as described in *journal behaviour*.

This option leads to increased server load when processing IXFR, including network traffic.

Default: `off`

7.22.25 ixfr-from-axfr

If a primary sends AXFR-style-IXFR upon an IXFR request, compute the difference and process it as an incremental zone update (e.g. by storing the changeset in the journal).

Default: `off`

7.22.26 zone-max-size

Maximum size of the zone. The size is measured as size of the zone records in wire format without compression. The limit is enforced for incoming zone transfers and dynamic updates.

For incremental transfers (IXFR), the effective limit for the total size of the records in the transfer is twice the configured value. However the final size of the zone must satisfy the configured value.

Default: unlimited

7.22.27 adjust-threads

Parallelize internal zone adjusting procedures by using specified number of threads. This is useful with huge zones with NSEC3. Speedup observable at server startup and while processing NSEC3 re-salt.

Default: `1` (no extra threads)

7.22.28 external-validation

A *reference* to external validation section.

If configured, every change to the zone (zone file update, incoming IXFR/AXFR, dynamic update, and DNSSEC re-signing, but not changes over control socket – `knotc zone-begin`) is paused just before applying the new zone. At that point, validation and confirmation is awaited from the user (or potentially a user-defined script).

Note

In the case of server shutdown or configuration reload, the server waits until all outstanding external validations are either committed or aborted, or until the *timeout* elapses.

In the referenced `external` section, it is possible to define paths to files where the new zone contents and/or differences are written (in the zone file format) just before every validation.

Tip

If *dbus-event* is set to `external-verify`, a corresponding signal is emitted when the server is awaiting external validation.

Default: none

7.22.29 dnssec-signing

If enabled, automatic DNSSEC signing for the zone is turned on.

Default: `off`

7.22.30 dnssec-validation

If enabled, the zone contents are validated for being correctly signed (including NSEC/NSEC3 chain) with DNSSEC signatures every time the zone is loaded or changed (including AXFR/IXFR).

When the validation fails, the zone being loaded or update being applied is cancelled with an error, and either none or previous zone state is published.

List of DNSSEC checks:

- Every zone RRSet is correctly signed by at least one present DNSKEY.
- For every RRSIG there are at most 3 non-matching DNSKEYs with the same keytag.
- DNSKEY RRSet is signed by KSK.
- NSEC(3) RR exists for each name (unless opt-out) with correct bitmap.
- Every NSEC(3) RR is linked to the lexicographically next one.

The validation is not affected by *dnssec-policy* configuration, except for *signing-threads* option, which specifies the number of threads for parallel validation, and *rrsig-refresh*, which defines minimal allowed remaining RRSIG validity (otherwise a warning is logged).

Note

Redundant or garbage NSEC3 records are ignored.

This mode is not compatible with *dnssec-signing*.

Tip

If *dbus-event* is set to `dnssec-invalid`, a corresponding signal is emitted when the validation fails.

Default: not set

7.22.31 dnssec-policy

A *reference* to DNSSEC signing policy.

Note

A configured policy called "default" won't be used unless explicitly referenced.

Default: an imaginary policy with all default values

7.22.32 ds-push

Per zone configuration of *ds-push*. This option overrides possible per policy option. Empty value is allowed for template value overriding.

Default: not set

7.22.33 zonemd-verify

On each zone load/update, verify that ZONEMD is present in the zone and valid.

Note

Zone digest calculation may take much time and CPU on large zones.

Tip

If *dbus-event* is set to `dnssec-invalid`, a corresponding signal is emitted when the verification fails.

Default: `off`

7.22.34 zonemd-generate

On each zone update, calculate ZONEMD and put it into the zone.

Possible values:

- `none` – No action regarding ZONEMD.
- `zonemd-sha384` – Generate ZONEMD using SHA384 algorithm.
- `zonemd-sha512` – Generate ZONEMD using SHA512 algorithm.
- `remove` – Remove any ZONEMD from the zone apex.

Default: `none`

7.22.35 serial-policy

Specifies how the zone serial is updated after a dynamic update or automatic DNSSEC signing. If the serial is changed by the dynamic update, no change is made.

Possible values:

- `increment` – The serial is incremented according to serial number arithmetic.
- `unixtime` – The serial is set to the current unix time.
- `dateserial` – The 10-digit serial (YYYYMMDDnn) is incremented, the first 8 digits match the current iso-date.

Note

If the resulting serial for `unixtime` or `dateserial` is lower than or equal to the current serial (this happens e.g. when migrating from other policy or frequent updates), the serial is incremented instead.

To avoid user confusion, use `dateserial` only if you expect at most 100 updates per day per zone and `unixtime` only if you expect at most one update per second per zone.

Generated catalog zones use `unixtime` only.

Default: `increment` (`unixtime` for generated catalog zones)

7.22.36 serial-modulo

The option value is a string consisting of two parts (with no separator between them), each of which is optional.

The first part specifies that the zone serials must be congruent modulo the specified value. The format is `R/M`, where `R < M <= 256` are positive integers. Whenever the zone serial is incremented, it is ensured that `serial`

`% M == R`. This can be useful in the case of multiple inconsistent primaries, where distinct zone serial sequences prevent cross-master-IXFR by any secondary.

Note

Because the zone serial effectively always increments by `M` instead of `1`, it is not recommended to use `dateserial` or even `unixtime` *serial-policy* in the case of rapidly updated zone.

The second part specifies a numeric shift for the generated zone serial. The shift is formatted as a signed integer, including the sign (`+` or `-`). It is mostly useful with `unixtime` *serial-policy*, where the generated zone serial is shifted relative to the Unix time.

Note

In order to ensure the congruent policy, this option is only allowed with *DNSSEC signing enabled* and *zonefile-load* to be either `difference-no-serial` or `none`.

Default: `0/1+0`

7.22.37 reverse-generate

A list of zone names for which automatic generation of reverse PTR records based on A/AAAA records is enabled. The entire generated zone is automatically stored in the journal.

The auto-generated reverse zone is re-generated whenever any of the specified zones is updated. This includes the situation when reverse generation had failed due to some of the specified zones were not yet loaded or had expired.

Current limitations:

- Is slow for large zones (even when changing a little).
- Recomputes all reverse records upon any change in any of the reversed zones.

In case of secondary zone (i.e. *master* is specified) this option implies *ixfr-from-axfr*: *on* and *journal-content*: *all*, otherwise *zonefile-load*: *difference-no-serial* and *journal-content*: *all*.

Default: none

7.22.38 include-from

A list of subzones that should be flattened into this zone. The flattening deletes all delegation-related records (including NS, SOA, ...) from both zones and copies all other records from the subzone to this zone.

In case of secondary zone (i.e. *master* is specified) this option implies *ixfr-from-axfr*: *on* and *journal-content*: *all*, otherwise *zonefile-load*: *difference-no-serial* and *journal-content*: *all*.

Default: none

7.22.39 refresh-min-interval

Forced minimum zone refresh interval (in seconds) to avoid flooding primary server.

Minimum: `2`

Default: `2`

7.22.40 refresh-max-interval

Forced maximum zone refresh interval (in seconds).

Default: not set

7.22.41 retry-min-interval

Forced minimum zone retry interval (in seconds) to avoid flooding primary server.

Minimum: 1

Default: 1

7.22.42 retry-max-interval

Forced maximum zone retry interval (in seconds).

Default: not set

7.22.43 expire-min-interval

Forced minimum zone expire interval (in seconds) to avoid flooding primary server.

Minimum: 3

Default: 3

7.22.44 expire-max-interval

Forced maximum zone expire interval (in seconds).

Default: not set

7.22.45 catalog-role

Trigger zone catalog feature. Possible values:

- `none` – Not a catalog zone.
- `interpret` – A catalog zone which is loaded from a zone file or XFR, and member zones shall be configured based on its contents.
- `generate` – A catalog zone whose contents are generated according to assigned member zones.
- `member` – A member zone that is assigned to one generated catalog zone.

> **Note**
>
> If set to `generate`, the *zonefile-load* option has no effect since a zone file is never loaded.

Default: `none`

7.22.46 catalog-template

For the catalog member zones, the specified configuration template will be applied.

Multiple catalog templates may be defined. The first one is used unless the member zone has the *group* property defined, matching another catalog template.

> **Note**
>
> This option must be set if and only if *catalog-role* is *interpret*.
>
> Nested catalog zones aren't supported. Therefore catalog templates can't contain *catalog-role* set to `interpret` or `generate`.

Default: not set

7.22.47 catalog-zone

Assign this member zone to specified generated catalog zone.

> **Note**
>
> This option must be set if and only if *catalog-role* is *member*.
>
> The referenced catalog zone must exist and have *catalog-role* set to *generate*.

Default: not set

7.22.48 catalog-group

Assign this member zone to specified catalog group (configuration template).

> **Note**
>
> This option has effect if and only if *catalog-role* is *member*.

Default: not set

7.22.49 module

An ordered list of references to query modules in the form of *module_name* or *module_name/module_id*. These modules apply only to the current zone queries.

Default: not set

CHAPTER
EIGHT

MODULES

8.1 authsignal – Automatic Authenticated DNSSEC Bootstrapping records

This module is able to synthesize records for automatic DNSSEC bootstrapping (**RFC 9615**).

Records are synthesized only if the query can't be satisfied from the zone.

Synthesized records also need to be signed. Typically, this can be done using the *onlinesign* module, as shown below.

8.1.1 Example

Automatic forward records

```
mod-onlinesign:
  - id: authsignal
    nsec-bitmap: [CDS, CDNSKEY]

zone:
  - domain: example.net
    dnssec-signing: on
  - domain: _signal.ns1.example.com
    module: [mod-authsignal, mod-onlinesign/authsignal]
```

Result:

```
$ kdig CDS _dsboot.example.net._signal.ns1.example.com.
...
;; QUESTION SECTION:
;; _dsboot.example.net._signal.ns1.example.com.      IN      CDS

;; ANSWER SECTION:
_dsboot.example.net._signal.ns1.example.com. 0       IN      CDS     45504 13 2 2F2D518FD9DBB2B1403F51398A9931F2832B89F0F85C146B130D383FC23584FA
```

8.2 cookies — DNS Cookies

DNS Cookies (**RFC 7873**) is a lightweight security mechanism against denial-of-service and amplification attacks. The server keeps a secret value (the Server Secret), which is used to generate a cookie, which is sent to the client in the OPT RR. The server then verifies the authenticity of the client by the presence of a correct cookie. Both the server and the client have to support DNS Cookies, otherwise they are not used.

Note

This module introduces two statistics counters:

- `presence` – The number of queries containing the COOKIE option.
- `dropped` – The number of dropped queries due to the slip limit.

8.2.1 Example

It is recommended to enable DNS Cookies globally, not per zone. The module may be used without any further configuration.

```
template:
    - id: default
      global-module: mod-cookies  # Enable DNS Cookies globally with defaults
```

Module configuration may be supplied if necessary.

```
mod-cookies:
  - id: custom
    badcookie-slip: 3  # The server replies only to every third query with a wrong␣
↪cookie
    secret: 0xdeadbeefdeadbeefdeadbeefdeadbeef  # Explicit Server Secret

template:
  - id: default
    global-module: mod-cookies/custom  # Enable DNS Cookies globally with custom␣
↪settings
```

8.2.2 Module reference

```
mod-cookies:
  - id: STR
    secret-lifetime: TIME
    badcookie-slip: INT
    secret: STR | HEXSTR ...
```

id

A module identifier.

secret-lifetime

This option configures in seconds how often the Server Secret is regenerated. The maximum allowed value is 36 days (**RFC 7873 Section 7.1**).

Default: `26h` (26 hours)

badcookie-slip

This option configures how often the server responds to queries containing an invalid cookie by sending them the correct cookie.

- The value **1** means that the server responds to every query.
- The value **2** means that the server responds to every second query with an invalid cookie, the rest of the queries is dropped.
- The value **N > 2** means that the server responds to every Nth query with an invalid cookie, the rest of the queries is dropped.

Default: 1

secret

Use this option to set the Server Secret manually. If this option is used, the Server Secret remains the same until changed manually and the *secret-lifetime* option is ignored. The size of the Server Secret currently MUST BE 16 bytes, or 32 hexadecimal characters.

It's possible to specify a second Server Secret, which is used for the fallback when the cookie verification with the first Server Secret fails (secret rollover).

Default: not set

8.3 `dnsproxy` – Tiny DNS proxy

The module forwards all queries, or all specific zone queries if configured per zone, to the indicated server for resolution. If configured in the fallback mode, only locally unsatisfied queries are forwarded. I.e. a tiny DNS proxy. There are several uses of this feature:

- A substitute public-facing server in front of the real one
- Local zones (poor man's "views"), rest is forwarded to the public-facing server
- Using the fallback to forward queries to a resolver
- etc.

Note

The module does not alter the query/response as the resolver would, and the original transport protocol (UDP or TCP) is kept as well.

8.3.1 Example

The configuration is straightforward and just a single remote server is required:

```
remote:
  - id: hidden
    address: 10.0.1.1

mod-dnsproxy:
  - id: default
    remote: hidden
    fallback: on

template:
  - id: default
    global-module: mod-dnsproxy/default

zone:
  - domain: local.zone
```

When a client queries anything in the `local.zone` (which must exist), it will receive a local response. All other requests (to unknown zones) will be forwarded to the specified server (`10.0.1.1` in this case).

8.3.2 Module reference

```
mod-dnsproxy:
  - id: STR
    remote: remote_id
    timeout: INT
    address: ADDR[/INT] | ADDR-ADDR | STR ...
    fallback: BOOL
    tcp-fastopen: BOOL
    catch-nxdomain: BOOL
```

id

A module identifier.

remote

A *reference* to a remote server where the queries are forwarded to.

Required

> **Note**
>
> If the remote has more addresses configured, other addresses are used sequentially as fallback. In this case, for the N-th address the N-th via address is taken if configured.

timeout

A remote response timeout in milliseconds.

Default: `500` (milliseconds)

address

An ordered list of IP addresses, absolute UNIX socket paths, network subnets, or network ranges. If the query's source address does not fall into any of the configured ranges, the query isn't forwarded.

Default: not set

fallback

If enabled, locally unsatisfied queries leading to REFUSED (no zone) are forwarded. If disabled, all queries are directly forwarded without any local attempts to resolve them.

Default: `on`

tcp-fastopen

If enabled, TCP Fast Open is used when forwarding TCP queries.

Default: `off`

catch-nxdomain

If enabled, locally unsatisfied queries leading to NXDOMAIN are forwarded. This option is only relevant in the fallback mode.

Default: `off`

8.4 `dnstap` – Dnstap traffic logging

A module for query and response logging based on the dnstap library. You can capture either all or zone-specific queries and responses; usually you want to do the former.

