Cascade User Manual

Release 0.1.0-beta2-dev

NLnet Labs <cascade@nlnetlabs.nl>

Jun 08, 2026

GETTING STARTED

A friendly, stand-alone *DNSSEC* signing solution designed to communicate what it's doing and is easy to interact with.

Cascade is currently in the **beta** development phase. We would love for you to *get to know Cascade* and are eager to hear *your experiences* so we can improve every aspect. Please consider the current *Known Limitations*.

Cascade has the following design goals:

Flexibility
: The Cascade pipeline runs as a single binary and does not require additional database software. Install with ease using a *binary package* or build from source. Run on-premise or in the cloud, with keys on disk or a *Hardware Security Module* of your choice.

Sensible defaults
: *Get started easily* with default settings based on industry best practices.

Controllability
: Cascade gives you tight control over *key management* automation and the DNSSEC signing process.

Observability
: With Cascade you cut out the guesswork. *Review Hooks* allow you to use your preferred solutions to automate verification of the unsigned and signed zone, before publishing it.

Open-source with professional support services
: NLnet Labs offers professional support and consultancy services with a service-level agreement.

CHAPTER

ONE

FEEDBACK

Our goal is to gather operator feedback. Don't be shy and reach out by creating a GitHub issue, sending us an email, finding us in the NLnet Labs DNS channel on the DNS OARC Mattermost server, or mentioning us on Mastodon. Or if you want to discuss operational experiences, if you have feedback or if you want to request a feature, please use the NLnet Labs Community Forum.

Examples of things we're interested in:

- If these documentation pages don't answer your question, tell us what we missed.
- Performance and memory usage are expected to improve but if you think it won't meet your needs, tell us about your use case.
- Not all intended functionality has been implemented at this point. If a feature that you need is missing, please let us know.
- We are actively working to shape the user experience to operator needs. We have a lot more ideas for improvement and we'd love to hear yours too.
- Do tell us about your positive experiences. We particularly appreciate hearing O/S, HSM and size/number of zones you worked with.

1.1 An Intro to DNSSEC

DNS Security Extensions (DNSSEC) protect end users against forged or modified DNS responses, both deliberate and accidental, by digitally signing DNS record data to allow its authenticity to be verified.

Though DNSSEC is often perceived as a complicated topic, Cascade tries to make the experience as understandable and robust as possible. In this section we explain the basic concepts that you will encounter in the DNSSEC signing process.

1.1.1 The Value of DNSSEC

DNSSEC mitigates attacks that compromise the integrity and authenticity of DNS data. These can be categorised into roughly four types:

1. DNS Spoofing: the act of sending a forged DNS response to a user or resolver.
2. Cache Poisoning: an attacker injects fake DNS data into a DNS resolver's cache. When other users request the same record, the resolver serves the malicious, cached information. DNSSEC's *validation* process prevents this by ensuring the cached data is cryptographically signed by the authoritative source.
3. On-Path Attacks: DNSSEC helps prevent attackers from intercepting and altering DNS queries and responses, ensuring data integrity during transit.

4. Authenticated Denial of Existence: attackers can try to exploit vulnerabilities to forge an "unauthenticated" response to a non-existent record (NXDOMAIN). DNSSEC uses *NSEC* or *NSEC3* records to provide a cryptographically signed proof that a record does not exist, making the zone resistant to this type of attack.

DNSSEC uses digital signatures to ensure the response is authentic and has not been tampered with. By verifying DNS responses, DNSSEC prevents malicious actors from for example redirecting users to fake or malicious websites.

1.1.2 How DNSSEC Works

DNSSEC adds an extra layer of information to DNS responses, allowing resolvers to verify that each answer is authentic. This is done by adding a digital signature to resource records grouped by type (*RRsets*) in a zone. These digital signatures are stored in DNS nameservers alongside common record types like A, AAAA, MX, CNAME, etc.

The keys that are used for DNSSEC are asymmetric, such as RSA (Rivest-Shamir-Adleman cryptosystem) and ECDSA (Elliptic Curve Digital Signature Algorithm). The private part is used for signing and must be kept secret all all times.

DNSSEC adds these record types:

- RRSIG, which contains a cryptographic signature over an *Resource Record Set (RRset)*
- DNSKEY, which contains a public signing key
- DS, which stores a hashed representation of a DNSKEY record
- NSEC and NSEC3, which offer proof that a DNS record *doesn't* exist
- CDNSKEY and CDS, allowing a child zone to request updates to DS record(s) in the parent zone

Each DNSSEC-enabled zone, including the root `(.)`, a Top-Level Domain (TLD) such as `(.com)`, or a Second-Level Domain (SLD) such as `(.example.com)` has its public key stored in DNSKEY records. The corresponding private key is used to generate so-called Resource Record Signature (RRSIG) records, which are cryptographic signatures over the data in your zone.

Resource Record Sets

Rather than signing each resource record individually, the DNSKEY is used to sign a *Resource Record Set (RRset)*, which is a collection of individual DNS records that share the same name and type. For example, all the A (IPv4) records for a specific domain are grouped into a single RRset. RRsets are the fundamental records for DNSSEC signing, as the entire set is signed digitally to ensure its integrity.

Zone Signing Keys

Each zone in DNSSEC has a *Zone signing key (ZSK)* set. The private portion of this key set key digitally signs each RRset in the zone. The the public portion of the set is used to verify the signature. You create digital signatures for each RRset using the private ZSK and store them on your nameserver as RRSIG records.

We now almost have all the puzzle pieces in place for a *Validating resolver* to verify the signatures. The remaining part you need to make available is the public part of the ZSK by adding it to your nameserver in a DNSKEY record.

Now, the resolver can use the RRset, RRSIG, and public ZSK to validate if the response is authentic.

Key Signing Keys

In addition to a zone signing key, DNSSEC name servers also have a *Key signing key (KSK)*. The KSK only signs the *apex* DNSKEY RRset in a zone. The KSK signs the public ZSK, creating an RRSIG for the DNSKEY.

The public part of the KSK in published in another DNSKEY record. Both the public KSK and public ZSK are signed by the private KSK. Validating resolvers can use the public KSK to validate the public ZSK.

Building a Chain of Trust

DNSSEC relies on a chain of trust by creating a hierarchical system where trust in each DNS zone is bootstrapped from the zone above it, starting from the root zone, which acts as a trusted starting point with signatures that ship with DNS resolver software. Without a chain of trust, a DNSSEC-validating resolver wouldn't know where to begin trusting DNS data.

The chain of trust works though the interaction of two key DNSSEC record types: DNSKEY records and Delegation Signer (DS) records. The DNSSEC Trust Anchor is the top of this chain, representing a public *Key signing key (KSK)* that is implicitly trusted by a DNSSEC-validating resolver.

A parent zone doesn't directly sign the data in a child zone. To establish a secure delegation, the parent zone signs a hash of the child zone's KSK. This is called a DS record.

To do this, the operator of a *child zone* (such as example.com) generates a KSK and then calculates a hash over it. This hash (aka digest) is then given to the *parent zone* (in this case .com). The parent zone publishes this digest as a DS record within its own zone file and signs it with its own Zone Signing Key. This DS record effectively acts as a secure pointer to the child zone's KSK. This process is repeated all the way down the hierarchy.

Validation

The chain of trust must remain unbroken at all times. If, for example, a DS record points to an incorrect DNSKEY, or if a signature is invalid or missing, resolvers will not be able to verify the data. This results in a *"bogus"* status, telling you that the DNS record does not pass DNSSEC authentication checks.

The other possible DNSSEC validation states are *"secure"*, *"insecure"* and *"indeterminate"*.

Variants

Some operators prefer to combine the role of the *Key Signing Key (KSK)* with that of the *Zone Signing Key (ZSK)*. In this setup, the DS record points to a so-called *Combined Signing Key (CSK)* that signs all RRsets, not just the DNSKEY RRset.

1.2 Before You Start

1.2.1 Placement

Cascade is what is known as a "hidden signer". It is meant to run as a dedicated server with restricted access that takes local zones or zones received from an upstream primary nameserver, signs those zones and makes the results available to downstream, Internet facing *secondary* nameservers.

> **Warning**
>
> Do *not* run Cascade as an Internet facing service, as it is designed to answer a limited subset of the DNS protocol that a full authoritative nameserver must support.

One possible authoritative server that could be used up and downstream of Cascade is NSD, but any solution can be used provided that it supports transferring zones via XFR zone transfers to and from Cascade.

1.2.2 Intended Audience

Cascade is currently targeted for use by TLD operators, but will evolve over time to cater to other audiences. As a successor to OpenDNSSEC, Cascade is clearly intended to offer continuity and a migration path for current users, but Cascade will also offer superior performance, flexibility and user experience.

Important

Cascade does *not* require the use of a Hardware Security Module (HSM). It can make use of on-disk key files and, if desired, use PKCS#11 and KMIP compatible HSMs.