8.4.1 Example

The configuration comprises only a *sink* path parameter, which can be either a file, a UNIX socket, or a TCP address:

```
mod-dnstap:
  - id: capture_all
    sink: /tmp/capture.tap

template:
  - id: default
    global-module: mod-dnstap/capture_all
```

Note

To be able to use a Unix socket you need an external program to create it. Knot DNS connects to it as a client using the libfstrm library. It operates exactly like syslog.

Note

Dnstap log files can also be created or read using *kdig*.

8.4.2 Module reference

For all queries logging, use this module in the *default* template. For zone-specific logging, use this module in the proper zone configuration.

```
mod-dnstap:
  - id: STR
    sink: STR
    identity: STR
    version: STR
    log-queries: BOOL
    log-responses: BOOL
    responses-with-queries: BOOL
```

id

A module identifier.

sink

A sink path, which can be either a file, a UNIX socket when prefixed with `unix:`, or a TCP *address@port* when prefixed with `tcp:`. The file may be specified as an absolute path or a path relative to the *knotd* startup directory.

Required

Warning

File is overwritten on server startup or reload.

identity

A DNS server identity. Set empty value to disable.

Default: FQDN hostname

version

A DNS server version. Set empty value to disable.

Default: server version

log-queries

If enabled, query messages will be logged.

Default: `on`

log-responses

If enabled, response messages will be logged.

Default: `on`

responses-with-queries

If enabled, dnstap `AUTH_RESPONSE` messages will also include the original query message as well as the response message sent by the server.

Default: `off`

8.5 `geoip` — Geography-based responses

This module offers response tailoring based on client's subnet, geographic location, or a statistical weight. It supports GeoIP databases in the MaxMind DB format, such as GeoIP2 or the free version GeoLite2.

The module can be enabled only per zone.

> **Note**
>
> If *EDNS Client Subnet* support is enabled and if a query contains this option, the module takes advantage of this information to provide a more accurate response.

8.5.1 DNSSEC support

There are several ways to enable DNSSEC signing of tailored responses.

Full zone signing

If *automatic DNSSEC signing* is enabled, the whole zone is signed by the server and all alternative RRsets, which are responded by the module, are pre-signed when the module is loaded.

This has a speed benefit, however note that every RRset configured in the module should have a **default** RRset of the same type contained in the zone, so that the NSEC(3) chain can be built correctly. Also, it is STRONGLY RECOMMENDED to use *manual key management* in this setting, as the corresponding zone has to be reloaded when the signing key changes and to have better control over key synchronization to all instances of the server.

> **Note**
>
> DNSSEC keys for computing record signatures MUST exist in the KASP database or be generated before the module is launched, otherwise the module fails to compute the signatures and does not load.

Module signing

If *automatic DNSSEC signing* is disabled, it's possible to combine externally pre-signed zone with module presigning of the alternative RRsets when the module is loaded. In this mode, only ZSK has to be present in the KASP database. Also in this mode every RRset configured in the module should have a **default** RRset of the same type contained in the zone.

Example:

```
policy:
  - id: presigned_zone
    manual: on
    unsafe-operation: no-check-keyset

mod-geoip:
  - id: geo_dnssec
    ...
    dnssec: on
    policy: presigned_zone

zone:
  - domain: example.com.
    module: mod-geoip/geo_dnssec
```

Online signing

Alternatively, the *geoip* module may be combined with the *onlinesign* module and the tailored responses can be signed on the fly. This approach is much more computationally demanding for the server.

Note

If the GeoIP module is used with online signing, it is recommended to set the *nsec-bitmap* option of the onlinesign module to contain all Resource Record types potentially generated by the module.

8.5.2 Example

An example configuration:

```
mod-geoip:
  - id: default
    config-file: /path/to/geo.conf
    ttl: 20
    mode: geodb
    geodb-file: /path/to/GeoLite2-City.mmdb
    geodb-key: [ country/iso_code, city/names/en ]

zone:
  - domain: example.com.
    module: mod-geoip/default
```

8.5.3 Configuration file

Every instance of the module requires an additional *config-file* in which the desired responses to queries from various locations are configured. This file has the following simple format:

```
domain-name1:
  - geo|net|weight: value1
```

(continues on next page)

(continued from previous page)

```
    RR-Type1: RDATA
    RR-Type2: RDATA
    ...
  - geo|net|weight: value2
    RR-Type1: RDATA
  ...
domain-name2:
...
```

8.5.4 Module configuration examples

This section contains some examples for the module's *config-file*.

Using subnets

```
foo.example.com:
  - net: 10.0.0.0/24
    A: [ 192.168.1.1, 192.168.1.2 ]
    AAAA: [ 2001:DB8::1, 2001:DB8::2 ]
    TXT: "subnet\ 10.0.0.0/24"
  ...
bar.example.com:
  - net: 2001:DB8::/32
    A: 192.168.1.3
    AAAA: 2001:DB8::3
    TXT: "subnet\ 2001:DB8::/32"
...
```

Clients from the specified subnets will receive the responses defined in the module config. Others will receive the default records defined in the zone (if any).

Note

If a space or a quotation mark is a part of record data, such a character must be prefixed with a backslash. The following notations are equivalent:

```
Multi-word\ string
"Multi-word\ string"
"\"Multi-word string\""
```

Using geographic locations

```
foo.example.com:
  - geo: "CZ;Prague"
    CNAME: cz.foo.example.com.
  - geo: "US;Las Vegas"
    CNAME: vegas.foo.example.net.
  - geo: "US;*"
    CNAME: us.foo.example.net.
...
```

Clients from the specified geographic locations will receive the responses defined in the module config. Others will receive the default records defined in the zone (if any). See *geodb-key* for the syntax and semantics of the location definitions.

Using weighted records

```
foo.example.com:
  - weight: 1
    CNAME: canary.foo.example.com.
  - weight: 10
    CNAME: prod1.foo.example.com.
  - weight: 10
    CNAME: prod2.foo.example.com.
...
```

Each response is generated through a random pick where each defined record has a likelihood of its weight over the sum of all weights for the requested name to. Records defined in the zone itself (if any) will never be served.

Result:

```
$ for i in $(seq 1 100); do kdig @192.168.1.242 CNAME foo.example.com +short; done |␣
↪sort | uniq -c
   3 canary.foo.example.com.foo.example.com.
  52 prod1.foo.example.net.foo.example.com.
  45 prod2.foo.example.net.foo.example.com.
```

8.5.5 Module reference

```
mod-geoip:
  - id: STR
    config-file: STR
    ttl: TIME
    mode: geodb | subnet | weighted
    dnssec: BOOL
    policy: policy_id
    geodb-file: STR
    geodb-key: STR ...
```

id

A module identifier.

config-file

A path to the response configuration file as described above. A non-absolute path is relative to the *knotd* startup directory.

Required

ttl

The time to live of Resource Records returned by the module, in seconds.

Default: `60`

mode

The mode of operation of the module.

Possible values:

- `subnet` – Responses are tailored according to subnets.
- `geodb` – Responses are tailored according to geographic data retrieved from the configured database.
- `weighted` – Responses are tailored according to a statistical weight.

Default: `subnet`

dnssec

If explicitly enabled, the module signs positive responses based on the module policy (*policy*). If explicitly disabled, positive responses from the module are not signed even if the zone is pre-signed or signed by the server (*dnssec-signing*).

> **Warning**
>
> This configuration must be used carefully. Otherwise the zone responses can be bogus. DNSKEY rotation isn't supported. So *manual* mode is highly recommended.

Default: current value of *dnssec-signing* with *dnssec-policy*

policy

A *reference* to DNSSEC signing policy which is used if *dnssec* is enabled.

Default: an imaginary policy with all default values

geodb-file

A path to a .mmdb file containing the GeoIP database. A non-absolute path is relative to the *knotd* startup directory.

Required if mode is set to **geodb**

geodb-key

Multi-valued item, can be specified up to **8** times. Each **geodb-key** specifies a path to a key in a node in the supplied GeoIP database. The module currently supports two types of values: **string** or **32-bit unsigned int**. In the latter case, the key has to be prefixed with **(id)**. Common choices of keys include:

- **continent/code**
- **country/iso_code**
- **(id)country/geoname_id**
- **city/names/en**
- **(id)city/geoname_id**
- **isp**
- ...

The exact keys available depend on the database being used. To get the full list of keys available, you can e.g. do a sample lookup on your database with the mmdblookup tool.

In the zone's config file for the module the values of the keys are entered in the same order as the keys in the module's configuration, separated by a semicolon. Enter the value **"*"** if the key is allowed to have any value.

8.6 `noudp` — No UDP response

The module sends empty truncated reply to a query over UDP. Replies over TCP are not affected.

8.6.1 Example

To enable this module for all configured zones and every UDP reply:

```
template:
  - id: default
    global-module: mod-noudp
```

Or with specified UDP allow rate:

```
mod-noudp:
  - id: sometimes
    udp-allow-rate: 1000  # Don't truncate every 1000th UDP reply

template:
  - id: default
    module: mod-noudp/sometimes
```

8.6.2 Module reference

```
mod-noudp:
 - id: STR
   udp-allow-rate: INT
   udp-truncate-rate: INT
```

Note

Both *udp-allow-rate* and *udp-truncate-rate* cannot be specified together.

udp-allow-rate

Specifies frequency of UDP replies that are not truncated. A non-zero value means that every Nth UDP reply is not truncated.

Note

The rate value is associated with one UDP worker. If more UDP workers are configured, the specified value may not be obvious to clients.

Default: not set

udp-truncate-rate

Specifies frequency of UDP replies that are truncated (opposite of *udp-allow-rate*). A non-zero value means that every Nth UDP reply is truncated.

Note

The rate value is associated with one UDP worker. If more UDP workers are configured, the specified value may not be obvious to clients.

Default: 1

8.7 `onlinesign` — Online DNSSEC signing

The module provides online DNSSEC signing. Instead of pre-computing the zone signatures when the zone is loaded into the server or instead of loading an externally signed zone, the signatures are computed on-the-fly during answering.

The main purpose of the module is to enable authenticated responses with zones which use other dynamic module (e.g., automatic reverse record synthesis) because these zones cannot be pre-signed. However, it can be also used as a simple signing solution for zones with low traffic and also as a protection against zone content enumeration (zone walking).

In order to minimize the number of computed signatures per query, the module produces a bit different responses from the responses that would be sent if the zone was pre-signed. Still, the responses should be perfectly valid for a DNSSEC validating resolver.

Differences from statically signed zones:

- The NSEC records are constructed as Minimally Covering NSEC Records (**RFC 7129 Appendix A**). Therefore the generated domain names cover the complete domain name space in the zone's authority.
- NXDOMAIN responses are promoted to NODATA responses. The module proves that the query type does not exist rather than that the domain name does not exist.
- Domain names matching a wildcard are expanded. The module pretends and proves that the domain name exists rather than proving a presence of the wildcard.

Records synthesized by the module:

- DNSKEY record is synthesized in the zone apex and includes public key material for the active signing key.
- NSEC records are synthesized as needed.
- RRSIG records are synthesized for authoritative content of the zone.
- CDNSKEY and CDS records are generated as usual to publish valid Secure Entry Point.

Limitations:

- Due to limited interaction between the server and the module, after any change to KASP DB (including *knotc zone-ksk-submitted* command) or when a scheduled DNSSEC event shall be processed (e.g. transition to next DNSKEY rollover state) the server must be reloaded or queried to the zone (with the DO bit set) to apply the change or to trigger the event. For optimal operation, the recommended query frequency is at least ones per second for each zone configured.
- The NSEC records may differ for one domain name if queried for different types. This is an implementation shortcoming as the dynamic modules cooperate loosely. Possible synthesis of a type by other module cannot be predicted. This dissimilarity should not affect response validation, even with validators performing aggressive negative caching (**RFC 8198**).
- The module isn't compatible with the Offline KSK mode yet.

Recommendations:

- Configure the module with an explicit signing policy which has the *rrsig-lifetime* value in the order of hours.
- Note that *single-type-signing* should be set explicitly to avoid fallback to backward-compatible default.

8.7.1 Example

- Enable the module in the zone configuration with the default signing policy:

```
zone:
  - domain: example.com
    module: mod-onlinesign
```

  Or with an explicit signing policy:

```
policy:
  - id: rsa
    algorithm: RSASHA256
```

(continues on next page)

(continued from previous page)

```
    ksk-size: 2048
    rrsig-lifetime: 25h
    rrsig-refresh: 20h

mod-onlinesign:
  - id: explicit
    policy: rsa

zone:
  - domain: example.com
    module: mod-onlinesign/explicit
```

Or use manual policy in an analogous manner, see *Manual key management*.

- Make sure the zone is not signed and also that the automatic signing is disabled. All is set, you are good to go. Reload (or start) the server:

```
$ knotc reload
```

The following example stacks the online signing with reverse record synthesis module:

```
mod-synthrecord:
  - id: lan-forward
    type: forward
    prefix: ip-
    ttl: 1200
    network: 192.168.100.0/24

zone:
  - domain: corp.example.net
    module: [mod-synthrecord/lan-forward, mod-onlinesign]
```

8.7.2 Module reference

```
mod-onlinesign:
  - id: STR
    policy: policy_id
    nsec-bitmap: STR ...
```

id

A module identifier.

policy

A *reference* to DNSSEC signing policy. A special *default* value can be used for the default policy setting.

Default: an imaginary policy with all default values

nsec-bitmap

A list of Resource Record types included in an NSEC bitmap generated by the module. This option should reflect zone contents or synthesized responses by modules, such as *synthrecord* and *GeoIP*.

Default: `[A, AAAA]`

8.8 probe — DNS traffic probe

The module allows the server to send simplified information about regular DNS traffic through *UNIX* sockets. The exported information consists of data blocks where each data block (datagram) describes one query/response pair. The response part can be empty. The receiver can be an arbitrary program using *libknot* interface (C or Python). In case of high traffic, more channels (sockets) can be configured to allow parallel processing.

Tip

A simple probe client in Python.

8.8.1 Example

Default module configuration:

```
template:
  - id: default
    global-module: mod-probe
```

Per zone probe with 8 channels and maximum 1M logs per second limit:

```
mod-probe:
  - id: custom
    path: /tmp/knot-probe
    channels: 8
    max-rate: 1000000

zone:
  - domain: example.com.
    module: mod-probe/custom
```

8.8.2 Module reference

```
mod-probe:
  - id: STR
    path: STR
    channels: INT
    max-rate: INT
```

id

A module identifier.

path

A directory path where the UNIX sockets are located. A non-absolute path is relative to the *knotd* startup directory.

Note

It's recommended to use a directory with the execute permission restricted to the intended probe consumer process owner only.

Default: *rundir*

channels

Number of channels (UNIX sockets) the traffic is distributed to. In case of high DNS traffic which is beeing processed by many UDP/XDP/TCP workers, using more channels reduces the module overhead.

Default: 1

max-rate

Maximum number of queries/replies per second the probe is allowed to transfer. If the limit is exceeded, the over-limit traffic is ignored. Zero value means no limit.

Default: `100000` (one hundred thousand)

8.9 `queryacl` — Limit queries by remote address or target interface

This module provides a simple way to whitelist incoming queries according to the query's source address or target interface. It can be used e.g. to create a restricted-access subzone with delegations from the corresponding public zone. The module may be enabled both globally and per-zone.

> **Note**
>
> The module limits only regular queries. Notify, transfer and update are handled by *ACL*.

8.9.1 Example

```
mod-queryacl:
  - id: default
    address: [192.0.2.73-192.0.2.90, 203.0.113.0/24]
    interface: 198.51.100

zone:
  - domain: example.com
    module: mod-queryacl/default
```

8.9.2 Module reference

```
mod-queryacl:
  - id: STR
    address: ADDR[/INT] | ADDR-ADDR | STR ...
    interface: ADDR[/INT] | ADDR-ADDR | STR ...
```

id

A module identifier.

address

An ordered list of IP addresses, absolute UNIX socket paths, network subnets, or network ranges. If the query's address does not fall into any of the configured ranges, NOTAUTH rcode is returned.

Default: not set

interface

An ordered list of IP addresses, absolute UNIX socket paths, network subnets, or network ranges. If the interface does not fall into any of the configured ranges, NOTAUTH rcode is returned. Note that every interface used has to be configured in *listen*.

Note

Don't use values *0.0.0.0* and *::0*. These values are redundant and don't work as expected.

Default: not set

8.10 `rrl` — Response rate limiting

Response rate limiting (RRL) is a method to combat DNS reflection amplification attacks. These attacks rely on the fact that the source address of a UDP query can be forged, and without a worldwide deployment of BCP38, such a forgery cannot be prevented. An attacker can use a DNS server (or multiple servers) as an amplification source to flood a victim with a large number of unsolicited DNS responses. RRL lowers the amplification factor of these attacks by sending some responses as truncated or by dropping them altogether.

This module can also help protect the server from excessive utilization by limiting incoming packets (including handshakes) based on consumed time. If a packet is time rate limited, it's dropped. This function works with all supported non-UDP transport protocols (TCP, TLS, and QUIC) and cannot be configured per zone.

Note

This module introduces three statistics counters:

- `slipped` – The number of slipped UDP responses.
- `dropped` – The number of dropped UDP responses due to the rate limit.
- `dropped-time` – The number of dropped non-UDP packets due to the time rate limit.

Note

If the *Cookies* module is active and configured before this module, RRL is not applied to UDP responses with a valid DNS cookie.

Note

The time limiting applies even to handshakes of incoming authorized requests (e.g. NOTIFY, AXFR). In such cases, setting *whitelist* or reusing already established connections (e.g. *remote-pool-timeout* on the remote server) can mitigate this issue.

8.10.1 Example

You can enable RRL by setting the module globally

```
template:
  - id: default
    global-module: mod-rrl  # Default module configuration
```

or per zone

```
mod-rrl:
  - id: custom
    rate-limit: 200

zone:
```

(continues on next page)

(continued from previous page)

```
  - domain: example.com
    module: mod-rrl/custom  # Custom module configuration
```

8.10.2 Module reference

```
mod-rrl:
  - id: STR
    rate-limit: INT
    instant-limit: INT
    slip: INT
    time-rate-limit: INT
    time-instant-limit: INT
    table-size: INT
    whitelist: ADDR[/INT] | ADDR-ADDR | STR ...
    log-period: INT
    dry-run: BOOL
```

id

A module identifier.

rate-limit

Maximal allowed number of UDP queries per second from a single IPv6 or IPv4 address.

Rate limiting is performed for the whole address and several chosen prefixes. The limits of prefixes are constant multiples of *rate-limit*.

The specific prefixes and multipliers, which might be adjusted in the future, are for IPv6 /128: 1, /64: 2, /56: 3, /48: 4, /32: 64; for IPv4 /32: 1, /24: 32, /20: 256, /18: 768.

With each host/network, a counter of unrestricted responses is associated; if the counter would exceed its capacity, it is not incremented and the response is restricted. Counters use exponential decay for lowering their values, i.e. they are lowered by a constant fraction of their value each millisecond. The specified rate limit is reached, when the number of queries is the same every millisecond; sending many queries once a second or even a larger timespan leads to a more strict limiting.

Set to 0 to disable the rate limiting.

Default: `50`

instant-limit

Maximal allowed number of queries at a single point in time from a single IPv6 or IPv4 address. The limits for prefixes use the same multipliers as for *rate-limit*.

This limit is reached when many queries come from a new host/network, or after a longer time of inactivity.

The *instant-limit* sets the actual capacity of each counter of responses, and together with the *rate-limit* they set the fraction by which the counter is periodically lowered. The *instant-limit* may be at least *rate-limit* **/ 1000**, at which point the counters are zeroed each millisecond.

Default: `125`

slip

As attacks using DNS/UDP are usually based on a forged source address, an attacker could deny services to the victim's netblock if all responses would be completely blocked. The idea behind SLIP mechanism is to send each Nth response as truncated, thus allowing client to reconnect via TCP for at least some degree of service. It is worth noting, that some responses can't be truncated (e.g. SERVFAIL).

- Setting the value to **0** will cause that all rate-limited responses will be dropped. The outbound bandwidth and packet rate will be strictly capped by the *rate-limit* option. All legitimate requestors affected by the limit will face denial of service and will observe excessive timeouts. Therefore this setting is not recommended.
- Setting the value to **1** will cause that all rate-limited responses will be sent as truncated. The amplification factor of the attack will be reduced, but the outbound data bandwidth won't be lower than the incoming bandwidth. Also the outbound packet rate will be the same as without RRL.
- Setting the value to **2** will cause that approximately half of the rate-limited responses will be dropped, the other half will be sent as truncated. With this configuration, both outbound bandwidth and packet rate will be lower than the inbound. On the other hand, the dropped responses enlarge the time window for possible cache poisoning attack on the resolver.
- Setting the value to anything **larger than 2** will keep on decreasing the outgoing rate-limited bandwidth, packet rate, and chances to notify legitimate requestors to reconnect using TCP. These attributes are inversely proportional to the configured value. Setting the value high is not advisable.

Default: `1`

time-rate-limit

This limit works similarly to *rate-limit* but considers the time consumed (in microseconds) by the remote over non-UDP transport protocols.

Set to 0 to disable the time limiting.

Default: `5000` (microseconds)

time-instant-limit

This limit works similarly to *instant-limit* but considers the time consumed (in microseconds) by the remote over non-UDP transport protocols.

Default: `5000` (microseconds)

table-size

Maximal number of stored hosts/networks with their counters. The data structure tries to store only the most frequent sources, so it is safe to set it according to the expected maximal number of limited ones.

Use *1.4 * maximum_qps / rate-limit*, where *maximum_qps* is the number of queries which can be handled by the server per second. There is at most *maximum_qps / rate-limit* limited hosts; larger networks have higher limits and so require only a fraction of the value (handled by the *1.4* multiplier). The value will be rounded up to the nearest power of two.

The same table size is used for both counting-based and time-based limiting; the maximum number of time-limited hosts is expected to be lower, so it's not typically needed to be considered. There is at most *1 000 000 * #cpus / time-rate-limit* of them.

The memory occupied by one table structure is *8 * table-size B*.

Default: `524288`

whitelist

An ordered list of IP addresses, absolute UNIX socket paths, network subnets, or network ranges to exempt from rate limiting. Empty list means that no incoming connection will be white-listed.

Default: not set

log-period

Minimal time in milliseconds between two log messages, or zero to disable logging.

If a response is limited, the address and the prefix on which it was blocked is logged and logging is disabled for the *log-period* milliseconds. As long as limiting is needed, one source is logged each period and sources with more blocked queries have greater probability to be chosen.

The approach is used by counting-based and time-based limiting separately, so you can expect one message per *log-period* from each of them.

Default: `30000` (milliseconds)

dry-run

If enabled, the module doesn't alter any response. Only query classification is performed with possible statistics counter incrementation.

Default: `off`

8.11 `stats` — Query statistics

The module extends server statistics with incoming DNS request and corresponding response counters, such as used network protocol, total number of responded bytes, etc (see module reference for full list of supported counters). This module should be configured as the last module.

Note

Server initiated communication (outgoing NOTIFY, incoming *XFR,...) is not counted by this module.

Note

Leading 16-bit message size over TCP is not considered.

8.11.1 Example

Common statistics with default module configuration:

```
template:
  - id: default
    global-module: mod-stats
```

Per zone statistics with explicit module configuration:

```
mod-stats:
  - id: custom
    edns-presence: on
    query-type: on

template:
  - id: default
    module: mod-stats/custom
```

8.11.2 Module reference

```
mod-stats:
  - id: STR
    request-protocol: BOOL
    server-operation: BOOL
    request-bytes: BOOL
```

(continues on next page)

(continued from previous page)

```
    response-bytes: BOOL
    edns-presence: BOOL
    flag-presence: BOOL
    response-code: BOOL
    request-edns-option: BOOL
    response-edns-option: BOOL
    reply-nodata: BOOL
    query-type: BOOL
    query-size: BOOL
    reply-size: BOOL
```

id

A module identifier.

request-protocol

If enabled, all incoming requests are counted by the network protocol:

- udp4 - UDP over IPv4
- tcp4 - TCP over IPv4
- quic4 - QUIC over IPv4
- tls4 - TLS over IPv4
- udp6 - UDP over IPv6
- tcp6 - TCP over IPv6
- quic6 - QUIC over IPv6
- tls6 - TLS over IPv6
- udp4-xdp - UDP over IPv4 through XDP
- tcp4-xdp - TCP over IPv4 through XDP
- quic4-xdp - QUIC over IPv4 through XDP
- udp6-xdp - UDP over IPv6 through XDP
- tcp6-xdp - TCP over IPv6 through XDP
- quic6-xdp - QUIC over IPv6 through XDP

Default: `on`

server-operation

If enabled, all incoming requests are counted by the server operation. The server operation is based on message header OpCode and message query (meta) type:

- query - Normal query operation
- update - Dynamic update operation
- notify - NOTIFY request operation
- axfr - Full zone transfer operation
- ixfr - Incremental zone transfer operation
- invalid - Invalid server operation

Default: `on`

request-bytes

If enabled, all incoming request bytes are counted by the server operation:

- query - Normal query bytes
- update - Dynamic update bytes
- other - Other request bytes

Default: `on`

response-bytes

If enabled, outgoing response bytes are counted by the server operation:

- reply - Normal response bytes
- transfer - Zone transfer bytes
- other - Other response bytes

> **Warning**
>
> Dynamic update response bytes are not counted by this module.

Default: `on`

edns-presence

If enabled, EDNS pseudo section presence is counted by the message direction:

- request - EDNS present in request
- response - EDNS present in response

Default: `off`

flag-presence

If enabled, some message header flags are counted:

- TC - Truncated Answer in response
- DO - DNSSEC OK in request

Default: `off`

response-code

If enabled, outgoing response code is counted:

- NOERROR
- ...
- NOTZONE
- BADVERS
- ...
- BADCOOKIE
- other - All other codes

Note

In the case of multi-message zone transfer response, just one counter is incremented.

Warning

Dynamic update response code is not counted by this module.

Default: `on`

request-edns-option

If enabled, EDNS options in requests are counted by their code:

- CODE0
- ...
- EDNS-KEY-TAG (CODE14)
- other - All other codes

Default: `off`

response-edns-option

If enabled, EDNS options in responses are counted by their code. See *request-edns-option*.

Default: `off`

reply-nodata

If enabled, NODATA pseudo RCODE (**RFC 2308 Section 2.2**) is counted by the query type:

- A
- AAAA
- other - All other types

Default: `off`

query-type

If enabled, normal query type is counted:

- A (TYPE1)
- ...
- TYPE65
- SPF (TYPE99)
- ...
- TYPE110
- ANY (TYPE255)
- ...
- TYPE260
- other - All other types

> **Note**
>
> Not all assigned meta types (IXFR, AXFR,...) have their own counters, because such types are not processed as normal query.

Default: `off`

query-size

If enabled, normal query message size distribution is counted by the size range in bytes:

- 0-15
- 16-31
- ...
- 272-287
- 288-65535

Default: `off`

reply-size

If enabled, normal reply message size distribution is counted by the size range in bytes:

- 0-15
- 16-31
- ...
- 4080-4095
- 4096-65535

Default: `off`

8.12 `synthrecord` – Automatic forward/reverse records

This module is able to synthesize either forward or reverse records for a given prefix and subnet.

Records are synthesized only if the query can't be satisfied from the zone. Both IPv4 and IPv6 are supported.

8.12.1 Example

Automatic forward records

```
mod-synthrecord:
  - id: test1
    type: forward
    prefix: dynamic-
    ttl: 400
    network: 2620:0:b61::/52

zone:
  - domain: test.
    file: test.zone # Must exist
    module: mod-synthrecord/test1
```

Result:

```
$ kdig AAAA dynamic-2620-0-b61-100--1.test.
...
;; QUESTION SECTION:
;; dynamic-2620-0-b61-100--1.test. IN AAAA

;; ANSWER SECTION:
dynamic-2620-0-b61-100--1.test. 400 IN AAAA 2620:0:b61:100::1
```

You can also have CNAME aliases to the dynamic records, which are going to be further resolved:

```
$ kdig AAAA alias.test.
...
;; QUESTION SECTION:
;; alias.test. IN AAAA

;; ANSWER SECTION:
alias.test. 3600 IN CNAME dynamic-2620-0-b61-100--2.test.
dynamic-2620-0-b61-100--2.test. 400 IN AAAA 2620:0:b61:100::2
```

Automatic reverse records

```
mod-synthrecord:
  - id: test2
    type: reverse
    prefix: dynamic-
    origin: test
    ttl: 400
    network: 2620:0:b61::/52

zone:
  - domain: 1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa.
    file: 1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa.zone # Must exist
    module: mod-synthrecord/test2
```

Result:

```
$ kdig -x 2620:0:b61::1
...
;; QUESTION SECTION:
;; 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa. IN PTR

;; ANSWER SECTION:
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa. 400 IN PTR␣
↪dynamic-2620-0-b61--1.test.
```

8.12.2 Module reference

```
mod-synthrecord:
  - id: STR
    type: forward | reverse
    prefix: STR
    origin: DNAME
    ttl: INT
    network: ADDR[/INT] | ADDR-ADDR ...
    reverse-short: BOOL
```

id

A module identifier.

type

The type of generated records.