1.2.3 The Moving Parts

Cascade consists of three main components and an optional fourth:

- The **cascaded** daemon for receiving zone data, signing it, and serving the signed result. It supports *Review Hooks* during the processing pipeline, allowing you to use your preferred solutions to automate verification of the unsigned and signed zone, before publishing it.
- The **cascade** command line interface (CLI) for controlling and interacting with the **cascaded** daemon.
- A tool called **dnst keyset**, which is responsible for the *key management* of Cascade, including automation of key rolls. It is invoked as needed by the **cascaded** daemon.
- The *optional* **cascade-hsm-bridge** daemon, which is only required when using a PKCS#11 compatible *HSM*.

1.2.4 Supported Inputs/Outputs

Cascade supports:

- Receiving zone data via AXFR or IXFR *zone transfers*, or from on-disk files.
- Publishing data via AXFR or IXFR.

On-disk files, while not supported directly, could be achieved by AXFR of the signed zone to an on-disk file.

Important

Fully automatic key rolls are enabled by default. For this to work, Cascade requires access to all nameservers of the zone and the parent zone. If this is not available, make sure to *disable automatic key rolls*.

1.2.5 System Requirements

Cascade is able to run with fairly limited CPU and memory. Exact figures are not yet available, but in principle more CPU cores allow for more parallel operations and more memory makes it possible to load and sign larger zones.

Hint

During testing, Cascade currently uses using about 30GiB of RAM when signing a 1GiB zone file with about 25 million resource records and adding roughly 10 million records while signing.

Right now, signing speed is not likely to be a bottleneck for most use cases, but there are many speed improvements in the pipeline, especially when using an HSM.

Cascade can currently be used by operators with at most a few small to medium size zones. As development progresses, it will also support operators with very large zones or operators with many zones.

Cascade is *not* yet intended for operation as a clustered deployment.

1.3 Architecture

1.3.1 The Pipeline

The Cascade pipeline runs as a single binary and no additional database software is required. Zone changes cascade through several stages, letting you review and approve at each step:

Fig. 1: A schematic view of of the Cascade pipeline with the verification stages.

1.3.2 Flexible Signing

Cascade does not require a *Hardware security module (HSM)* to operate. Cascade is able to use OpenSSL and ring software cryptography to generate signing keys and to cryptographically sign DNS *RRset* data, storing the generated keys in on-disk files.

For operators wishing to use an HSM, Cascade can connect directly to KMIP compatible HSMs, or to PKCS#11 compatible HSMs via our **`cascade-hsm-bridge`** daemon, which is installed automatically as part of our Cascade packages.

Hint

Separating the main Cascade and cascade-hsm-bridge daemons avoids running untrusted third-party code inside the main Cascade process. This eliminates a source of potential instability and unpredictable behaviour, as well as limiting resource usage.

1.3.3 Bespoke Zone Verification

Using *Review Hooks*, Cascade supports optional verification of your zone data at two critical stages: verification of the unsigned zone, and verification of the signed zone. In both cases verification consists of executing an operator supplied script or application which can verify the zone using whatever mechanisms are required to satisfy your policy.

Verification of the zone can be done by retrieving it using the *zone transfer* protocol from dedicated "review" name-servers within Cascade, either verifying the zone directly or writing the zone to disk for verification by tools that only support working with files.

On completion of the verification process, approval or rejection is signalled back to Cascade via the script exit code.

Rejecting a zone "soft" halts the Cascade pipeline for the zone, preventing it from cascading further down the pipeline, but allowing a newer version of the zone to be completely processed (unless that too should fail verification).

Serious errors in the pipeline may result in a "hard" halt for the pipeline of a zone, preventing any further processing of that zone for the current and future versions until an operator manually resumes the pipeline.

1.3.4 Managing State

Cascade stores its state in on-disk files in JSON format, by default at various locations under a single parent directory. No additional database software is required. As such, state is human-readable and easily backed up. Note however, that state files are not intended to be modified by the user.

Some configuration is done via the Cascade command line interface (CLI), such as adding zones and HSMs. Other configuration is done by editing policy and application configuration files and instructing the Cascade daemon via the CLI to reload them.

The Cascade daemon updates its on-disk state files periodically, and when signalled to stop, reloading them on next start.

As Cascade outsources PKCS#11 support to **cascade-hsm-bridge**, it does not require access to related configuration files or other vendor specific module dependencies.

1.3.5 Robustness

Cascade is written in the Rust programming language making it significantly less likely to crash or suffer from memory safety issues, and at the same time making it easier to leverage the higher core count of modern computers via Rust's "fearless concurrency" when needed.

1.4 Installation

1.4.1 Prerequisites

To use Cascade a v0.2.0 version of **dnst** is required. The Cascade DEB and RPM packages automatically ensure that the appropriate version of **dnst** is installed.

To use automatic keyrolls (which are on by default) the Cascade host machine will need to have IPv6 connectivity if any of its nameservers or parent nameservers have AAAA records.

1.4.2 Binary Packages

Getting started with Cascade is really easy by installing a binary package for either Debian and Ubuntu or for Red Hat Enterprise Linux (RHEL) and compatible systems such as Rocky Linux. Alternatively, you can run with Docker.

You can also build Cascade from the source code using Cargo, Rust's build system and package manager. Cargo lets you run Cascade on almost any operating system and CPU architecture. Refer to the *Building From Source* section to get started.

Debian

Ubuntu

RHEL

Docker

To install a Cascade package, you need the 64-bit version of one of these Debian versions:

- Debian Trixie 13
- Debian Bookworm 12
- Debian Bullseye 11

Packages are available for the `amd64/x86_64` architecture only.

First update the **apt** package index:

```
sudo apt update
```

Then install packages to allow **apt** to use a repository over HTTPS:

```
sudo apt install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

Add the GPG key from NLnet Labs:

```
curl -fsSL https://packages.nlnetlabs.nl/aptkey.asc | sudo gpg --dearmor -o /etc/apt/
↪keyrings/nlnetlabs-archive-keyring.gpg
```

Now, use the following command to set up the *proposed* repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/debian \
$(lsb_release -cs)-proposed main" | sudo tee /etc/apt/sources.list.d/nlnetlabs-proposed.
↪list > /dev/null
```

Update the **apt** package index once more:

```
sudo apt update
```

You can now install Cascade with:

```
sudo apt install cascade
```

After installing, refer to the *Quick Start* to get started.

Once you're ready, start Cascade with:

```
sudo systemctl start cascaded
```

You can also configure Cascade to start at boot:

```
sudo systemctl enable cascaded
```

You can check the status of Cascade with:

```
sudo systemctl status cascaded
```

You can view the logs with:

```
sudo journalctl --unit=cascaded
```

To install a Cascade package, you need the 64-bit version of one of these Ubuntu versions:

- Ubuntu Resolute 26.04 (LTS)
- Ubuntu Noble 24.04 (LTS)
- Ubuntu Jammy 22.04 (LTS)

Packages are available for the `amd64/x86_64` architecture only.

First update the **apt** package index:

```
sudo apt update
```

Then install packages to allow **apt** to use a repository over HTTPS:

```
sudo apt install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

Add the GPG key from NLnet Labs:

```
curl -fsSL https://packages.nlnetlabs.nl/aptkey.asc | sudo gpg --dearmor -o /etc/apt/
↪keyrings/nlnetlabs-archive-keyring.gpg
```

Now, use the following command to set up the *proposed* repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/ubuntu \
$(lsb_release -cs)-proposed main" | sudo tee /etc/apt/sources.list.d/nlnetlabs-proposed.
↪list > /dev/null
```

Update the **apt** package index once more:

```
sudo apt update
```

You can now install Cascade with:

```
sudo apt install cascade
```

After installing, refer to the *Quick Start* to get started.

Once you're ready, start Cascade with:

```
sudo systemctl start cascaded
```

You can also configure Cascade to start at boot:

```
sudo systemctl enable cascaded
```

You can check the status of Cascade with:

```
sudo systemctl status cascaded
```

You can view the logs with:

```
sudo journalctl --unit=cascaded
```

To install a Cascade package, you need Red Hat Enterprise Linux (RHEL) 8, 9 or 10 or compatible operating system such as Rocky Linux. Packages are available for the `amd64/x86_64` architecture only.

First create a file named `/etc/yum.repos.d/nlnetlabs-testing.repo`, enter this configuration and save it:

Tip

On Fedora systems replace $releasever with 10 (or 8 or 9 if 10 is too new for your Fedora) as there is no repository with Fedora numbers, e.g. 42, in our package repository.

```
[nlnetlabs-testing]
name=NLnet Labs Testing
baseurl=https://packages.nlnetlabs.nl/linux/centos/$releasever/proposed/$basearch
enabled=1
```

Add the GPG key from NLnet Labs:

```
sudo rpm --import https://packages.nlnetlabs.nl/aptkey.asc
```

You can now install Cascade with:

```
sudo yum install -y cascade
```

If you want to use a PKCS#11-based HSM, also install cascade-hsm-bridge with:

```
sudo yum install -y cascade-hsm-bridge
```

After installing, refer to the *Quick Start* to get started.