Possible values:

- `forward` – Forward records
- `reverse` – Reverse records

Required

prefix

A record owner prefix.

> **Note**
>
> The value doesn't allow dots, address parts in the synthetic names are separated with a dash.

Default: empty

origin

A zone origin (only valid for the *reverse type*).

Required

ttl

Time to live of the generated records.

Default: `3600`

network

An ordered list of IP addresses, network subnets or network ranges the query must match.

Required

reverse-short

If enabled, a shortened IPv6 address can be used for reverse record rdata synthesis.

Default: `on`

8.13 `whoami` — Whoami response

The module synthesizes an A or AAAA record containing the query source IP address, at the apex of the zone being served. It makes sure to allow Knot DNS to generate cacheable negative responses, and to allow fallback to extra records defined in the underlying zone file. The TTL of the synthesized record is copied from the TTL of the SOA record in the zone file.

Because a DNS query for type A or AAAA has nothing to do with whether the query occurs over IPv4 or IPv6, this module requires a special zone configuration to support both address families. For A queries, the underlying zone must have a set of nameservers that only have IPv4 addresses, and for AAAA queries, the underlying zone must have a set of nameservers that only have IPv6 addresses.

8.13.1 Example

To enable this module, you need to add something like the following to the Knot DNS configuration file:

```
zone:
  - domain: whoami.domain.example
    file: "/path/to/whoami.domain.example"
    module: mod-whoami

zone:
  - domain: whoami6.domain.example
    file: "/path/to/whoami6.domain.example"
    module: mod-whoami
```

The whoami.domain.example zone file example:

```
$TTL 1

@       SOA     (
                        whoami.domain.example.          ; MNAME
                        hostmaster.domain.example.      ; RNAME
                        2016051300                      ; SERIAL
                        86400                           ; REFRESH
                        86400                           ; RETRY
                        86400                           ; EXPIRE
                        1                               ; MINIMUM
                )

$TTL 86400

@       NS      ns1.whoami.domain.example.
@       NS      ns2.whoami.domain.example.
@       NS      ns3.whoami.domain.example.
@       NS      ns4.whoami.domain.example.

ns1     A       198.51.100.53
ns2     A       192.0.2.53
ns3     A       203.0.113.53
ns4     A       198.19.123.53
```

The whoami6.domain.example zone file example:

```
$TTL 1

@       SOA     (
                        whoami6.domain.example.         ; MNAME
                        hostmaster.domain.example.      ; RNAME
                        2016051300                      ; SERIAL
                        86400                           ; REFRESH
                        86400                           ; RETRY
                        86400                           ; EXPIRE
                        1                               ; MINIMUM
                )

$TTL 86400

@       NS      ns1.whoami6.domain.example.
@       NS      ns2.whoami6.domain.example.
```

(continues on next page)

(continued from previous page)

```
@       NS      ns3.whoami6.domain.example.
@       NS      ns4.whoami6.domain.example.

ns1     AAAA    2001:db8:100::53
ns2     AAAA    2001:db8:200::53
ns3     AAAA    2001:db8:300::53
ns4     AAAA    2001:db8:400::53
```

The parent domain would then delegate whoami.domain.example to ns[1-4].whoami.domain.example and whoami6.domain.example to ns[1-4].whoami6.domain.example, and include the corresponding A-only or AAAA-only glue records.

Note

This module is not configurable.

CHAPTER

NINE

UTILITIES

Knot DNS comes with a few DNS client utilities and a few utilities to control the server. This section collects manual pages for all provided binaries:

9.1 knotd – Knot DNS server daemon

9.1.1 Synopsis

knotd [*config_option*] [*options*]

9.1.2 Description

Knot DNS is a high-performance authoritative DNS server. The *knotd* program is the DNS server daemon.

Config options

-c, --config ***file***
: Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb ***directory***
: Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

Options

-m, --max-conf-size ***MiB***
: Set maximum size of the configuration database (default is @conf_mapsize@ MiB, maximum 10000 MiB).

-s, --socket ***path***
: Use a remote control UNIX socket path (default is `@run_dir@/knot.sock`).

-d, --daemonize [*directory*]
: Run the server as a daemon. New root directory may be specified (default is `/`).

-v, --verbose
: Enable debug output.

-h, --help
: Print the program help.

-V, --version
: Print the program version. The option **-VV** makes the program print the compile time configuration summary.

Signals

The *knotd* process handles the following signals:

- SIGINT – Stop the process.
- SIGHUP – Reload the server configuration and reopen configured log files.

- SIGUSR1 – Reload all configured zones (not recommended with catalog zones).

9.1.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.1.4 See Also

knot.conf(5), *knotc(8)*, *keymgr(8)*, *kjournalprint(8)*.

9.2 `knotc` – Knot DNS control utility

9.2.1 Synopsis

knotc [*config_option*] [*options*] [*action*]

9.2.2 Description

This program controls a running *knotd* process using a socket.

If an *action* is specified, it is performed and *knotc* exits, otherwise the program is executed in the interactive mode.

Config options

-c, --config ***file***
: Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb ***directory***
: Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

Options

-m, --max-conf-size ***MiB***
: Set maximum size of the configuration database (default is @conf_mapsize@ MiB, maximum 10000 MiB).

-s, --socket ***path***
: Use a control UNIX socket path (default is `@run_dir@/knot.sock`).

-t, --timeout ***seconds***
: Use a control timeout in seconds. Set to 0 for infinity (default is 60). The control socket operations are also subject to the *timeout* parameter set on the server side in server's Control configuration section.

-b, --blocking
: Zone event trigger commands wait until the event is finished. Control timeout is set to infinity if not forced by explicit timeout specification.

-e, --extended
: Show extended output (even empty items in zone status).

-f, --force
: Forced operation. Overrides some checks.

-x, --mono
: Don't generate colorized output.

-X, --color
: Force colorized output in extended output or to a pipe.

-v, --verbose
: Enable debug output.

-h, --help
: Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

Actions

status [*detail*]
Check if the server is running. Details are **version** for the running server version, **workers** for the numbers of worker threads, **configure** for the configure summary, or **cert-key** for the public key pin of the currently used certificate.

stop
Stop the server if running.

reload
Reload the server configuration and modified zone files, and reopen the log files if they are configured. All open zone transactions will be aborted!

stats [*module*[*.counter*]]
Show global statistics counter(s). To print also counters with value 0, use force option.

zone-check [*zone*...]
Test if the server can load the zone. Semantic checks are executed if enabled in the configuration. If invoked with the force option, an error is returned when semantic check warning appears. (*)

zone-status [*zone*...] [*filter*]
Show the zone status. Filters are **+role**, **+serial**, **+transaction**, **+events**, **+freeze**, and **+catalog**. Empty zone parameters are omitted, unless the **--extended** option is used. A single dash in the output represents an unset value. Automatic colorization can be overruled using the **--mono** and **--color** options.

The color code is: *green* - zone acts as a master / *red* - zone acts as a slave, *bold font (highlited)* - zone is active / *normal* - zone is empty, *underscored* - zone is an interpreted catalog member.

zone-reload [*zone*...]
Trigger a zone reload from a disk without checking its modification time. For secondary zone, the refresh event from primary server(s) is scheduled; for primary zone, the notify event to secondary server(s) is scheduled. An open zone transaction will be aborted! If invoked with the force option, also zone modules will be re-loaded, but blocking mode might not work reliably. (#)

zone-refresh [*zone*...]
Trigger a check for the zone serial on the zone's primary server. If the primary server has a newer zone, a transfer is scheduled. This command is valid for secondary zones. (#)

zone-retransfer [*zone*...]
Trigger a zone transfer from the zone's primary server. The server doesn't check the serial of the primary server's zone. This command is valid for secondary zones. (#)

zone-notify [*zone*...]
Trigger a NOTIFY message to all configured remotes. This can help in cases when previous NOTIFY had been lost or the secondary servers have been offline. (#)

zone-flush [*zone*...] [+outdir *directory*]
Trigger a zone journal flush to the configured zone file. If zonefile synchronization is disabled, the force option must be used. If an output directory is specified, the current zone is immediately dumped (in the blocking mode) to a zone file in the specified directory. See *Notes* below about the directory permissions. (#)

zone-backup [*zone*...] +backupdir *directory* [*filter*...]
Trigger a zone data and metadata backup to a specified directory. Available filters are **+zonefile**, **+journal**, **+timers**, **+kaspdb**, **+keysonly**, **+catalog**, **+quic**, and their negative counterparts **+nozonefile**, **+nojournal**, **+notimers**, **+nokaspdb**, **+nokeysonly**, **+nocatalog**, and **+noquic**. With these filters set, zone contents, zone's journal, zone-related timers, zone-related data in the KASP database together with keys (or keys without the KASP database), zone's catalog, and the server QUIC key and certificate, respectively, are backed up, or omitted from the backup. By default, filters **+zonefile**, **+timers**, **+kaspdb**, **+catalog**, **+quic**, **+nojournal**,

and **+nokeysonly** are set for backup. The same defaults are set for restore, with the only difference being **+noquic**. Setting a filter for an item doesn't change the default settings for other items. The only exception is **+keysonly**, which disables all other filters by default, but they can still be turned on explicitly. If zone flushing is disabled, the original zone file is backed up instead of writing out zone contents to a file. When backing-up a catalog zone, it is recommended to prevent ongoing changes to it by use of **zone-freeze**. The force option allows an already existing backupdir to be overwritten. See *Notes* below about the directory permissions. (#)

zone-restore [*zone*...] +backupdir *directory* [*filter*...]
Trigger a zone data and metadata restore from a specified backup directory. Optional filters are equivalent to the same filters of **zone-backup**. Restore from backups created by Knot DNS releases prior to 3.1 is possible with the force option. See *Notes* below about the directory permissions. (#)

zone-sign [*zone*...]
Trigger a DNSSEC re-sign of the zone. Existing signatures will be dropped. This command is valid for zones with DNSSEC signing enabled. (#)

zone-validate [*zone*...]
Trigger a DNSSEC validation of the zone. If the validation fails and the zone is secondary, the zone expires immediately! (#)

zone-keys-load [*zone*...]
Trigger a load of DNSSEC keys and other signing material from KASP database (which might have been altered manually). If suitable, re-sign the zone afterwards (keeping valid signatures intact). (#)

zone-key-rollover *zone key_type*
Trigger immediate key rollover. Publish new key and start a key rollover, even when the key has a lifetime to go. Key type can be **ksk** (also for CSK) or **zsk**. This command is valid for zones with DNSSEC signing and automatic key management enabled. Note that complete key rollover consists of several steps and the blocking mode relates to the initial one only! (#)

zone-ksk-submitted *zone*...
Use when the zone's KSK rollover is in submission phase. By calling this command the user confirms manually that the parent zone contains DS record for the new KSK in submission phase and the old KSK can be retired. (#)

zone-freeze [*zone*...]
Trigger a zone freeze. All running events will be finished and all new and pending (planned) zone-changing events (load, refresh, update, flush, and DNSSEC signing) will be held up until the zone is thawed. Up to 8 (this limit is hardcoded) DDNS updates per zone will be queued, subsequent updates will be refused. (#)

zone-thaw [*zone*...]
Trigger dismissal of zone freeze. (#)

zone-xfr-freeze [*zone*...]
Temporarily disable outgoing AXFR/IXFR for the zone(s). (#)

zone-xfr-thaw [*zone*...]
Dismiss outgoing XFR freeze. (#)

zone-read *zone* [*owner* [*type*]]
Get zone data that are currently being presented.

zone-begin *zone*... [+benevolent]
Begin a zone transaction. If **+benevolent** is used, the transaction will be committed even if it contains removals of non-existent records or additions of existing records.

zone-commit *zone*...
Commit the zone transaction. All changes are applied to the zone.

zone-abort *zone*...
Abort the zone transaction. All changes are discarded.

zone-diff *zone*
Get zone changes within the transaction.

zone-get ***zone*** **[*owner* [*type*]]**
Get zone data within the transaction.

zone-set ***zone owner*** **[*ttl*]** ***type rdata***
Add zone record within the transaction. The first record in a rrset requires a ttl value specified.

zone-unset ***zone owner*** **[*type* [*rdata*]]**
Remove zone data within the transaction.

zone-purge ***zone*...** **[+orphan] [*filter*...]**
Purge zone data, zone file, journal, timers, and/or KASP data of specified zones. Available filters are **+expire**, **+zonefile**, **+journal**, **+timers**, **+kaspdb**, and **+catalog**. If no filter is specified, all filters are enabled. If the zone is no longer configured, add **+orphan** parameter (zone file cannot be purged in this case). When purging orphans, always check the server log for possible errors. For proper operation, it's necessary to prevent ongoing changes to the zone and triggering of zone related events during purge; use of **zone-freeze** is advisable. This command always requires the force option. (#)

zone-stats ***zone*** **[*module*[.*counter*]]**
Show zone statistics counter(s). To print also counters with value 0, use force option.

conf-init
Initialize the configuration database. If the database doesn't exist yet, execute this command as an intended user to ensure the server is permitted to access the database (e.g. *sudo -u knot knotc conf-init*). (*)

conf-check
Check the server configuration. (*)

conf-import ***filename*** **[+nopurge]**
Import a configuration file into the configuration database. If the database doesn't exist yet, execute this command as an intended user to ensure the server is permitted to access the database (e.g. *sudo -u knot knotc conf-import ...*). An optional filter **+nopurge** prevents possibly existing configuration database from purging before the import itself. Also ensure the server is not using the configuration database at the same time! (*)

conf-export [*filename*] [+schema]
Export the configuration database (or JSON schema) into a file or stdout. (*)

conf-list [*item*]
List the configuration database sections or section items.

conf-read [*item*]
Read the item from the active configuration database.

conf-begin
Begin a writing configuration database transaction. Only one transaction can be opened at a time.

conf-commit
Commit the configuration database transaction.

conf-abort
Rollback the configuration database transaction.

conf-diff [*item*]
Get the item difference in the transaction.

conf-get [*item*]
Get the item data from the transaction.

conf-set ***item*** **[*data*...]**
Set the item data in the transaction.

conf-unset [*item*] [*data*...]
Unset the item data in the transaction.

Notes

Empty or **--** *zone* parameter means all zones or all zones with a transaction.

Use @ *owner* to denote the zone name.

Type *item* parameter in the form of *section*[[*id*]][.*name*].

(*) indicates a local operation which requires a configuration.

(#) indicates an optionally blocking operation.