Once you're ready, start Cascade with:

```
sudo systemctl start cascaded
```

You can also configure Cascade to start at boot:

```
sudo systemctl enable cascaded
```

You can check the status of Cascade with:

```
sudo systemctl status cascaded
```

You can view the logs with:

```
sudo journalctl --unit=cascaded
```

Note

Docker images are coming soon.

1.4.3 Updating

Danger

In its current beta version form Cascade will likely report errors if a newer version is started using existing state and policy files created by an older version.

Before updating, delete all state and policy files. Note that **this will delete signing keys stored on disk**. Signing keys stored in HSMs will **not** be affected but Cascade will no longer know about them. If HSM signing keys that are left behind are no longer wanted, you will need to remove them manually.

First stop Cascade, using systemd if in use on your system:

```
sudo systemctl stop cascaded.socket
sudo systemctl stop cascaded.service
```

Or by killing the Cascade daemon process otherwise:

```
pkill cascaded
```

Next, delete the state and policy files:

(if you modified any of the filesystem locations specified in your Cascade config file, use the updated paths instead of the default paths shown in these instructions)

```
sudo rm -R /var/lib/cascade
sudo rm -R /etc/cascade/policies
```

Debian

Ubuntu

RHEL

Docker

To update an existing Cascade installation, first update the repository using:

```
sudo apt update
```

You can use this command to get an overview of the available versions:

```
sudo apt policy cascade
```

You can upgrade an existing Cascade installation to the latest version using:

```
sudo apt --only-upgrade install cascade
```

To update an existing Cascade installation, first update the repository using:

```
sudo apt update
```

You can use this command to get an overview of the available versions:

```
sudo apt policy cascade
```

You can upgrade an existing Cascade installation to the latest version using:

```
sudo apt --only-upgrade install cascade
```

To update an existing Cascade installation, you can use this command to get an overview of the available versions:

```
sudo yum list --showduplicates cascade
```

You can update to the latest version using:

```
sudo yum update cascade
```

Note

Docker images are coming soon.

1.5 Building From Source

There are three things you need to build Cascade: a C toolchain, OpenSSL, and Rust. You can run Cascade on any operating system and CPU architecture where you can fulfil these requirements.

1.5.1 Dependencies

To get started, you need a C toolchain and OpenSSL because the cryptographic primitives used by Cascade require it. You also need Rust, because that's the programming language that Cascade has been written in. Additionally, you need a few tools used by Cascade. However, they are installed together with Cascade in the steps below.

C Toolchain

Some of the libraries Cascade depends on require a C toolchain to be present. Your system probably has some easy way to install the minimum set of packages to build from C sources. For example, this command will install everything you need on Debian/Ubuntu:

```
apt install build-essential
```

If you are unsure, try to run **cc** on a command line. If there is a complaint about missing input files, you are probably good to go.

OpenSSL

Your system will likely have a package manager that will allow you to install OpenSSL in a few easy steps. On Debian and Ubuntu this is usually **libssl-dev**. You will also need **pkg-config** for discovery of system libraries. On Red Hat Enterprise (RHEL) you will most likely need to install **openssl-devel**.

On Debian-like Linux distributions it should be as simple as running:

```
apt install libssl-dev pkg-config
```

Rust

The Rust compiler runs on, and compiles to, a great number of platforms, though not all of them are equally supported. The official Rust Platform Support page provides an overview of the various support levels.

While some system distributions include Rust as system packages, Cascade relies on a relatively new version of Rust, currently {'workspace': True} or newer. We therefore suggest using the canonical Rust installation via a tool called **rustup**.

Assuming you already have **curl** installed, you can install **rustup** and Rust by simply entering:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Alternatively, visit the Rust website for other installation methods.

> **Tip**
>
> During installation **rustup** will attempt to configure the `PATH`. Modifications to `PATH` may not take effect until the console is restarted, or the user is logged out, or it may not succeed at all. If, after installation, running **rustc --version** in the console fails, this is the most likely reason.

1.5.2 Building

In Rust, a library or executable program such as Cascade is called a *crate*. Crates are published on crates.io, the Rust package registry. Cargo is the Rust package manager. It is a tool that allows Rust packages to declare their various dependencies and ensure that you'll always get a repeatable build.

Cargo fetches and builds Cascade's dependencies into an executable binary for your platform. By default, you install from crates.io, but you can for example also install from a specific Git URL, as explained below.

Installing the latest Cascade (and **dnst**, a runtime dependency) is as simple as running:

```
cargo install --locked --git https://github.com/nlnetlabs/cascade cascade cascaded
cargo install --locked --bin dnst --git https://github.com/nlnetlabs/dnst dnst
```

The command will build Cascade and install it in the same directory that Cargo itself lives in, likely `$HOME/.cargo/bin`. Ensure this directory is in your PATH so you can run Cascade immediately.

If you want to use a PKCS#11 compatible *Hardware Security Module (HSM)* with Cascade, also install the cascade-hsm-bridge with:

```
cargo install --locked --git https://github.com/nlnetlabs/cascade-hsm-bridge
```

Finally, before running Cascade you will need to create a few directories and Cascade's config file. Create the directory where you want to store the config (let's say `./cascade` for this example), and generate an example config file:

```
mkdir ./cascade
cascade template config > ./cascade/config.toml
```

Then update the `config.toml` to use the appropriate paths.

Updating

> **Tip**
>
> Read the *general updating instructions* first.

If you want to update to the latest version of Cascade, it's recommended to update Rust itself as well, using:

```
rustup update
```

Use the `--force` option to overwrite an existing version with the latest Cascade release:

```
cargo install --locked --force --git https://github.com/nlnetlabs/cascade cascade␣
↪cascaded
cargo install --locked --force --bin dnst --git https://github.com/nlnetlabs/dnst
```

If you are using cascade-hsm-bridge, update it with:

```
cargo install --locked --force --git https://github.com/nlnetlabs/cascade-hsm-bridge
```

Installing Specific Versions

If you want to install a specific version of Cascade using Cargo, explicitly use the `--version` option. If needed, use the `--force` option to overwrite an existing version:

```
cargo install --locked --force --git https://github.com/nlnetlabs/cascade --tag 0.1.0-
↪alpha3 cascade cascaded
```

Make sure to install a compatible version of **dnst**.

All new features of Cascade are built on a branch and merged via a pull request, allowing you to easily try them out using Cargo. If you want to try a specific branch from the repository you can use the `--git` and `--branch` options:

```
cargo install --git https://github.com/NLnetLabs/cascade.git --branch main cascade␣
↪cascaded
```

See also

For more installation options refer to the Cargo book.

1.6 Quick Start

After *installing* Cascade you can immediately start using it, unless you need to adjust the addresses it listens on or need to modify the settings relating to daemonization.

Important

Fully automatic key rolls are enabled by default. For this to work, Cascade requires access to all nameservers of the zone and the parent zone. If this is not available, make sure to *disable automatic key rolls*.

1.6.1 Configuring Cascade

By default, Cascade only listens on the localhost address. If you want Cascade to listen on other addresses too, you need to configure them.

The `/etc/cascade/config.toml` file controls listen addresses, which filesystem paths Cascade uses, daemonization settings (running in the background, running as a different user), and log settings.

If using systemd to run Cascade some of these settings should be ignored and systemd features used instead.

Using systemd

Without systemd

On systems using systemd the `cascaded.socket` unit is used to bind to listen addresses on behalf of Cascade. By default, the provided listen address is `localhost:53`. If you wish to change the addresses bound, you will need to override the `cascaded.socket` unit. One way to do this is to use the `systemctl edit` command like so:

```
sudo systemctl edit cascaded.socket
```

and insert the following config:

```
[Socket]
# Uncomment the next line if you wish to disable listening on localhost.
#ListenStream=
ListenDatagram=<your-ip>:53
ListenStream=<your-ip>:53
```

Then notify systemd of the changes and (re)start Cascade:

```
sudo systemctl daemon-reload
sudo systemctl restart cascaded
```

When using Cascade without systemd, you need to configure the listen address in the `[server]` section of Cascade's `config.toml`:

```
[server]
servers = ["<your-ip>:53"]
```

Then you can start Cascade with (replace the config and state path with your appropriate values, and if your config uses privileged ports or the daemonization identity feature run the command as root):

```
cascaded --config /etc/cascade/config.toml --state /var/lib/cascade/state.db
```

1.6.2 Interacting with Cascade

Cascade consists of two parts: the **cascaded** daemon which runs continuously, receiving, signing and publishing zone records, and the **cascade** CLI (command-line interface) tool which can be used to inspect and control Cascade.

Using the CLI we can see that on first start Cascade has no policies and no zones:

```
$ cascade status
Signing queue:
  The signing queue is currently empty.

$ cascade policy list

$ cascade zone list
```

Note

The program:*cascade* CLI connects via HTTPS to the **cascaded** daemon. By default it connects to 127.0.0.1:4539. You can override this by passing `--server <IP>:<PORT>` to connect to a Cascade daemon running on another machine.