The **-b** and **-f** options can be placed right after the command name.

Responses returned by *knotc* commands depend on the mode:

- In the blocking mode, *knotc* reports if an error occurred during processing of the command by the server. If an error is reported, a more detailed information about the failure can usually be found in the server log.
- In the non-blocking (default) mode, *knotc* doesn't report processing errors. The *OK* response to triggering commands means that the command has been successfully sent to the server. To verify if the operation succeeded, it's necessary to check the server log.

Actions **zone-flush**, **zone-backup**, and **zone-restore** are carried out by the *knotd* process. The directory specified must be accessible to the user account that *knotd* runs under and if the directory already exists, its permissions must be appropriate for that user account.

Interactive mode

The utility provides interactive mode with basic line editing functionality, command completion, and command history.

Interactive mode behavior can be customized in *~/.editrc*. Refer to `editrc(5)` for details.

Command history is saved in *~/.knotc_history*.

9.2.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.2.4 Examples

Reload the whole server configuration

```
$ knotc reload
```

Flush the example.com and example.org zones

```
$ knotc zone-flush example.com example.org
```

Get the current server configuration

```
$ knotc conf-read server
```

Get the list of the current zones

```
$ knotc conf-read zone.domain
```

Get the primary servers for the example.com zone

```
$ knotc conf-read 'zone[example.com].master'
```

Add example.org zone with a zonefile location

```
$ knotc conf-begin
$ knotc conf-set 'zone[example.org]'
$ knotc conf-set 'zone[example.org].file' '/var/zones/example.org.zone'
$ knotc conf-commit
```

Get the SOA record for each configured zone

```
$ knotc zone-read -- @ SOA
```

9.2.5 See Also

knotd(8), *knot.conf(5)*, *editrc(5)*.

9.3 keymgr – Key management utility

9.3.1 Synopsis

keymgr [*config_option*] [*options*] *zone_name command*

keymgr [*config_option*] [*options*] *keystore_id command*

keymgr [*config_option*] [-j] **-l**

keymgr -t *parameter*...

9.3.2 Description

The **keymgr** utility serves for manual key management in Knot DNS server.

Functions for DNSSEC keys and KASP (Key And Signature Policy) management are provided.

The DNSSEC and KASP configuration is stored in a so called KASP database. The database is backed by LMDB.

Parameters

zone_name
Name of the zone the command is executed for.

Config options

-c, --config *file*
Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb *directory*
Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

-D, --dir *path*
Use specified KASP database path and default configuration.

Options

-t, --tsig *tsig_name* [*tsig_algorithm* [*tsig_bits*]]
Generates a TSIG key for the given name. Optionally the key algorithm can be specified by its *name* (default: hmac-sha256) and a bit length of the key (default: optimal length given by algorithm). The generated TSIG key is only displayed on *stdout*: the command does not create a file, nor include the key in a keystore.

-e, --extended
Extended output (listing of keys with full description).

-j, --json
: Print the zones or keys (not TSIG keys) in JSON format.

-l, --list
: Print the list of zones that have at least one key stored in the configured KASP database.

-x, --mono
: Don't generate colorized output.

-X, --color
: Force colorized output in the normal mode.

-h, --help
: Print the program help.

-V, --version
: Print the program version. The option **-VV** makes the program print the compile time configuration summary.

> **Note**
>
> Keymgr runs with the same user privileges as configured for *knotd*. For example, if keymgr is run as `root`, but the configured *user* is `knot`, it won't be able to read files (PEM files, KASP database, ...) readable only by `root`.

Commands

list [*timestamp_format*]
: Prints the list of key IDs and parameters of keys belonging to the zone.

generate [*arguments...*]
: Generates new DNSSEC key and stores it in KASP database. Prints the key ID. This action takes some number of arguments (see below). Values for unspecified arguments are taken from corresponding policy (if *-c* or *-C* options used) or from Knot policy defaults.

import-bind *BIND_key_file*
: Imports a BIND-style key into KASP database (converting it to PEM format). Takes one argument: path to BIND key file (private or public, but both MUST exist).

import-pub *BIND_pubkey_file*
: Imports a public key into KASP database. This key won't be rolled over nor used for signing. Takes one argument: path to BIND public key file.

import-pem *PEM_file* [*arguments...*]
: Imports a DNSSEC key from PEM file. The key parameters (same as for the generate action) need to be specified (mainly algorithm, timers...) because they are not contained in the PEM format.

import-pkcs11 *key_id* [*arguments...*]
: Imports a DNSSEC key from PKCS #11 storage. The key parameters (same as for the generate action) need to be specified (mainly algorithm, timers...) because they are not available. In fact, no key data is imported, only KASP database metadata is created.

nsec3-salt [*new_salt*]
: Prints the current NSEC3 salt used for signing. If *new_salt* is specified, the salt is overwritten. The salt is printed and expected in hexadecimal, or dash if empty.

local-serial [*new_serial*]
: Print SOA serial stored in KASP database when using on-secondary DNSSEC signing. If *new_serial* is specified, the serial is overwritten. After updating the serial, expire the zone (**zone-purge +expire +zonefile +journal**) if the server is running, or remove corresponding zone file and journal contents if the server is stopped.

master-serial [*new_serial*]
Print SOA serial of the remote master stored in KASP database when using on-secondary DNSSEC signing. If *new_serial* is specified, the serial is overwritten (not recommended).

set *key_spec* [*arguments*...]
Changes a timing argument (or ksk/zsk) of an existing key to a new value. *Key_spec* is either the key tag or a prefix of the key ID, with an optional *[id=|keytag=]* prefix; *arguments* are like for **generate**, but just the related ones.

ds [*key_spec*]
Generate DS record (all digest algorithms together) for specified key. *Key_spec* is like for **set**, if unspecified, all KSKs are used.

dnskey [*key_spec*]
Generate DNSKEY record for specified key. *Key_spec* is like for **ds**, if unspecified, all KSKs are used.

delete *key_spec*
Remove the specified key from zone. If the key was not shared, it is also deleted from keystore.

share *key_ID zone_from*
Import a key (specified by full key ID) from another zone as shared. After this, the key is owned by both zones equally.

Keystore commands

keystore-test
Conduct some tests on the specified keystore. For each algorithm, key generation, import, removal, and use (signing and verification) are tested. Use a configured *keystore_id* or **-** for the default.

keystore-bench [*num_threads*]
Conduct a signing benchmark on the specified keystore. Random blocks of data are signed by the selected number of threads (default is 1) in a loop, and the average number of signing operations per second for each algorithm is returned. Use a configured *keystore_id* or **-** for the default.

Commands related to Offline KSK feature

pregenerate [*timestamp-from*] *timestamp-to*
Pre-generate ZSKs for use with offline KSK, for the specified period starting from now or specified time. This function also applies to non-offline KSK keys.

show-offline [*timestamp-from*] [*timestamp-to*]
Print pre-generated offline key-related records for specified time interval. If *timestamp_to* is omitted, it will be to infinity. If *timestamp-from* is omitted, it will start from the beginning.

del-offline *timestamp-from timestamp-to*
Delete pre-generated offline key-related records in specified time interval.

del-all-old
Delete old keys that are in state 'removed'. This function also applies to non-offline KSK keys.

generate-ksr [*timestamp-from*] *timestamp-to*
Print to stdout KeySigningRequest based on pre-generated ZSKs for specified time period. If *timestamp-from* is omitted, timestamp of the last offline records set is used or now if no records available.

sign-ksr *ksr_file*
Read KeySigningRequest from a text file, sign it using local keyset and print SignedKeyResponse to stdout.

validate-skr *skr_file*
Read SignedKeyResponse from a text file and validate the RRSIGs in it if not corrupt.

import-skr *skr_file*
Read SignedKeyResponse from a text file and import the signatures for later use in zone. If some signatures have already been imported, they will be deleted for the period from beginning of the SKR to infinity.

Generate arguments

Arguments are separated by space, each of them is in format 'name=value'. A *(bool)* argument accepts values **yes** or **no**. If the value is omitted, it's interpreted as **yes**.

algorithm
Either an algorithm number (e.g. 14) or *algorithm name* (e.g. ecdsap256sha256).

size
Key length in bits.

ksk
(bool) The key will be used for signing DNSKEY rrset. The generated key will also have the Secure Entry Point flag set to 1.

zsk
(bool) The key will be used for signing zone (except DNSKEY rrset). This flag can be set concurrently with the **ksk** flag (for a CSK key).

sep
(bool) Overrides the standard setting of the Secure Entry Point flag.

for-later
(bool) The key will be determined for later use. It means once a new key ought to be generated, this one will be used instead if the key type and algorithm match.

The following arguments are timestamps of key lifetime (see *DNSSEC key states*):

pre_active
Key started to be used for signing, not published (only for algorithm rollover).

publish
Key published.

ready
Key is waiting for submission (only for KSK).

active
Key used for signing.

retire_active
Key still used for signing, but another key is active (only for KSK or algorithm rollover).

retire
Key still published (only if ZSK), but no longer used for signing.

post_active
Key no longer published, but still used for signing (only for algorithm rollover).

revoke
Key revoked according to **RFC 5011** trust anchor roll-over.

remove
Key deleted.

Timestamps

0
Zero timestamp means infinite future.

UNIX_time
Positive number of seconds since 1970 UTC.

YYYYMMDDHHMMSS
Date and time in this format without any punctuation.

relative_timestamp
A sign character (**+**, **-**), a number, and an optional time unit (**y**, **mo**, **d**, **h**, **mi**, **s**). The default unit is one second. E.g. +1mi, -2mo.

Output timestamp formats

(none)
The timestamps are printed as UNIX timestamp.

human
The timestamps are printed relatively to now using time units (e.g. -2y5mo, +1h13s).

iso
The timestamps are printed in the ISO8601 format (e.g. 2016-12-31T23:59:00).

9.3.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.3.4 Examples

1. Generate new TSIG key:

```
$ keymgr -t my_name hmac-sha384
```

2. Generate new DNSSEC key:

```
$ keymgr example.com. generate algorithm=ECDSAP256SHA256 size=256 \
  ksk=true created=1488034625 publish=20170223205611 retire=+10mo remove=+1y
```

3. Import a DNSSEC key from BIND:

```
$ keymgr example.com. import-bind ~/bind/Kharbinge4d5.+007+63089.key
```

4. Import a CSK DNSSEC key from a PEM file:

```
$ keymgr example.com. import-pem 085d3890e8c22d854586678d9263933f2d02d795.pem␣
↪ksk=yes zsk=yes
```

5. Configure key timing:

```
$ keymgr example.com. set 4208 active=+2mi retire=+4mi remove=+5mi
```

6. Share a KSK from another zone:

```
$ keymgr example.com. share e687cf927029e9db7184d2ece6d663f5d1e5b0e9 another-
↪zone.com.
```

9.3.5 See Also

RFC 6781 - DNSSEC Operational Practices. RFC 7583 - DNSSEC Key Rollover Timing Considerations.

knot.conf(5), *knotc(8)*, *knotd(8)*.

9.4 kjournalprint – Knot DNS journal print utility

9.4.1 Synopsis

kjournalprint [*config_option*] [*options*] *zone_name*

kjournalprint [*config_option*] **-z**

9.4.2 Description

The program prints zone history stored in a journal database. As default, changes are colored for terminal.

Parameters

zone_name
A name of the zone to print the history for.

Config options

-c, --config ***file***
Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb ***directory***
Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

-D, --dir ***path***
Use specified journal database path and default configuration.

Options

-z, --zone-list
Instead of reading the journal, display the list of zones in the DB.

-l, --limit ***limit***
Limits the number of displayed changes.

-s, --serial ***soa***
Start at a specific SOA serial.

-M, --merge
Print the changesets merged into one changeset. If zone-in-journal is present, the stored contents with all the changesets applied will be printed.

-H, --check
Enable additional journal semantic checks during printing.

-d, --debug
Debug mode brief output.

-x, --mono
Don't generate colorized output.

-X, --color
Force colorized output.

-h, --help
Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

9.4.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.4.4 Examples

Last (most recent) 5 changes without colors:

```
$ kjournalprint -xl 5 /var/lib/knot/journal example.com.
```

9.4.5 See Also

knotd(8), *knot.conf(5)*.

9.5 `kcatalogprint` – Knot DNS catalog print utility

9.5.1 Synopsis

kcatalogprint [*config_option*] [*options*]

9.5.2 Description

The program prints zone catalog stored in a catalog database.

Config options

-c, --config ***file***
Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb ***directory***
Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

-D, --dir ***path***
Use specified catalog database path and default configuration.

Options

-a, --catalog
Filter the output by catalog zone name.

-m, --member
Filter the output by member zone name.

-h, --help
Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

9.5.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.5.4 See Also

knotd(8), *knot.conf(5)*.

9.6 `kzonecheck` – Knot DNS zone file checking tool

9.6.1 Synopsis

kzonecheck [*options*] *filename*

9.6.2 Description

The utility checks zone file syntax and runs semantic checks on the zone content. The executed checks are the same as the checks run by the Knot DNS server.

Please, refer to the `semantic-checks` configuration option in *knot.conf(5)* for the full list of available semantic checks.

Parameters

filename
Path to the zone file to be checked. For reading from **stdin** use **/dev/stdin** or just **-**.

Options

-o, --origin ***origin***
Zone origin. If not specified, the SOA record owner in the zone file is used and the zone file name (without possible **.zone** suffix) is considered as the initial zone origin in case the owner isn't FQDN.

-d, --dnssec on|off
Also check DNSSEC-related records. The default is to decide based on the existence of a RRSIG for SOA.

-z, --zonemd
Also check the zone hash against a ZONEMD record, which is required to exist.

-t, --time ***time***
Current time specification. Use UNIX timestamp, YYYYMMDDHHmmSS format, or [+/-]*time*[unit] format, where unit can be **Y**, **M**, **D**, **h**, **m**, or **s**. Default is current UNIX timestamp.

-j, --jobs ***jobs***
The number of threads used for DNSSEC validation. Default is all CPU threads available.

-p, --print
Print the zone on stdout.

-v, --verbose
Enable debug output.

-h, --help
Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

9.6.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.6.4 See Also

knotd(8), *knot.conf(5)*.

9.7 `kzonesign` – DNSSEC signing utility

9.7.1 Synopsis

kzonesign [*config_option*] [*options*] *zone_name*

9.7.2 Description

This utility reads the zone's zone file, signs the zone according to given configuration, and writes the signed zone file back. An alternative mode is DNSSEC validation of the given zone. The signing or validation can run in parallel if enabled in the configuration (see policy.signing-threads and zone.adjust-threads).

Parameters

zone_name
A name of the zone to be signed.