The **cascade** CLI is the primary means of interacting with the **cascaded** daemon.

For monitoring purposes Cascade supports Prometheus which when combined with other tools such as Grafana and Alertmanager enable visual insight into the behaviour of Cascade and early warning of unexpected situations.

Additionally, while normally not needed, the CLI and the daemon produce logs which can be inspected and if needed can be made more verbose. The CLI logs to the terminal while the daemon typically logs to syslog or to a file. Both the CLI and the daemon take a `--log-level` argument which can be used to adjust the verbosity of the produced log output. It is also possible to use the CLI to adjust the verbosity of an already running daemon, for example:

```
$ cascade debug change-logging --level debug
Changed log-level to: debug
```

1.6.3 Defining Policy

After configuring Cascade, you can begin adding zones. Cascade supports zones sourced from a local file or fetched from another nameserver using XFR *zone transfers*.

Zones take a lot of their settings from policy. Policies allow easy re-use of settings across multiple zones and control things like whether or not zones should be reviewed and how, what DNSSEC settings should be used to sign the zone, and more.

Adding a policy is done by creating a file. To make it easy to get started we provide a default policy template so we'll use that to create a policy for our zone to use. The name of the policy is taken from the file name. The directory to save the policy file to is determined by the *policy-dir* setting as configured in `/etc/cascade/config.toml`.

In the example below, the **`sudo tee`** command is needed because the default policy directory is not writable by the current user.

> **Tip**
>
> Cascade needs to running before you proceed further. See *Configuring Cascade* above on how to configure and start Cascade.

```
cascade template policy | sudo tee /etc/cascade/policies/default.toml
cascade policy reload
```

1.6.4 Signing Your First Zone

Adding a zone will trigger Cascade to load, sign and publish it. If you have configured *Review Hooks*, they will be executed and may intentionally prevent your zone reaching publication.

To add a zone use:

```
cascade zone add --source <file-path|ip-address> --policy default <zone-name>
```

Cascade will now generate signing keys for the zone and attempt to load and sign it.

1.6.5 Checking the Result

You can view the status of a zone with:

```
cascade zone status <zone-name>
```

For example:

```
zone:   example.com
policy: default
source: /path/to/zonefile/example.txt

review
  loaded: off
  signed: off

last published
  loaded serial: 2001062501
  signed serial: 2026050600
  timestamp:     2026-06-02T12:53:10.158779414Z
  size:          10 records

status: idle

Published zone available at 127.0.0.1:4542
```

From the above you can see that the signed zone can be retrieved from `127.0.0.1:4542` using a DNS client, e.g.:

```
dig @127.0.0.1 -p 4542 AXFR example.com
```

If you have the BIND dnssec-verify tool installed, you can check that the zone is correctly DNSSEC signed:

```
$ dig @127.0.0.1 -p 4542 example.com AXFR | dnssec-verify -o example.com /dev/stdin
Loading zone 'example.com' from file '/dev/stdin'

Verifying the zone using the following algorithms:
- ECDSAP256SHA256
Zone fully signed:
Algorithm: ECDSAP256SHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                            ZSKs: 1 active, 0 stand-by, 0 revoked
```

1.6.6 Next Steps

- Establishing the chain of trust to the parent.
- *Automating pre-publication checks*.
- *Using a Hardware Security Module*.
- Migrating an existing DNSSEC signed zone.
- Getting support.

1.7 Key Management

Note

The key management strategy is fundamentally different than the the implementation in OpenDNSSEC and for example BIND.

The goal is that key rolls should always go through the same sequence of steps. As much as possible, we strive to get all key rolls in a single mold. They will always follow the same pattern, while the details of a *KSK* and *ZSK* roll will be different.

1.7.1 Implementation

The key manager is responsible for two things:

1. For each zone, it maintains and provides key rolls for a set of keys that are used to sign.
2. Signing DNSKEY, CDS, and CDNSKEY *RRsets*.

Cascade uses an external key manager, which is part of our **`dnst`** toolset. The actual key management is provided by the `keyset` subcommand of **`dnst`**.

The reason for having an external key manager is to have the flexibility to use different ones. The current key manager requires that keys are online, either in files or in a *Hardware Security Module (HSM)* and does not (explicitly) support a multi-signer setup. We envision future key managers that support offline keys or multi-signing. Finally, a separate key manager makes it relatively easy for Cascade to support high-availability setups, though it does not do that at the moment.

1.7.2 Operations

User interaction with the key manager is designed to be done through Cascade. The key manager manages a set of DNSSEC (**RFC 9364**) signing keys and generates a signed DNSKEY RRset. The key manager expects a separate signer, in this case Cascade, to use the zone signing keys in the key set, sign the zone and include the DNSKEY RRset (as well as the CDS and CDNSKEY RRsets). The key manager supports keys stored in files and keys stored in an HSM.

The key manager operates on one zone at a time. For each zone, the key manager has configuration parameters for key generation (which algorithm to use, whether to use a *CSK* or a *KSK* and *ZSK* pair), parameters for key rolls (whether key rolls are automatic or not), the lifetimes of keys and signatures, etc.

Maintaining State

The key manager maintains a state file for each zone. The state file lists the keys in the key set, the current key roll state, and has the DNSKEY, CDS, and CDNSKEY RRsets, key generation (which algorithm to use, whether to use a CSK or a KSK and a ZSK), parameters for key rolls (whether key rolls are automatic or not), the lifetimes of keys and signatures, etc.

In addition to the configuration and state files, the key manager maintains files for keys that are stored in the filesystem.

Updating Keys

The signatures of the DNSKEY, CDS and CDNSKEY RRsets need to updated periodically. In addition, key roll automation requires periodic invocation of the key manager to start new key rolls and to make progress on ones that are currently executing. For this purpose, Cascade invokes the key manager periodically.

1.7.3 New Zones and Keys

When a new zone is added to Cascade, Cascade will invoke the key manager to create empty key state for the new zone. When adding a zone it is possible to either let the key manager generate new keys or import keys from an existing signer.

When the key manager creates new keys, it will start an algorithm roll instead of using the new keys directly. The reason for this is that the new zone may be an existing unsigned zone that now needs to become a DNSSEC signed zone. The algorithm roll makes sure that the DNSKEY RRset and the zone signatures have propagated before adding the DS record at the parent.

1.7.4 Key Rolls

The key manager can perform KSK, ZSK, CSK, and algorithm rolls.

A KSK roll replaces one KSK with a new KSK. Similarly, a ZSK roll replaces one ZSK with a new ZSK. A CSK roll also replaces a CSK with a new CSK but the roll also treats a pair of KSK and ZSK keys as equivalent to a CSK. So, a CSK roll can also roll from KSK plus ZSK to a new CSK or from a CSK to new a KSK and ZSK pair. Note that a roll from KSK plus ZSK to a new KSK plus ZSK pair is also supported.

Finally, an algorithm roll is similar to a CSK roll, but it is designed in a specific way to handle the case where the new key or keys have an algorithm that is different from one used by the current signing keys.

The KSK and ZSK rolls are completely independent and can run in parallel. Consistency checks are performed at the start of a key roll. For example, a KSK key roll cannot start when another KSK roll is in progress or when a CSK or algorithm roll is in progress. A KSK roll cannot start either when the current signing key is a CSK or when the configuration specifies that the new signing key has to be a CSK.

Finally, KSK rolls are also prevented when the algorithm for new keys is different from the one used by the current key. Similar limitations apply to the other roll types. Note however that an algorithm roll can be started even when it is not needed.

Automatic Key Rolls

For automatic key rolls, the key manager will check the propagation of changes to the DNSKEY RRset, the DS RRset at the parent and the zone's signatures to all nameservers of the zone or the parent zone. To be able to do this, the key manager needs network access to those nameservers.

Important

If Cascade cannot access the nameservers of the zone and parent zone, automation will fail. In that case it is best to *disable (part of) this functionality*.

To check the signatures in the zone, the key manager will issue an AXFR request to the primary nameserver listed in the SOA record of the zone or to a configurable nameserver with configurable TSIG keys for authentication.

The automatic key roll checks have two limitations:

1. They do not work in a multi-signer setup where signers use different keys to sign the zone.
2. Propagation cannot be checked in an any-cast setup. The key manager may continue with the key roll before all nodes in the any-cast cluster have received the new version of the zone.

Future Development

Tip

We explicitly solicit *your input* on how to improve this feature.

We would like to avoid time-based solutions, because that could mean that the key roll will continue even if propagation is not complete. Solutions we are thinking about are a measurement program at the edge of the operator's network that reports back to the key manager about the state of propagation. For propagation in an any-cast cluster, a system such as RIPE Atlas could be used to check propagation across the Internet.

Key Roll Steps

A key roll consists of six steps:

1. *start-roll*
2. *propagation1-complete*
3. *cache-expired1*
4. *propagation2-complete*
5. *cache-expired2*
6. *roll-done*

For each key roll these six steps follow in the same order. Associated with each step is a (possibly empty) list of actions, which fall in three categories:

1. Actions that require updating the zone or the parent zone.
2. Actions that require checking if changes have propagated to all nameservers and require reporting of the TTLs of the changed RRset as seen at the nameservers.
3. Waiting for changes to propagate to all nameservers but there is no need to report the TTL.

Typically, in a list of actions, an action of the first category is paired with one from the second of third category. For example, `UpdateDnskeyRrset` is paired with either `ReportDnskeyPropagated` or `WaitDnskeyPropagated`.

A key roll starts with the *start-roll* step, which creates new keys.

The next step, *propagation1-complete*, has a TTL argument which is the maximum of the TTLs of the Report actions. The *cache-expired1* and *cache-expired2* have no associated actions. They simply require waiting for

the TTL (in seconds) reported by the preceding *propagation1-complete* or *propagation2-complete* steps. The *propagation2-complete* step is similar to the *propagation1-complete* step.

Finally, the *roll-done* step typically has associated Wait actions. These actions are cleanup actions and are harmless but confusing if they are skipped.

Controlling Automation

The key manager provides fine grained control over automation, which can be configured separately for each of the four roll types: KSK, ZSK, CSK and algorithm. For each roll type, there are four booleans: *start*, *report*, *expire* and *done*.

These booleans make it possible to automate KSK or algorithm rolls without starting them automatically. You can also let a key roll progress automatically except for doing the `cache-expired` steps manually, in order to be able to insert extra manual steps.

Important

The *report* and *done* automations require that Cascade has network access to all nameservers of the zone and all nameservers of the parent. If network access is unavailable, make sure to disable them.

Start

When set, the *ksk|zsk|csk|algorithm.auto-start* booleans direct the key manager to start a key roll when a relevant key has expired.

A KSK or a ZSK key roll can start automatically if the respective KSK or ZSK has expired. A CSK roll can start automatically when a CSK has expired, but also when a KSK or ZSK has expired and the new key will be a CSK. Finally, an algorithm roll can start automatically when the new algorithm is different from the one used by the existing keys and any key has expired.

Report

The *ksk|zsk|csk|algorithm.auto-report* options control the automation of the *propagation1-complete* and *propagation2-complete* steps. When enabled, the `cron` subcommand contacts the nameservers of the zone or (in the case of `ReportDsPropagated`, the nameservers of the parent zone) to check if changes have propagated to all nameservers.

The check obtains the list of nameservers from the apex of the (parent) zone and collects all IPv4 and IPv6 addresses. For the `ReportDnskeyPropagated` and `ReportDsPropagated` actions, each address is the queried to see if the DNSKEY RRset or DS RRset match the KSKs. The `ReportRrsigPropagated` action is more complex.

First, the entire zone is transferred from the primary nameserver listed in the SOA record. Then all relevant signatures are checked if they have the expected key tags. The maximum TTL in the zone is recorded to be reported. Finally, all addresses of listed nameservers are checked to see if they have a SOA serial that is greater than or equal to the one that was checked.

Expire

Automation of *cache-expired1* and *cache-expired2* is controlled by the *ksk|zsk|csk|algorithm.auto-expire* policy options. When enabled, the cron subcommand simply checks if enough time has passed to invoke *cache-expired1* or *cache-expired2*.

Done

Finally, the *ksk|zsk|csk|algorithm.auto-done* booleans enable automation of the *roll-done* step. This automation is very similar to the *report* automation. The only difference is that the Wait actions are automated so propagation is tracked but no TTL is reported.

1.7.5 Importing Keys

The key manager supports importing existing keys. Both standalone public keys as well as public/private key pairs can be imported. A standalone public key can only be imported from a file whereas public/private key pairs can be either files or references to keys stored in an HSM.

> **Note**
>
> The public and private key either need to be both files or both stored in an HSM.

There are three basic ways to import existing keys:

1. A public-key stored in a file
2. A public/private key pair stored in files
3. A public/private key pair stored on an HSM

Public Key in a File

A public key can only be imported from a file. When the key is imported the name of the file is converted to a URL and stored in the key set and the key will be included in the DNSKEY RRset. This is useful for certain migrations and to manually implement a multi-signer DNSSEC signing setup. Note that automation does not work for this case.

Public/Private Key Pair in Files

A public/private key pair can be imported from files. It is sufficient to give the name of the file that holds the public key if the filename ends in `.key` and the filename of the private key is the same except that it ends in `.private`. If this is not the case then the private key filename must be specified separately.

Public/Private Key Pair in an HSM

Importing a public/private key stored in an HSM requires specifying the KMIP server ID, the ID of the public key, the ID of the private key, the DNSSEC algorithm of the key and the flags (typically 256 for a ZSK and 257 for a KSK).

Ownership

Normally, the key manager assumes ownership of any keys it holds. This means that when a key is deleted from the key set, the key manager will also delete the files that hold the public and private keys or delete the keys from the HSM that was used to create them.

For an imported public/private key pair this is considered too dangerous because another signer may need the keys. For this reason keys are imported in so-called `decoupled` state. When a decoupled key is deleted, only the reference to the key is deleted from the key set, the underlying keys are left untouched. There is a `--coupled` option to tell keyset to take ownership of the key.

1.8 Hardware Security Modules (HSMs)

> **Note**
>
> Cascade does not *require* a *Hardware Security Module (HSM)* to operate. While it is common practice to secure cryptographic key material using an HSM, not all operators use an HSM. Cascade is able to use OpenSSL and/or ring software cryptography to generate signing keys and to cryptographically sign DNS *RRset* data, storing the generated keys in on-disk files.

1.8.1 An Introduction to HSMs

A Hardware Security Module is typically a tamper proof hardware vault capable of generating and securely storing cryptographic keys and performing signing operations using those keys on data provided via an interface and returning the signed result via the same interface.

HSM Interfaces

In most cases, you interact with an HSM via an interface that is compliant with the Oasis PKCS#11 (Public-Key Cryptography Standard) specification. Some HSMs also or alternatively support a newer Oasis specification called KMIP (Key Management Interoperability Protocol).

KMIP is a data (de)serialization protocol that operates on top of the widely used TCP and TLS combination of protocols. As such, it requires no additional software or special configuration to use and poses no direct security or stability threat to the client process.

This is quite different to PKCS#11, which requires the HSM vendor to provide a library of code that offers a C language style interface to be used by the client at runtime by loading the library (a.k.a. module) into its own process with no knowledge of or control over what that code is going to do.

1.8.2 Cascade and HSMs

Cascade supports both PKCS#11 and KMIP compatible HSMs. KMIP is supported natively, while PKCS#11 is supported through our **`cascade-hsm-bridge`**.

Cascade is an application written in Rust. Crossing the divide between the Rust host application and a loaded C library means giving up the stability and memory safety guarantees offered by Rust. As such, Cascade was designed to *not* load PKCS#11 modules directly but instead to hand that risk off to a helper tool: **`cascade-hsm-bridge`**.

To interact with a HSM over its PKCS#11 interface, Cascade sends KMIP requests to **`cascade-hsm-bridge`**, which executes them against a loaded PKCS#11 vendor library.

Supported HSMs

In principle any HSM supporting PKCS#11 v2.40 or KMIP 1.2 should be supported. To work with an HSM using its PKCS#11 interface, Cascade requires our **`cascade-hsm-bridge`**.

Several HSMs have been tested with Cascade. Our testing was limited to normal usage only, not attempting to deliberately cause problems, and not attempting to stress or performance test the interface. The tested HSMs are:

Table 1: Supported HSMs

HSM	Type	Interface	Integration guide
Fortanix DSM	Cloud	KMIP	
Thales Cloud HSM	Cloud	PKCS#11	*view*
Nitrokey NetHSM[1]	Docker image	PKCS#11	*view*
YubiHSM 2	USB key	PKCS#11	
SoftHSM v2.6.1	Software	PKCS#11	*view*
SmartCard-HSM	Smart Card	PKCS#11	*view*

Note

Cascade requires TLS 1.3 for connections to the KMIP server, even though KMIP 1.2 requires servers to offer support for old versions of the TLS protocol with known security vulnerabilities. For this reason, PyKMIP is **NOT** supported by Cascade as it only supports older vulnerable TLS versions.

Setting up cascade-hsm-bridge

If you installed Cascade via a DEB or RPM package you should also already have the **cascade-hsm-bridge** software installed, unless you explicitly opted not to install it. You can also *build the software* from source.

See also

We provide man pages for both the cascade-hsm-bridge daemon and configuration file.