Config options

-c, --config ***file***
: Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, --confdb ***directory***
: Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

Options

-o, --outdir ***dir_name***
: Write the output zone file to the specified directory instead of the configured one.

-r, --rollover
: Allow key roll-overs and NSEC3 re-salt. In order to finish possible KSK submission, set the KSK's **active** timestamp to now (**+0**) using *keymgr*.

-v, --verify
: Instead of (re-)signing the zone, just verify that the zone is correctly signed.

-t, --time ***timestamp***
: Sign/verify the zone (and roll the keys if necessary) as if it was at the time specified by timestamp.

-h, --help
: Print the program help.

-V, --version
: Print the program version. The option **-VV** makes the program print the compile time configuration summary.

9.7.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.7.4 See Also

`knot.conf(5)`, *`keymgr(8)`*.

9.8 `kdig` – Advanced DNS lookup utility

9.8.1 Synopsis

`kdig` [*common-settings*] [*query* [*settings*]]...

`kdig` **-h**

9.8.2 Description

This utility sends one or more DNS queries to a nameserver. Each query can have individual *settings*, or it can be specified globally via *common-settings*, which must precede *query* specification.

Parameters

query
: *name* | **-q** *name* | **-x** *address* | **-G** *tapfile*

common-settings*, *settings
: [*query_class*] [*query_type*] [*@server*]... [*options*]

name
: Is a domain name that is to be looked up.

server
: Is a domain name or an IPv4 or IPv6 address of the nameserver to send a query to. An additional port can be specified using address:port ([address]:port for IPv6 address), address@port, or address#port notation. A value which begins with '/' character is considered an absolute UNIX socket path. If no server is specified, the servers from `/etc/resolv.conf` are used.

If no arguments are provided, **`kdig`** sends NS query for the root zone.

Query classes

A *query_class* can be either a DNS class name (IN, CH) or generic class specification **CLASS***XXXXX* where *XXXXX* is a corresponding decimal class number. The default query class is IN.

Query types

A *query_type* can be either a DNS resource record type (A, AAAA, NS, SOA, DNSKEY, ANY, etc.) or one of the following:

TYPE*XXXXX*
: Generic query type specification where *XXXXX* is a corresponding decimal type number.

AXFR
: Full zone transfer request.

IXFR=*serial*
: Incremental zone transfer request for specified SOA serial number (i.e. all zone updates since the specified zone version are to be returned).

NOTIFY=*serial*
: Notify message with a SOA serial hint specified.

NOTIFY
: Notify message with a SOA serial hint unspecified.

The default query type is A.

Options

-4
: Use the IPv4 protocol only.

-6
: Use the IPv6 protocol only.

-b *address*
: Set the source IP address of the query to *address*. The address must be a valid address for local interface or :: or 0.0.0.0. An optional port can be specified in the same format as the *server* value.

-c *class*
: An explicit *query_class* specification. See possible values above.

-d
: Enable debug messages.

-dd
: Enable more verbose debug messages (print Subject Alternative Name).

-h, --help
: Print the program help.

-k *keyfile*
: Use the TSIG key stored in a file *keyfile* to authenticate the request. The file must contain the key in the same format as accepted by the **-y** option.

-p *port*
: Set the nameserver port number or service name to send a query to. The default port is 53.

-q *name*
Set the query name. An explicit variant of *name* specification. If no *name* is provided, empty question section is set.

-t *type*
An explicit *query_type* specification. See possible values above.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

-x *address*
Send a reverse (PTR) query for IPv4 or IPv6 *address*. The correct name, class and type is set automatically.

-y [*alg*:]*name*:*key*
Use the TSIG key named *name* to authenticate the request. The *alg* part specifies the algorithm (the default is hmac-sha256) and *key* specifies the shared secret encoded in Base64.

-E *tapfile*
Export a dnstap trace of the query and response messages received to the file *tapfile*.

-G *tapfile*
Generate message output from a previously saved dnstap file *tapfile*.

+[no]multiline
Wrap long records to more lines and improve human readability.

+[no]short
Show record data only.

+[no]generic
Use the generic representation format when printing resource record types and data.

+[no]crypto
Display the DNSSEC keys and signatures values in base64, instead of omitting them. (*)

+[no]aaflag
Set the AA flag.

+[no]tcflag
Set the TC flag.

+[no]rdflag
Set the RD flag. (*)

+[no]recurse
Same as **+[no]rdflag**

+[no]raflag
Set the RA flag.

+[no]zflag
Set the zero flag bit.

+[no]adflag
Set the AD flag. (*)

+[no]cdflag
Set the CD flag.

+[no]doflag
Set the DO flag.

+[no]dnssec
Same as **+[no]doflag**

+[no]all
Show all packet sections.

+[no]qr
Show the query packet.

+[no]header
Show the packet header. (*)

+[no]comments
Show commented section names. (*)

+[no]opt
Show the EDNS pseudosection. (*)

+[no]opttext
Try to show unknown EDNS options as text.

+[no]optpresent
Show EDNS in presentation format according to the specification in version draft-peltan-edns-presentation-format-01.

+[no]question
Show the question section. (*)

+[no]answer
Show the answer section. (*)

+[no]authority
Show the authority section. (*)

+[no]additional
Show the additional section. (*)

+[no]tsig
Show the TSIG pseudosection. (*)

+[no]stats
Show trailing packet statistics. (*)

+[no]class
Show the DNS class. (*)

+[no]ttl
Show the TTL value. (*)

+[no]tcp
Use the TCP protocol (default is UDP for standard query and TCP for AXFR/IXFR).

+[no]fastopen
Use TCP Fast Open.

+[no]ignore
Don't use TCP automatically if a truncated reply is received.

+[no]keepopen
Keep TCP connection open for the following query if it has the same connection configuration. This applies to +tcp, +tls, +https, and +quic operations. The connection is considered in the context of a single kdig call only.

+[no]tls
Use TLS with the Opportunistic privacy profile (**RFC 7858 Section 4.1**).

+[no]tls-ca[=*FILE*]
Use TLS with a certificate validation. Certification authority certificates are loaded from the specified PEM file (default is system certificate storage if no argument is provided). Can be specified multiple times. If the +tls-hostname option is not provided, the name of the target server (if specified) is used for strict authentication.

+[no]tls-pin=*BASE64*
Use TLS with the Out-of-Band key-pinned privacy profile (**RFC 7858 Section 4.2**). The PIN must be a Base64 encoded SHA-256 hash of the X.509 SubjectPublicKeyInfo. Can be specified multiple times.

+[no]tls-hostname=*STR*
Use TLS with a remote server hostname check.

+[no]tls-sni=*STR*
Use TLS with a Server Name Indication.

+[no]tls-keyfile=*FILE*
Use TLS with a client keyfile.

+[no]tls-certfile=*FILE*
Use TLS with a client certfile.

+[no]tls-ocsp-stapling[=*H*]
Use TLS with a valid stapled OCSP response for the server certificate (%u or specify hours). OCSP responses older than the specified period are considered invalid.

+[no]https[=*URL*]
Use HTTPS (DNS-over-HTTPS) in wire format (**RFC 1035 Section 4.2.1**). It is also possible to specify URL=[authority][/path] where request will be sent to. Any leading scheme and authority indicator (i.e. //) are ignored. Authority might also be specified as the *server* (using the parameter @). If *path* is specified and *authority* is missing, then the *server* is used as authority together with the specified *path*. Library *libnghttp2* is required.

+[no]https-get
Use HTTPS with HTTP/GET method instead of the default HTTP/POST method. Library *libnghttp2* is required.

+[no]quic
Use QUIC (DNS-over-QUIC).

+[no]nsid
Request the nameserver identifier (NSID).

+[no]zoneversion
Request the EDNS zone version.

+[no]bufsize=*B*
Set EDNS buffer size in bytes (default is 1232 bytes).

+[no]padding[=*B*]
Use EDNS(0) padding option to pad queries, optionally to a specific size. The default is to pad queries with a sensible amount when using +tls, and not to pad at all when queries are sent without TLS. With no argument (i.e., just +padding) pad every query with a sensible amount regardless of the use of TLS. With +nopadding, never pad.

+[no]alignment[=*B*]
Align the query to B-byte-block message using the EDNS(0) padding option (default is no or 128 if no argument is specified).

+[no]subnet=*SUBN*
Set EDNS(0) client subnet SUBN=addr/prefix.

+[no]edns[=*N*]
Use EDNS version (default is 0).

+[no]msgdelay=*T*
Wait the specified number of milliseconds before receiving each AXFR/IXFR message.

+[no]timeout=*T*
Set the wait-for-reply interval in seconds (default is 5 seconds). This timeout applies to each query attempt. Zero value or *notimeout* is interpreted as infinity.

+[no]retry=*N*
: Set the number (>=0) of UDP retries (default is 2). This doesn't apply to AXFR/IXFR.

+[no]expire
: Sets the EXPIRE EDNS option.

+[no]cookie[=*HEX*]
: Attach EDNS(0) cookie to the query.

+[no]badcookie
: Repeat a query with the correct cookie.

+[no]ednsopt[=*CODE*[:*HEX*]]
: Send custom EDNS option. The *CODE* is EDNS option code in decimal, *HEX* is an optional hex encoded string to use as EDNS option value. This argument can be used multiple times. +noednsopt clears all EDNS options specified by +ednsopt.

+[no]proxy=*SRC_ADDR*[#*SRC_PORT*]-*DST_ADDR*[#*DST_PORT*]
: Add PROXYv2 header with the specified source and destination addresses to the query. The default source port is 0 and destination port 53.

+[no]json
: Use JSON for output encoding (RFC 8427).

+noidn
: Disable the IDN transformation to ASCII and vice versa. IDN support depends on libidn2 availability during project building! If used in *common-settings*, all IDN transformations are disabled. If used in the individual query *settings*, transformation from ASCII is disabled on output for the particular query. Note that IDN transformation does not preserve domain name letter case.

9.8.3 Notes

By default, only options marked with (*) are enabled.

Options **-k** and **-y** can not be used simultaneously.

Dnssec-keygen keyfile format is not supported. Use *`keymgr(8)`* instead.

9.8.4 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.8.5 Examples

1. Get A records for example.com:

```
$ kdig example.com A
```

2. Perform AXFR for zone example.com from the server 192.0.2.1:

```
$ kdig example.com -t AXFR @192.0.2.1
```

3. Get A records for example.com from 192.0.2.1 and reverse lookup for address 2001:DB8::1 from 192.0.2.2. Both using the TCP protocol:

```
$ kdig +tcp example.com -t A @192.0.2.1 -x 2001:DB8::1 @192.0.2.2
```

4. Get SOA record for example.com, use TLS, use system certificates, check for specified hostname, check for certificate pin, and print additional debug info:

```
$ kdig -d @185.49.141.38 +tls-ca +tls-host=getdnsapi.net \
  +tls-pin=foxZRnIh9gZpWnl+zEiKa0EJ2rdCGroMWm02gaxSc9S= soa example.com
```

5. DNS over HTTPS examples (various DoH implementations):

```
$ kdig @1.1.1.1 +https example.com.
$ kdig @193.17.47.1 +https=/doh example.com.
$ kdig @8.8.4.4 +https +https-get example.com.
$ kdig @8.8.8.8 +https +tls-hostname=dns.google +fastopen example.com.
```

6. More queries share one DoT connection:

```
$ kdig @1.1.1.1 +tls +keepopen abc.example.com A mail.example.com AAAA
```

7. Query the name server identifier and version according to **RFC 4892**:

```
$ kdig @a.iana-servers.net. CH TXT id.server version.server
```

9.8.6 Files

`/etc/resolv.conf`

9.8.7 See Also

khost(1), *knsupdate(1)*, *keymgr(8)*.

9.9 `khost` – Simple DNS lookup utility

9.9.1 Synopsis

khost [*options*] *name* [*server*]

9.9.2 Description

This utility sends a DNS query for the *name* to the *server* and prints a reply in more user-readable form. For more advanced DNS queries use *kdig* instead.

Parameters

name

Is a domain name that is to be looked up. If the *name* is IPv4 or IPv6 address the PTR query type is used.

server

Is a name or an address of the nameserver to send a query to. The address can be specified using [address]:port notation. If no server is specified, the servers from `/etc/resolv.conf` are used.

If no arguments are provided, **khost** prints a short help.

Options

-4

Use the IPv4 protocol only.

-6

Use the IPv6 protocol only.

-a

Send ANY query with verbose mode.

-d

Enable debug messages.

-h, --help

Print the program help.

-r

Disable recursion.

-T
: Use the TCP protocol.

-v
: Enable verbose output.

-V, --version
: Print the program version. The option **-VV** makes the program print the compile time configuration summary.

-w
: Wait forever for the reply.

-c ***class***
: Set the query class (e.g. CH, CLASS4). The default class is IN.

-t ***type***
: Set the query type (e.g. NS, IXFR=12345, TYPE65535). The default is to send 3 queries (A, AAAA and MX).

-R ***retries***
: The number (>=0) of UDP retries to query a nameserver. The default is 1.

-W ***wait***
: The time to wait for a reply in seconds. This timeout applies to each query try. The default is 2 seconds.

9.9.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.9.4 Examples

1. Get the A, AAAA and MX records for example.com:

   ```
   $ khost example.com
   ```

2. Get the reverse record for address 192.0.2.1:

   ```
   $ khost 192.0.2.1
   ```

3. Perform a verbose zone transfer for zone example.com:

   ```
   $ khost -t AXFR -v example.com
   ```

9.9.5 Files

`/etc/resolv.conf`

9.9.6 See Also

`kdig(1)`, *`knsupdate(1)`*.

9.10 `knsec3hash` – NSEC hash computation utility

9.10.1 Synopsis

`knsec3hash` *salt algorithm iterations name*

`knsec3hash` *algorithm flags iterations salt name*

`knsec3hash` [-*h*] [-*V*]

9.10.2 Description

This utility generates a NSEC3 hash for a given domain name and parameters of NSEC3 hash.

Parameters

salt
Specifies a binary salt encoded as a hexadecimal string.

algorithm
Specifies a hashing algorithm by number. Currently, the only supported algorithm is SHA-1 (number 1).

iterations
Specifies the number of additional iterations of the hashing algorithm.

name
Specifies the domain name to be hashed.

flags
Specifies NSEC3 flags as an unsigned integer.

Options

-h, --help
Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

9.10.3 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.10.4 Examples

```
$ knsec3hash 1 0 10 c01dcafe knot-dns.cz
7PTVGE7QV67EM61ROS9238P5RAKR2DM7 (salt=c01dcafe, hash=1, iterations=10)
```

```
$ knsec3hash - 1 0 net
A1RT98BS5QGC9NFI51S9HCI47ULJG6JH (salt=-, hash=1, iterations=0)
```

9.10.5 See Also

RFC 5155 – DNS Security (DNSSEC) Hashed Authenticated Denial of Existence.

knotc(8), *knotd(8)*.