When installed via a package the daemon will not be run automatically. This is because you will need to:

- Edit the cascade-hsm-bridge configuration file to set the location of your PKCS#11 module.
- Depending on your PKCS#11 module, you may need to set vendor specific environment variables for the **cascade-hsm-bridge** process. You may also need to ensure that vendor specific configuration files and possibly other software is installed and correctly configured.
- Ensure that the **cascade-hsm-bridge** user has access to the resources needed by the PKCS#11 module to be loaded.
- Use the (vendor specific) PKCS#11 module setup process to create a token label and PIN that Cascade should use to authenticate with the HSM.
- Optionally, generate a proper TLS certificate for use by **cascade-hsm-bridge** and set the `cascade-hsm-bridge:cert_path` and `cascade-hsm-bridge:key_path` in `/etc/cascade-hsm-bridge/config.toml` to point the certificate file and accompanying private key. If you omit these settings, **cascade-hsm-bridge** will generate a long-lived self-signed TLS certificate each time it starts.

Note

There is currently no way to test that the cascade-hsm-bridge configuration is correct other than trying to use it with Cascade.

When ready, start **cascade-hsm-bridge** either via systemd (if installed from a package) or directly:

[1] Works with v1.7.2 of their PKCS#11 module. v2.0.0 and above are NOT currently supported due to a known bug in cascade-hsm-bridge.

```
cascade-hsm-bridge --config /etc/cascade-hsm-bridge/config.toml -d --user <USER> --group
↪<GROUP>
```

> **Tip**
>
> Use the `cascade-hsm-bridge:--user` and `cascade-hsm-bridge:--group` arguments to make **`cascade-hsm-bridge`** run as the same user that has access to any necessary resources required by PKCS#11 module vendor.

Using cascade-hsm-bridge with Cascade

To use **`cascade-hsm-bridge`** with Cascade we must tell it that there is a HSM running that it can connect to. In the instructions below the PKCS#11 token label and PIN are the values you configured above.

```
cascade hsm add --insecure --username <PKCS#11 token label> --password <PKCS#11 PIN>␣
↪cascade-hsm-bridge 127.0.0.1
```

> **Note**
>
> The *`--insecure`* option must be used if using a self-signed TLS certificate, which is the default. 127.0.0.1 should be changed if your **`cascade-hsm-bridge`** instance is running on a different address.

Cascade will verify that it can connect and that the target server appears to be a KMIP compatible HSM.

> **Note**
>
> Cascade does **not** yet verify that the target KMIP server supports the operations needed by Cascade. For **`cascade-hsm-bridge`** this isn't a problem as it is designed to work with Cascade.

Next, we need to add the HSM to a policy so that when zones are added the keys for the zones will be generated using the HSM.

To do this, edit `/etc/cascade/policies/<your_policy>.toml` and set:

```
[key-manager.generation]
hsm-server-id = "cascade-hsm-bridge"
```

Now when you use `cascade zone add --policy <your_policy>` the HSM will be used for key generation and signing.

1.9 Review Hooks

Cascade offers two review hook points in its signing pipeline for automated zone validation by user provided review scripts. These review hooks can be used to perform any validation the user requires to ensure their zone is correct at all stages, using any (third-party) tools desired.

A review script in Cascade is a custom program created by the user and configured in the zone's policy, as shown below in *Review with Script*. It performs the desired validation and signals approval or rejection to Cascade via the program's exit code. An exit code of 0 means the zone is approved, and any other exit code means the zone is rejected.

The first review hook is available after a zone is loaded by Cascade before it is signed. The second review hook is available after the zone is signed and not yet published. The review script can approve or reject a zone at either of these stages.

If a review script approves the zone, then that version of the zone will continue through the pipeline as usual. If a review script rejects the zone instead, then there are two things that can happen, depending on the configuration. Either the new version will be discarded and Cascade will simply wait for a new version of the zone, or Cascade will halt and stop doing new operations to the zone until an operator resolves the issue.

A review script receives relevant information about the zone in environment variables listed in the *policy file manual*. The review script then needs to use the address provided in the environment variables to fetch the zone via AXFR *zone transfer* and perform the required checks.

Note

When Cascade is run as a daemon with the provided systemd service unit, its file system access is limited. It only has write access to the `/var/lib/cascade` directory and a private `/tmp` directory. If a review script needs to write some state to a file, for example to store zonefiles of previously reviewed zones, it should do so in `/var/lib/cascade`.

If you need to write to other locations, they can be allowed with the `ReadWritePaths` option in the unit file. See the `systemd.exec` manpage for more information.

1.9.1 Review with Script

To configure a script as review hook, you set the review *`mode = "script"`* policy option, and specify the review script using the *`hook`* option in the *[loader.review]* and/or *[signer.review]* policy file sections.

Review scripts (or programs) are called using `sh -c`, so you can provide arguments to your review script, e.g.: *`hook = "script.sh --stage unsigned"`*.

Here is an example configuration for automatic review:

```
[loader.review]
mode = "script"
hook = "review_loaded.sh"

[signer.review]
mode = "script"
hook = "review_signed.sh"
```

1.9.2 Manual Review

You can also enable manual review by setting the review *`mode = "manual"`* option under *[loader.review]* or *[signer.review]*.

If no hook command is provided, but review is required, manual approval or rejection has to be performed using the CLI commands **`cascade zone approve`** or **`cascade zone reject`**.

Here is an example configuration for manual review:

```
[loader.review]
mode = "manual"

[signer.review]
mode = "manual"
```

1.9.3 Decide What Happens on Rejection

Cascade can do two things when a zone instance is rejected: halt or discard. This is configured through the `on-reject` field under *[loader.review]* or *[signer.review]*.

If it is configured to `"discard"`, it will simply go back to the idle state as if the loading or signing operation didn't happen. This is the most fault-tolerant option.

If it is instead set to `"halt"` then **Cascade will stop doing any operations** to the zone. This allows the operator to investigate the issue before Cascade continues. If the zone should be accepted anyway, the **`cascade zone override`** command can be used to override the previous rejection. If the rejection was correct, the **`cascade zone reset`** command can be used to discard the loaded instance.

```
[loader.review]
mode = "script"
hook = "review_loaded.sh"
on-reject = "halt"
```

1.9.4 Example

In this example we will use validns to validate the unsigned zone, and dnssec-verify to validate the signed zone.

To do this, we need to write a shell script that fetches the zone using AXFR and performs the relevant checks. Let's save the following review script[1] as `/usr/local/bin/cascade-review.sh`:

```
#!/usr/bin/env sh

set -e

logger -p daemon.notice -t cascade "Validating ${CASCADE_ZONE} of serial ${CASCADE_
↪SERIAL} from ${CASCADE_SERVER}"

# Using `validns` to check the unsigned zone
# and `dnssec-verify` to check the signed zone
# Unfortunately, dig logs some errors on standard output... Nothing to do there
dig @${CASCADE_SERVER_IP} -p ${CASCADE_SERVER_PORT} "${CASCADE_ZONE}" AXFR | \
    if [ "$1" = "unsigned" ]; then
        # validns does not handle Ed25519
        validns -z "${CASCADE_ZONE}" -p all -
    else
        dnssec-verify -q -o "${CASCADE_ZONE}" /dev/stdin
    fi
```

Next, we update the zone's policy to use the review script for both stages:

```
# Keep the other settings in the policy as is ...

[loader.review]
mode = "script"
hook = "/usr/local/bin/cascade-review.sh unsigned"

[signer.review]
mode = "script"
hook = "/usr/local/bin/cascade-review.sh"
```

[1] Original review script example by Stéphane Bortzmeyer on GitHub

Finally, we need to reload the policy with **cascade policy reload** to apply the policy changes.

1.10 Zone Transfers

Cascade is expected to be deployed between a hidden upstream nameserver and public downstream nameservers. The hidden upstream serves the unsigned zone, Cascade signs it, and serves the signed zone to downstream nameservers for publication to consumers.

Communication of changed zone records from upstream to downstream will be done via the network using the **RFC 5936** (AXFR) and **RFC 1995** (IXFR) protocols.

Authentication of transferring parties can be done using **RFC 8945** (TSIG) keys, using a shared secret communicated out of band to the nameservers sending and receiving the zone records.

Cascade supports timely discovery of zone changes by sending SOA queries to the upstream nameserver, either in response to an **RFC 1996** NOTIFY message or based on the zone's SOA timers.

> **Note**
>
> Cascade also supports loading the zone from a file. However, if only a small fraction of the records in the zone change from one version to the next, loading the entire file every time the zone file changes will require more time, CPU and memory compared to processing only the differences when using IXFR. Cascade doesn't yet support direct writing of signed zones to a file, though a signed zone review hook could be used to AXFR the signed zone to a file on disk to achieve this.

1.10.1 Using zone transfers with an upstream nameserver

To instruct Cascade to transfer a zone via the network instead of loading it from a file you must supply an upstream nameserver IP address when adding the zone. See **cascade** `zone add`, and optionally a TSIG key to use to authenticate communication.

Cascade will then attempt to fetch the zone. Where possible it will fetch newer versions of the zone incrementally, as this is more efficient.

Cascade can be instructed to authenticate the upstream nameserver by use of a TSIG key. The TSIG key to use must be provided to Cascade _before_ adding the zone. See **cascade** `tsig add`.