9.11 knsupdate – Dynamic DNS update utility

9.11.1 Synopsis

knsupdate [**-v**] [*options*] [*filename*]

knsupdate [**-q**] [*quic_options*] [*options*] [*filename*]

9.11.2 Description

This utility sends Dynamic DNS update messages to a DNS server. Update content is read from a file (if the parameter *filename* is given) or from the standard input.

The format of updates is textual and is made up of commands. Every command is placed on the separate line of the input. Lines starting with a semicolon are comments and are not processed.

Parameters

filename
Path to the file with knsupdate commands.

Options

-T, --tcp
Use a TCP connection. (**-v** can be used for compatibility with nsupdate).

-S, --tls
Use a TLS connection.

-Q, --quic
Use a QUIC connection.

-p, --port ***number***
Set the port to use for connections to the server (if not explicitly specified in the update). The default is 53 for UDP/TCP or 853 for QUIC.

-r, --retry ***count***
The number of retries for UDP requests. The default is 3.

-t, --timeout ***seconds***
The total timeout (for all UDP update tries) of the update request in seconds. The default is 12. If set to zero, the timeout is infinite.

-y, --tsig [*alg*:]*name*:*key*
Use the TSIG key with a name *name* to authenticate the request. The *alg* part specifies the algorithm (the default is hmac-sha256) and *key* specifies the shared secret encoded in Base64.

-k, --tsigfile ***path***
Use the TSIG key stored in a file *keyfile* to authenticate the request. The file should contain the key in the same format, which is accepted by the **-y** option.

-d, --debug
Enable debug messages.

-h, --help
Print the program help.

-V, --version
Print the program version. The option **-VV** makes the program print the compile time configuration summary.

QUIC/TLS options

-H, --hostname ***string***
Enable remote server hostname validation.

-P, --pin ***base64***
Use Out-of-Band key-pinned privacy profile (RFC 7858#section-4.2). The PIN must be a Base64 encoded SHA-256 hash of the X.509 SubjectPublicKeyInfo. Can be specified multiple times.

-A, --ca [*path*]
Enable certificate validation. Certification authority certificates are loaded from the specified PEM file (default is system certificate storage if no argument is provided). Can be specified multiple times.

-E, --certfile ***path***
Path to a client certificate file.

-K, --keyfile ***path***
Path to a client key file.

-s, --sni ***string***
Use specified Server Name Indication.

Commands

server ***name*** **[*port*]**
: Specifies a receiving server of the dynamic update message. The *name* parameter can be either a host name or an IP address. If the *port* is not specified, the default port is used. The default port value can be controlled using the **-p** program option.

local ***address*** **[*port*]**
: Specifies outgoing *address* and *port*. If no local is specified, the address and port are set by the system automatically. The default port number is 0.

zone ***name***
: Specifies that all updates are done within a zone *name*. The zone name doesn't have a default and must be set explicitly.

origin ***name***
: Specifies fully qualified domain name suffix which is appended to non-fqd owners in update commands. The default is the terminal label (**.**).

class ***name***
: Sets *name* as the default class for all updates. If not used, the default class is IN.

ttl ***value***
: Sets *value* as the default TTL (in seconds). If not used, the default value is 3600.

key [*alg*:]*name key*
: Specifies the TSIG *key* named *name* to authenticate the request. An optional *alg* algorithm can be specified. This command has the same effect as the program option **-y**.

[prereq] nxdomain *name*
: Adds a prerequisite for a non-existing record owned by *name*.

[prereq] yxdomain *name*
: Adds a prerequisite for an existing record owned by *name*.

[prereq] nxrrset *name* [*class*] *type*
: Adds a prerequisite for a non-existing record of the *type* owned by *name*. Internet *class* is expected.

[prereq] yxrrset *name* [*class*] *type* [*data*]
: Adds a prerequisite for an existing record of the *type* owned by *name* with optional *data*. Internet *class* is expected.

[update] add *name* [*ttl*] [*class*] *type data*
: Adds a request to add a new resource record into the zone. Please note that if the *name* is not fully qualified domain name, the current origin name is appended to it.

[update] del[ete] *name* [*ttl*] [*class*] [*type*] [*data*]
: Adds a request to remove all (or matching *class*, *type* or *data*) resource records from the zone. There is the same requirement for the *name* parameter as in **update add** command. The *ttl* item is ignored.

show
: Displays current content of the update message.

send
: Sends the current update message and cleans the list of updates.

answer
: Displays the last answer from the server.

debug
: Enable debugging. This command has the same meaning as the **-d** program option.

exit
: End the program.

9.11.3 Notes

Options **-k** and **-y** can not be used simultaneously.

Neither *tsig-keygen(8)* nor *dnssec-keygen(1)* keyfile formats are supported. Use `keymgr(8)` to construct a string for **-y** or the file passed to **-k**.

Zone name/server guessing is not supported if the zone name/server is not specified.

An empty line doesn't send the update.

Interactive mode

The utility provides interactive mode with basic line editing functionality, command completion, and command history.

Interactive mode behavior can be customized in *~/.editrc*. Refer to `editrc(5)` for details.

Command history is saved in *~/.knsupdate_history*.

9.11.4 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.11.5 Examples

1. Send one update of the zone example.com to the server 192.168.1.1. The update contains two new records:

```
$ knsupdate
knsupdate> server 192.168.1.1
knsupdate> zone example.com.
knsupdate> origin example.com.
knsupdate> ttl 3600
knsupdate> add test1.example.com. 7200 A 192.168.2.2
knsupdate> add test2 TXT "hello"
knsupdate> show
knsupdate> send
knsupdate> answer
knsupdate> exit
```

9.11.6 See Also

`kdig(1)`, `khost(1)`, `keymgr(8)`, `editrc(5)`.

9.12 kxdpgun – DNS benchmarking tool

9.12.1 Synopsis

kxdpgun [*options*] **-i** *filename target*

9.12.2 Description

Powerful generator of DNS traffic, sending and receiving packets through XDP.

Queries are generated according to a textual file which is read sequentially in a loop until a configured duration elapses. The order of queries is not guaranteed. Responses are received (unless disabled) and counted, but not checked against queries.

The number of parallel threads is autodetected according to the number of queues configured for the network interface.

Parameters

filename
Path to the queries file. See the description below regarding the file format.

target
Either the domain name, IPv4 or IPv6 address of a remote target.

Options

-t, --duration ***seconds***
Duration of traffic generation, specified as a decimal number in seconds (default is 5.0).

-T, --tcp[=*debug_mode*]
Send queries over TCP. See the list of optional debug modes below.

-U, --quic[=*debug_mode*]
Send queries over QUIC. See the list of optional debug modes below.

-Q, --qps ***queries***
Number of queries-per-second (approximately) to be sent (default is 1000). The program is not optimized for low speeds at which it may lose communication packets. The recommended minimum speed is 2 packets per thread (Rx/Tx queue).

-b, --batch ***size***
Send more queries in a batch. Improves QPS but may affect the counterpart's packet loss (default is 10 for UDP and 1 for TCP/QUIC).

-r, --drop
Drop incoming responses. Improves QPS, but disables response statistics.

-p, --port ***number***
Remote destination port (default is 53 for UDP/TCP, 853 for QUIC).

-F, --affinity ***cpu_spec***
CPU affinity for all threads specified in the format [<cpu_start>][s<cpu_step>], where <cpu_start> is the CPU ID for the first thread and <cpu_step> is the CPU ID increment for next thread (default is 0s1).

-i, --infile ***filename***
Path to a file with query templates.

-B, --binary
Specify that input file is in binary format. This format is similar to the TCP DNS message format. The file contains records formatted as 2-octet length (network order) followed by a message in DNS wire format.

-I, --interface ***interface***
Network interface for outgoing communication. This can be useful in situations when the interfaces are in a bond for example.

-l, --local ***localIP*[*/prefix*]**
Override the auto-detected source IP address. If an address range is specified instead, various IPs from the range will be used for different queries uniformly (address range not supported in the QUIC mode).

-L, --mac-local
Override auto-detected local MAC address.

-R, --mac-remote
Override auto-detected remote MAC address.

-v, --vlan ***id***
Add VLAN 802.1Q header with the given id. VLAN offloading should be disabled.

-e, --edns-size ***size***
EDNS UDP payload size, range 512-4096 (default is 1232). Note that over XDP the maximum supported MTU is 1790.

-m, --mode ***mode***

Set the XDP mode. Supported values are:

- **auto** (default) – the XDP mode is selected automatically to achieve the best performance, which means that native driver support is preferred over the generic one, and zero-copy is used if available.
- **copy** – the XDP socket copy mode is forced even if zero-copy is available. This can resolve various driver issues, but at the cost of lower performance.
- **generic** – the generic XDP implementation is forced even if native implementation is available. This mode doesn't require support from the driver nor hardware, but offers the worst performance.

-G, --qlog ***path***

Generate qlog files in the directory specified by *path*. The directory has to exist.

This option is ignored if not in the QUIC mode. The recommended usage is with **--quic=R** or with low QPS. Otherwise, too many files are generated.

-j, --json

Print statistics formatted as json.

-S, --stats-period ***period***

Report statistics automatically every *period* milliseconds.

These reports contain only metrics collected in the given period.

-h, --help

Print the program help.

-V, --version

Print the program version. The option **-VV** makes the program print the compile time configuration summary.

Queries file format

Each line describes a query in the form:

query_name query_type [*flags*]

Where *query_name* is a domain name to be queried, *query_type* is a record type name, and *flags* is a single character:

E Send query with EDNS.

D Request DNSSEC (EDNS + DO flag).

TCP/QUIC debug modes

0

Perform full handshake for all connections (QUIC only).

1

Just send SYN (Initial) and receive SYN-ACK (Handshake).

2

Perform TCP/QUIC handshake and don't send anything, allow close initiated by counterpart.

3

Perform TCP/QUIC handshake and don't react further.

5

Send incomplete query (N-1 bytes) and don't react further.

7

Send query and don't ACK the response or anything further.

8

Don't close the connection and ignore close by counterpart.

9

Operate normally except for not ACKing the final FIN+ACK (TCP only).

R

Instead of opening a connection for each query, reuse connections.

Signals

Sending USR1 signal to a running process triggers current statistics dump to the standard output. In combination with **-S** may cause erratic printout timing.

9.12.3 Notes

Linux kernel 4.18+ is required.

The utility has to be executed under root or with these capabilities: CAP_NET_RAW, CAP_NET_ADMIN, CAP_SYS_ADMIN, CAP_IPC_LOCK, and CAP_SYS_RESOURCE (Linux < 5.11).

The utility allocates source UDP/TCP ports from the range 2000-65535.

Due to the multi-threaded program structure there are slight discrepancies in the timespan during which metrics are collected for any given thread. The statistics printouts ignore this and are thus ever-so-slightly inaccurate. The error margin decreases proportionally to the volume of data & timespan over which they are collected.

9.12.4 Exit values

Exit status of 0 means successful operation. Any other exit status indicates an error.

9.12.5 Examples

Manually created queries file:

```
abc6.example.com. AAAA
nxdomain.example.com. A
notzone. A
a.example.com. NS E
ab.example.com. A D
abcd.example.com. DS D
```

Queries file generated from a zone file (Knot DNS format):

```
cat ZONE_FILE | awk "{print \$1,\$3}" | grep -E "(NS|DS|A|AAAA|PTR|MX|SOA)$" | sort -
↪u -R > queries.txt
```

Basic usage:

```
# kxdpgun -i ~/queries.txt 2001:DB8::1
```

Using UDP with increased batch size:

```
# kxdpgun -t 20 -Q 1000000 -i ~/queries.txt -b 20 -p 8853 192.0.2.1
```

Using TCP:

```
# kxdpgun -t 20 -Q 100000 -i ~/queries.txt -T -p 8853 192.0.2.1
```

9.12.6 See Also

kdig(1).

CHAPTER

TEN

MIGRATION

10.1 Upgrade 2.4.x to 2.5.x

This chapter describes some steps necessary after upgrading Knot DNS from version 2.4.x to 2.5.x.

10.1.1 Building changes

The `--enable-dnstap` configure option now enables the dnstap support in *kdig* only! To build the dnstap query module, `--with-module-dnstap` have to be used.

Since Knot DNS version 2.5.0 each query module can be configured to be:

- disabled: `--with-module-MODULE_NAME=no`
- embedded: `--with-module-MODULE_NAME=yes`
- external: `--with-module-MODULE_NAME=shared` (excluding `dnsproxy` and `onlinesign`)

The `--with-timer-mapsize` configure option was replaced with the runtime `template.max-timer-db-size` configuration option.

10.1.2 KASP DB migration

Knot DNS version 2.4.x and earlier uses JSON files to store DNSSEC keys metadata, one for each zone. 2.5.x versions store those in binary format in a LMDB, all zones together. The migration is possible with the pykeymgr script:

```
$ pykeymgr -i path/to/keydir
```

The path to KASP DB directory is configuration-dependent, usually it is the `keys` subdirectory in the zone storage.

In rare installations, the JSON files might be spread across more directories. In such case, it is necessary to put them together into one directory and migrate at once.

10.1.3 Configuration changes

It is no longer possible to configure KASP DB per zone or in a non-default template. Ensure just one common KASP DB configuration in the default template.

As Knot DNS version 2.5.0 brings dynamically loaded modules, some modules were renamed for technical reasons. So it is necessary to rename all occurrences (module section names and references from zones or templates) of the following module names in the configuration:

```
mod-online-sign -> mod-onlinesign

mod-synth-record -> mod-synthrecord
```

10.2 Upgrade 2.5.x to 2.6.x

Upgrading from Knot DNS version 2.5.x to 2.6.x is almost seamless.

10.2.1 Configuration changes

The `dsa` and `dsa-nsec3-sha1` algorithm values are no longer supported by the *algorithm* option.

The `ixfr-from-differences` zone/template option was deprecated in favor of the *zonefile-load* option.

10.3 Upgrade 2.6.x to 2.7.x

Upgrading from Knot DNS version 2.6.x to 2.7.x is seamless if no obsolete configuration or module rosedb is used.

10.4 Upgrade 2.7.x to 2.8.x

Upgrading from Knot DNS version 2.7.x to 2.8.x is seamless.

However, if the previous version was migrated (possibly indirectly) from version 2.5.x, the format of the keys stored in Keys And Signature Policy Database is no longer compatible and needs to be updated.