1.10.2 Providing zone transfers to a downstream server

By default, Cascade allows downstream servers to access published zones by zone transfer, no configuration is needed.

To ensure timely update by secondaries, Cascade can be configured to send **RFC 1996** NOTIFY messages to specified secondaries. This is done via the policy setting `server.outbound.send-notify-to`, optionally specifying an :RFC:8945` TSIG key to use to authenticate communication.

> **Tip**
>
> Remember to reload the policy file after changing it. See **cascade** `policy reload`.

> **Tip**
>
> Use **cascade** `tsig add` to add a TSIG key to Cascade _before_ reloading policy file changes.

1.10.3 Controlling automatic key rollover zone transfer settings

When using automatic key rollover (the default) Cascade will attempt to verify that certain key properties of the signed zone being served to consumers are correct.

This verification is done by transferring the zone and inspecting it. By default transfer is attempted from the nameserver identified by the MNAME field of the apex SOA record in the zone.

If an alternate nameserver should be queried instead of the MNAME nameserver, or if a specific port number or TSIG key should be used to request the transfer, you will also need to configure the Cascade key manager to fetch the zone correctly. This can be done via the `key-manager.publication-nameservers` policy setting.

1.11 Maintenance Mode

By default, Cascade will automatically reload and resign the zone when necessary, but sometimes you might need a bit more control over that process. That is what "maintenance mode" is for. It blocks any operations on the zone, allowing the operator to inspect what's going wrong.

Maintenance mode works per zone, allowing you to take control over a particular zone while the rest of the zones will be updated normally.

> **Note**
>
> Maintenance mode is a work in progress. We plan on allowing triggering certain operations manually via the CLI while maintenance mode is enabled in the future.

1.11.1 Toggling Maintenance Mode

Maintenance mode for a zone can be enabled from the CLI with the following command:

```
cascade zone maintenance enable <zone-name>
```

It can similarly be disabled with:

```
cascade zone maintenance disabled <zone-name>
```

You can check whether a zone is in maintenance mode by looking at the `zone status` of the zone in question. If it is in maintenance mode, then a warning will be shown at the bottom:

```
WARNING: This zone is in maintenance mode
  Cascade will not automatically start new loading and signing operations
  Run `cascade zone maintenance disable <zone-name>` to resume normal operation
```

1.12 Integrating with NSD

> Name Server Daemon (NSD) by NLnet Labs is an authoritative DNS name server.
>
> —https://nsd.docs.nlnetlabs.nl/

1.12.1 Suggested reading

The Zone Transfers page explains the general functionality in Cascade that is referred to below.

1.12.2 Using NSD as a primary to Cascade

To use NSD as an upstream name server of Cascade you must add a zone to NSD that refers to Cascade as a secondary name server. If enabled in NSD, NSD will send an **RFC 1996** DNS NOTIFY message to Cascade notifying it when changes to the zone occur.

The NOTIFY message will trigger Cascade to perform an AXFR transfer to fetch the full zone content from NSD, or, if already fetched and IXFR is enabled in NSD, an IXFR transfer will be performed to fetch just the incremental changes since the last fetch.

If NOTIFY is NOT enabled in NSD, Cascade will monitor NSD for a newer version of the zone by periodically sending SOA queries according to the number of seconds defined by the REFRESH field of the zone apex SOA record.

Optionally NSD and Cascade can be configured with the same TSIG key to authenticate the NOTIFY and XFR messages.

The NSD settings relevant here are:

- notify
- provide-xfr

For example the following NSD configuration file fragment adds an `example.com` zone to NSD that is to be served as input to a Cascade daemon running on host 192.168.0.2 listening on the default port 4542:

```
zone:
  name: example.com
      zonefile: /etc/nsd/example.com.zone
      notify: 192.168.0.2@4542 NOKEY
      provide-xfr: 192.168.0.2 NOKEY
      store-ixfr: yes
      create-ixfr: yes
```

A TSIG key can be used to authenticate the NOTIFY and XFR communications. For example:

```
key:
  name: "sec1_key"
  algorithm: hmac-sha256
  secret: "..."

 zone:
   name: example.com
       zonefile: /etc/nsd/example.com.zone
       notify: 192.168.0.2@4542 sec1_key
       provide-xfr: 192.168.0.2 sec1_key
       store-ixfr: yes
       create-ixfr: yes
```

See https://nsd.docs.nlnetlabs.nl/en/latest/running/using-tsig.html for more information.

> **Tip**
>
> Remember to reload the NSD configuration or restart NSD so that changes to the configuration take effect.

To add the TSIG key to Cascade use **`cascade`** `tsig add`:

```
$ cascade tsig add --name sec1_key --alg hmac-sha256 --secret "...=="
```

To use the new TSIG key it must be specified when adding a zone to Cascade. Assuming that NSD is running on host 192.168.0.1 on port 53, the following command instructs Cascade to add the `example.com` zone sourced from the NSD server using the `sec1_key` TSIG key to authenticate with NSD:

```
$ cascade zone add --source "192.168.0.1^sec1_key" --policy default example.com
```

1.12.3 Using NSD as a secondary to Cascade

The NSD settings relevant here are:

- allow-notify
- request-xfr

To allow NSD to receive and respond to NOTIFY messages from Cascade, and to instruct NSD to fetch zone updates via zone transfer from Cascade, these settings need to be added to the zone configuration.

For example the following configures NSD for an upstream Cascade server running at 192.168.0.2 listening on the default port 4542:

```
zone:
  name: example.com
      allow-notify: 192.168.0.2
      request-xfr: 192.168.0.2@4542
```

To authenticate zone transfer related messages using TSIG you must first have added the key to both Cascade and NSD using the same **`cascade`** `tsig add` command and `key:` block as shown above.

To instruct Cascade to use TSIG when communicating with a downstream nameserver such as NSD you must set the required policy settings and reload the policy. The relevant policy settings in Cascade are:

- `server.outbound.send-notify-to`
- `server.outbound.provide-xfr-to`

1.13 Integrating with SoftHSMv2

> **Note**
>
> The instructions on this page are for an Ubuntu 24.04 host and assume that Cascade has already been installed using our DEB package.

> SoftHSM is an implementation of a cryptographic store accessible through a PKCS#11 interface. You can use it to explore PKCS#11 without having a Hardware Security Module. It was originally developed as a part of the OpenDNSSEC project, but is now a standalone project. SoftHSM uses Botan or OpenSSL for its cryptographic operations.
>
> —https://www.softhsm.org/

1.13.1 Install SoftHSMv2

```
# apt install -y softhsm2
# softhsm2-util --init-token --label Cascade --pin 1234 --so-pin 1234 --free
```

1.13.2 Configure `cascade-hsm-bridge`

`cascade-hsm-bridge` needs to know where to find the SoftHSMv2 PKCS#11 module. As PKCS#11 modules are loaded into a host application, any access to resources needed by the PKCS#11 module must be granted to the host application. For SoftHSM that involves granting the user that **`cascade-hsm-bridge`** runs as read-write access to the */var/lib/softhsm* directory. Let's set this all up and start the daemon:

```
# sed -i -e 's|^lib_path = .\+|lib_path = "/usr/lib/softhsm/libsofthsm2.so"|' /etc/
↪cascade-hsm-bridge/config.toml
# chown -R cascade-hsm-bridge: /var/lib/softhsm
# systemctl start cascade-hsm-bridge
```

1.13.3 Create a Cascade Policy that uses SoftHSM

Create a Cascade policy called `softhsm` and set it to use a HSM called `cascade-hsm-bridge`.

```
# cascade template policy | tee /etc/cascade/policies/softhsm.toml
# sed -i -e 's|^#hsm-server-id = .\+|hsm-server-id = "cascade-hsm-bridge"|' /etc/cascade/
↪policies/softhsm.toml
```

Start the Cascade daemon:

```
# systemctl start cascaded
# cascade policy reload
```

Configure a HSM in Cascade called `cascade-hsm-bridge` that will connect to the locally running **`cascade-hsm-bridge`** daemon:

```
# cascade hsm add --insecure --username Cascade --password 1234 cascade-hsm-bridge 127.0.
↪0.1
Added KMIP server 'cascade-hsm-bridge 0.1.0-alpha using PKCS#11 token with label Cascade␣
↪in slot SoftHSM slot ID 0x1948bafd via library libsofthsm2.so'.
```

1.13.4 Sign a Test Zone with SoftHSM

Create a test zone to load and sign and ensure the Cascade daemon has access to it:

```
# mkdir /etc/cascade/zones
# cat > /etc/cascade/zones/example.com << EOF
example.com.    3600    IN      SOA     ns.example.com. username.example.com. 1 86400␣
↪7200 2419200 300
example.com.            IN      NS      ns
ns                      IN      A       192.0.2.1
EOF
# chown -R cascade: /etc/cascade/zones
```