The easiest ways to update how keys are stored in KASP DB is to modify with Keymgr version 2.7.x some of each key's parameters in an undamaging way, e.g.:

```
$ keymgr example.com. list
$ keymgr example.com. set <keyTag> created=1
$ keymgr example.com. set <keyTag2> created=1
...
```

10.5 Upgrade 2.8.x to 2.9.x

Upgrading from Knot DNS version 2.8.x to 2.9.x is almost seamless but check the following changes first.

10.5.1 Configuration changes

- Imperfect runtime reconfiguration of *udp-workers*, *tcp-workers*, and *listen* is no longer supported.
- Replaced options (with backward compatibility):

Old section	Old item name	New section	New item name
server	`tcp-reply-timeout` [s]	*server*	*tcp-remote-io-timeout* [ms]
server	`max-tcp-clients`	*server*	*tcp-max-clients*
server	`max-udp-payload`	*server*	*udp-max-payload*
server	`max-ipv4-udp-payload`	*server*	*udp-max-payload-ipv4*
server	`max-ipv6-udp-payload`	*server*	*udp-max-payload-ipv6*
template	`journal-db`	*database*	*journal-db*
template	`journal-db-mode`	*database*	*journal-db-mode*
template	`max-journal-db-size`	*database*	*journal-db-max-size*
template	`kasp-db`	*database*	*kasp-db*
template	`max-kasp-db-size`	*database*	*kasp-db-max-size*
template	`timer-db`	*database*	*timer-db*
template	`max-timer-db-size`	*database*	*timer-db-max-size*
zone	`max-journal-usage`	*zone*	*journal-max-usage*
zone	`max-journal-depth`	*zone*	*journal-max-depth*
zone	`max-zone-size`	*zone*	*zone-max-size*
zone	`max-refresh-interval`	*zone*	*refresh-max-interval*
zone	`min-refresh-interval`	*zone*	*refresh-min-interval*

- Removed options (no backward compatibility):
  - `server.tcp-handshake-timeout`
  - `zone.request-edns-option`
- New default value for:
  - *tcp-workers*
  - *tcp-max-clients*
  - *udp-max-payload*
  - *udp-max-payload-ipv4*
  - *udp-max-payload-ipv6*
- New DNSSEC policy option *rrsig-pre-refresh* may affect configuration validity, which is `rrsig-refresh + rrsig-pre-refresh < rrsig-lifetime`

10.5.2 Miscellaneous changes

- Memory use estimation via `knotc zone-memstats` was removed
- Based on https://tools.ietf.org/html/draft-ietf-dnsop-server-cookies the module *DNS Cookies* was updated to be interoperable
- Number of open files limit is set to 1048576 in upstream packages

10.6 Upgrade 2.9.x to 3.0.x

Knot DNS version 3.0.x is functionally compatible with 2.9.x with the following exceptions.

10.6.1 ACL

Configuration option *update-owner-name* is newly FQDN-sensitive. It means that values `a.example.com` and `a.example.com.` are not equivalent.

10.6.2 Module synthrecord

Reverse IPv6 address shortening is enabled by default. For example, the module generates:

```
dynamic-2620-0-b61-100--1.test. 400 IN AAAA 2620:0:b61:100::1
```

instead of:

```
dynamic-2620-0000-0b61-0100-0000-0000-0000-0001.test. 400 IN AAAA 2620:0:b61:100::1
```

10.6.3 Query module API change

The following functions require additional parameter (thread id – `qdata->params->thread_id`) on the second position:

```
knotd_mod_stats_incr()
knotd_mod_stats_decr()
knotd_mod_stats_store()
```

10.6.4 Building notes

- The embedded library *LMDB* is no longer part of the source code. Almost every modern operating system has a sufficient version of this library.
- DoH support in kdig requires optional library *libnghttp2*.

- XDP support on Linux requires optional library *libbpf >= 0.0.6*. If not available, an embedded library can be used via `--enable-xdp=yes` configure option.

10.7 Upgrade 3.0.x to 3.1.x

Knot DNS version 3.1.x is functionally compatible with 3.0.x with the following exceptions.

10.7.1 Configuration changes

- Automatic SOA serial incrementation (`zonefile-load: difference-no-serial`) requires having full zone stored in the journal (`journal-content: all`). This change is necessary for reliable operation.
- Replaced options (with backward compatibility):

Old section	Old item name	New section	New item name
server	`listen-xdp`	*xdp*	*listen*

- Ignored obsolete options (with a notice log):
  - `server.max-ipv4-udp-payload`
  - `server.max-ipv6-udp-payload`
  - `server.max-udp-payload`
  - `server.max-tcp-clients`
  - `server.tcp-reply-timeout`
  - `zone.max-journal-depth`
  - `zone.max-journal-usage`
  - `zone.max-refresh-interval`
  - `zone.min-refresh-interval`
  - `zone.max-zone-size`
  - `template.journal-db`
  - `template.kasp-db`
  - `template.timer-db`
  - `template.max-journal-db-size`
  - `template.max-timer-db-size`
  - `template.max-kasp-db-size`
  - `template.journal-db-mode`
- Silently ignored obsolete options:
  - `server.tcp-handshake-timeout`
  - `zone.disable-any`

10.7.2 Zone backup and restore

The online backup format has changed slightly since 3.0 version. For zone-restore from backups in the previous format, it's necessary to set the *-f* option. Offline restore procedure of zone files from online backups is different than what it was before. The details are described in *Data and metadata backup*.

10.7.3 Building notes

- The configure option `--enable-xdp=yes` has slightly changed its semantics. It first tries to find an external library *libbpf*. If it's not detected, the embedded one is used instead.
- The kxdpgun tool also depends on library *libmnl*.

10.7.4 Packaging

Users who use module *geoip* or *dnstap* might need installing an additional package with the module.

10.8 Upgrade 3.1.x to 3.2.x

Knot DNS version 3.2.x is functionally compatible with 3.1.x with the following exceptions.

10.8.1 Configuration changes

- Default value for:
  - *journal-max-depth* was lowered to 20. This change may trigger journal history merging.
  - *nsec3-iterations* was lowered to 0. This change may trigger complete NSEC3 chain reconstruction!
  - *rrsig-refresh* is set to *propagation-delay* + "zone maximum TTL". This change affects effective RRSIG lifetime!
- New checks:
  - *rrsig-refresh* must be high enough to ensure all RRSIGs are refreshed before their expiration.
  - A notice log message is emitted if *algorithm* is deprecated.
- Ignored obsolete option (with a notice log):
  - `server.listen-xdp`

10.8.2 Utilities:

- *knotc* prints simplified zones status by default. Use `-e` for full output.
- *keymgr* uses the brief key listing mode by default. Use `-e` for full output.
- *keymgr* parameter `-d` was renamed to `-D`.
- *kjournalprint* parameter `-c` was renamed to `-H`.

10.8.3 Packaging

- Linux distributions Debian 9 and Ubuntu 16.04 are no longer supported.
- Packages for CentOS 7 are stored in a separate COPR repository `cznic/knot-dns-latest-centos7`.
- Utilities *kzonecheck*, *kzonesign*, and *knsec3hash* are located in a new `knot-dnssecutils` package.

10.8.4 Python

- Compatibility with Python 2 was removed.

10.9 Upgrade 3.2.x to 3.3.x

There are some changes between Knot DNS versions 3.3.x and 3.2.x that should be taken into consideration before upgrading.

10.9.1 Configuration changes

- The configuration option `xdp_quic-log` has been replaced with a more general logging option *quic*, which applies to both conventional QUIC and QUIC over XDP.

10.9.2 Functionality

- Responses to forwarded DDNS requests are signed with the local TSIG key instead of the remote one if the TSIG secret is known. To forward DDNS requests signed with a locally unknown key, an ACL rule for the action `update` without a key must be configured for the zone.
- Addresses for the remote which is considered the source of the NOTIFY are tried in the order they are specified in the remote configuration, regardless of which address the NOTIFY came from.
- Semantic checks don't allow DS record at non-delegation point.
- The `Version:` prefix has been removed from the `status version` control output.
- DNS over QUIC requires `doq` ALPN. The previous versions `doq-i03` and `doq-i11` are no longer supported.

10.9.3 XDP

The embedded library `libbpf` has been removed from the project, and an external one is required for the XDP support. If `libbpf` is version 1.0 or higher, an additional library `libxdp` is also required.

10.9.4 Query module API change

The function `knotd_qdata_local_addr()` only takes one parameter.

10.10 Upgrade 3.3.x to 3.4.x

There are the following changes between Knot DNS versions 3.4.x and 3.3.x.

10.10.1 DNSSEC

- DNSSEC validation fails if the remaining RRSIG validity is shorter than the corresponding *rrsig-refresh* value.
- SKR verification fails if the end of a DNSKEY RRSIG validity period doesn't cover the next DNSKEY snapshot.
- If DNSSEC signing is enabled, the outbound request's EDNS expire value is lowered to the earliest RRSIG expiration if it is higher.

10.10.2 Semantic checks

- Just one SOA record is required.
- Unified DNAME and CNAME semantic checks (see *Handling CNAME and DNAME-related updates*).

10.10.3 Configuration changes

- The server no longer allows concurrent control zone and configuration transactions.
- The server no longer allows opening a zone transaction when a blocking command is running.
- Removed already ignored obsolete options:
  - `server.max-ipv4-udp-payload`
  - `server.max-ipv6-udp-payload`
  - `server.max-udp-payload`
  - `server.max-tcp-clients`

  - `server.tcp-handshake-timeout`
  - `server.tcp-reply-timeout`
  - `server.listen-xdp`
  - `xdp.quic-log`
  - `zone.max-journal-depth`
  - `zone.max-journal-usage`
  - `zone.max-refresh-interval`
  - `zone.min-refresh-interval`
  - `zone.max-zone-size`
  - `zone.disable-any`
  - `template.journal-db`
  - `template.kasp-db`
  - `template.timer-db`
  - `template.max-journal-db-size`
  - `template.max-timer-db-size`
  - `template.max-kasp-db-size`
  - `template.journal-db-mode`

10.10.4 Utilities

- Changed defaults:
  - *kdig*: enabled `+edns` and `+bufsize=1232`
- Removed legacy parameters:
  - *keymgr*: `--brief`
  - *kjournalprint*: `--no-color`
  - *kjournalprint*: database specification without `--dir`
  - *kjournalprint*: database specification without `--dir`

10.10.5 Documentation

- Info pages are no longer supported.

10.10.6 Building notes

- A GCC or LLVM Clang compiler with C11 support is required.
- Minimum required *GnuTLS* version is 3.6.10.
- *Libidn* version 1 is no longer supported.
- *Liburcu* must be available via pkg-config.
- Linux distributions CentOS 7, Debian 10, and Ubuntu 18.04 are no longer supported.

10.11 Upgrade 3.4.x to 3.5.x

There are the following changes between Knot DNS versions 3.5.x and 3.4.x.

10.11.1 DNSSEC

- The default value of *nsec3-salt-length* was lowered to 0, which means no extra SHA-1 iterations are used. If the option value is not explicitly set, zones that rely on the default will be re-signed upon upgrade.

10.11.2 Utilities

- *kzonecheck*'s zone origin auto-detection now uses the owner name of the first SOA record from the checked zone file. If the owner isn't a FQDN, the zone file name (without possible **.zone** suffix) is used as the initial zone origin.
- *knot-exporter*: the following metrics, labes, or units have been changed:

Old metric (old labels), New metric (new labels), New unit		
`knot_memory_usage(section, type)`	`knot_memory_usage(pid)`	`bytes`
`knot_*(section, type)`	`knot_stats_*(module, type)`	
`knot_*(section)`	`knot_stats_*(module, type="")`	
`knot_*(zone, section, type)`	`knot_zone_stats_*(zone, module, type)`	
`knot_*(zone, section)`	`knot_zone_stats_*(zone, module, type="")`	
`knot_zone_stats_*(zone)`	`knot_zone_status_*(zone)`	
`knot_zone_refresh(zone)`	`knot_zone_refresh(zone)`	`seconds`
`knot_zone_retry(zone)`	`knot_zone_retry(zone)`	`seconds`
`knot_zone_expiration(zone)`	`knot_zone_expiration(zone)`	`seconds`

10.12 Knot DNS for BIND users

10.12.1 Automatic DNSSEC signing

Migrating automatically signed zones from BIND to Knot DNS requires copying up-to-date zone files from BIND, importing existing private keys, and updating server configuration:

1. To obtain current content of the zone which is being migrated, request BIND to flush the zone into the zone file: `rndc sync example.com`.

   Note

   If dynamic updates (DDNS) are enabled for the given zone, you might need to freeze the zone before flushing it. That can be done similarly:

   ```
   $ rndc freeze example.com
   ```

2. Copy the fresh zone file into the zones *storage* directory of Knot DNS.
3. Import all existing zone keys into the KASP database. Make sure that all the keys were imported correctly:

   ```
   $ keymgr example.com. import-bind path/to/Kexample.com.+013+11111
   $ keymgr example.com. import-bind path/to/Kexample.com.+013+22222
   $ ...
   $ keymgr example.com. list
   ```

   Note

> If the server configuration file or database is not at the default location, add a configuration parameter (-c or -C). See *keymgr* for more info about required access rights to the key files.

4. Follow *Automatic DNSSEC signing* steps to configure DNSSEC signing.

CHAPTER

ELEVEN

APPENDICES

11.1 Compatible PKCS #11 Devices

This section has informative character. Knot DNS has been tested with several devices which claim to support PKCS #11 interface. The following table indicates which algorithms and operations have been observed to work. Please notice minimal GnuTLS library version required for particular algorithm support.

	Key generate	Key import	ED25519 256-bit	ECDSA 256-bit	ECDSA 384-bit	RSA 1024-bit	RSA 2048-bit	RSA 4096-bit
Feitian ePass 2003	**yes**	no	no	no	no	**yes**	**yes**	no
SafeNet Network HSM (Luna SA 4)	**yes**	no	no	no	no	**yes**	**yes**	**yes**
SoftHSM 2.0[1]	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Trustway Proteccio NetHSM	**yes**	ECDSA only	no	**yes**	**yes**	**yes**	**yes**	**yes**
Ultra Electronics CIS Keyper Plus (Model 9860-2)	**yes**	RSA only	no	**yes**	**yes**	**yes**	**yes**	**yes**
Utimaco SecurityServer (V4)[2]	**yes**	**yes**	no	**yes**	**yes**	**yes**	**yes**	**yes**

[1] Algorithms supported depend on support in OpenSSL on which SoftHSM relies. A command similar to the following may be used to verify what algorithms are supported: `$ pkcs11-tool --modul /usr/lib64/pkcs11/libsofthsm2.so -M`.

[2] Requires setting the number of background workers to 1!

INDEX