Add our test zone to Cascade and associate the policy that we created with the zone:

```
# cascade zone add --source /etc/cascade/zones/example.com --policy softhsm example.com
Added zone example.com
```

Check that the zone has been signed, and print out additional information which includes the identifiers of the signing keys that were used:

```
# cascade zone status example.com --detailed
Status report for zone 'example.com' using policy 'softhsm'
✓ Waited for a new version of the example.com zone
✓ Loaded version 1
  Loaded at 2025-10-01T21:44:13+00:00 (1m 46s ago)
  Loaded 196 B from the filesystem in 0 seconds
✓ Auto approving signing of version 1, no checks enabled in policy.
✓ Approval received to sign version 1, signing requested
✓ Signed version 1 as version 2025100101
  Signed at 2025-10-01T21:44:13+00:00 (1m 45s ago)
  Signed 3 records in 0s
✓ Auto approving publication of version 2025100101, no checks enabled in policy.
✓ Published version 2025100101
  Published zone available on 127.0.0.1:4543
DNSSEC keys:
  KSK tagged 16598:
    Reference: kmip://cascade-hsm-bridge/keys/C9623EAF300AF8E4A3DF6D5F6AD6674B49CCD322_
↪pub?algorithm=13&flags=257
    Actively used for signing
  ZSK tagged 50714:
    Reference: kmip://cascade-hsm-bridge/keys/3C95A4EC3A1E26BC67EC0336926ADBB212ADB3D8_
↪pub?algorithm=13&flags=256
    Actively used for signing
...
```

1.13.5 Inspect the SoftHSM Key Store

Install the `pkcs11-tool` program from the `opensc` package and use it to query SoftHSMv2 directly:

```
# apt install -y opensc
# pkcs11-tool --module /usr/lib/softhsm/libsofthsm2.so --token-label Cascade --so-pin␣
↪1234 -O
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:  ␣
↪04410489c96a67a451f26b75d0cbf903211d7d892e36c577a707e144a97309f20f47144a4bb1c5b437ac04fc1a2f44251253f
  EC_PARAMS:  06082a8648ce3d030107 (OID 1.2.840.10045.3.1.7)
  label:      example.com-50714-zsk-pub
  ID:         3c95a4ec3a1e26bc67ec0336926adbb212adb3d8
  Usage:      verify, verifyRecover
  Access:     local
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:  ␣
↪0441041517afa18dcf0eb9aec58de3bd54585e152e634ee332c4d73c587e4fb2ebded9432be24cd4ea34f34290ffbd5f27a1e
  EC_PARAMS:  06082a8648ce3d030107 (OID 1.2.840.10045.3.1.7)
  label:      example.com-16598-ksk-pub
  ID:         c9623eaf300af8e4a3df6d5f6ad6674b49ccd322
  Usage:      verify, verifyRecover
```

(continues on next page)

(continued from previous page)

```
Access:     local
```

Notice that the key IDs stored in SoftHSMv2 match those reported by Cascade.

End.

1.14 Integrating with Thales Cloud HSM

The instructions on this page are for use with the Thales Data Protection on Demand (DPoD) service.

These instructions show how to run **`cascade-hsm-bridge`** (the tool Cascade uses to connect to PKCS#11 compatible *Hardware Security Modules (HSMs)*) inside a Docker container and connect to its listen port from a server or another Docker container.

Docker is **not** required to use Cascade or **`cascade-hsm-bridge`**. This example uses Docker because the Thales documentation describes how you can easily get PKCS#11 connectivity to a Thales Luna Cloud HSM working using Docker.

Warning

DPoD is **not** free. An initial free trial is available but thereafter it is a paid service.

Note

The PKCS#11 specific part of the instructions below is hopefully similar for any modern Thales HSM service whether PCI card, rack unit or cloud service, but we have no access to such HSMs and thus have only been able to test with DPoD.

Tip

When running **`cascade-hsm-bridge`** as a systemd service, i.e. not in a Docker container as this page describes, be aware that you will need to use the `systemctl edit cascade-hsm-bridge` command to set some extra systemd settings that the Thales Luna Cloud HSM PKCS#11 module needs.

```
[Service]
WorkingDirectory=/usr/local/dpodclient/libs/64
Environment="ChrystokiConfigurationPath=/usr/local/dpodclient"
MemoryDenyWriteExecute=no
```

Note

We are not Docker or Thales experts. Consult the Thales and Docker documentation and other authoritative sources to learn more.

1.14.1 Acquire the PKCS#11 Module

The first step to using a Thales HSM with Cascade, assuming that the HSM itself is already provisioned, is to acquire the Thales PKCS#11 module that contains the code needed to connect to a Thales Luna Cloud HSM.

> **Note**
>
> Thales HSMs are commercial products and Thales do not make their software developer kit, of which the PKCS#11 module is part, publically available. One version of it is however available via the free trial of the Thales Data Protection on Demand service which we will demonstrate here.

1. Login to the Thales DPoD portal. This step assumes you already have an account.
2. Via the Service tab add a Luna Cloud HSM service to your account.
3. Enter a name for your service.
4. (optional) Tick the "Remove FIPS restrictions" box. For our test we left the "Remove FIPS restrictions" box unticked.
5. Click the name of your new Luna Cloud HSM service on the Service tab.
6. Click the "New Service Client" button.
7. Give your service client a name when asked.
8. Click the "Create Service Client" button.
9. Click the "Download Client" button that appears.

This will download a ZIP archive called "setup-<YOUR_CLIENT_NAME>.zip.

Inside the zip are the files needed to connect to the Luna Cloud HSM using a PKCS#11 client like **`cascade-hsm-bridge`**, including client certificates to authenticate, a PKCS#11 module configuration file called `Chrystoki.conf`, and a TAR archive containing the PKCS#11 module `libs/64/libCryptoki2.so`.

1.14.2 Testing the PKCS#11 Module

By this point you should in principle have everything needed to connect **`cascade-hsm-bridge`** or any other PKCS#11 client to the Luna Cloud HSM.

However, there are a lot of files in the downloaded service client ZIP and you'll need to work out which ones you need and how to use them.

Thales provide a guide for using a Docker container to Access a Luna Cloud HSM Service. We use that guide here to demonstrate that Cascade works with the Thales Luna Cloud HSM.

> **Tip**
>
> The following Thales documentation pages are particularly relevant in the next steps:
>
> - Create a Docker Container to Access a Luna Cloud HSM Service
> - Initializing an Application Partition
> - Partition Roles
> - Initializing the Crypto Officer Role

Follow the steps below to confirm that you can connect via PKCS#11 to your DPoD Luna Cloud HSM instance.

10. Build a Docker image as described at Create a Docker Container to Access a Luna Cloud HSM Service.

> **Note**
>
> Replace `FROM ubuntu:20.04` in the Docker instructions with `FROM ubuntu:22.04`.`
>
> When following the instructions to build the Docker image, replace references to `setup-myclient.zip` with **YOUR** service client ZIP that you downloaded in step 9 above.

11. Assuming that you have built your Docker image according to the Thales instructions using your downloaded service client ZIP, run a container based on the image and use the Thales `lunacm` command to setup access to your Luna Cloud HSM:

    > **Note**
    >
    > The docker command below has an additional `--publish` argument that is not present in the Thales documentation. This is needed to expose the **`cascade-hsm-bridge`** listen port outside the container so that you can connect to it from Cascade running on the host or inside another container.

    ```
    $ docker run -it \
        --name luna \
        --publish 5696:5696 \
        --entrypoint=./bin/64/lunacm \
        myimage
    lunacm:> partition init -label MyPartition -password mypartitionsopassword -domain␣
    ↪mydomainname
    lunacm:> role login -name po
    lunacm:> role init -name co
    lunacm:> role login -name co
    lunacm:> role changepw -name co
    ```

12. To test our settings before we use **`cascade-hsm-bridge`** we can use the opensc `pkcs11-tool` program *from another shell terminal*:

    ```
    $ docker exec -it luna /bin/bash
    # apt update
    # apt install -y opensc
    # pkcs11-tool --module ./libs/64/libCryptoki2.so -I
    Cryptoki version 2.20
    Manufacturer     SafeNet
    Library          Chrystoki                          (ver 10.9)
    Using slot 3 with a present token (0x3)
    # pkcs11-tool --module ./libs/64/libCryptoki2.so --login -O
    Using slot 3 with a present token (0x3)
    Logging in to "MyPartition".
    Please enter User PIN: <THE PASSWORD YOU CHOSE IN STEP 11 ABOVE>
    ```

Now that that works we can install **`cascade-hsm-bridge`**.

1.14.3 Installing and Configuring `cascade-hsm-bridge`

13. Continuing from the same `/bin/bash` session inside the Docker container, follow the *Installation* steps to install **`cascade-hsm-bridge`** for Ubuntu 24.04, the base image used by our DPoD Docker container.

> **Note**
>
> The installation instructions use `sudo` but this does not usually exist inside a Docker container as typically one executes commands as `root`. Either remove `sudo` from any commands you copy-paste, or execute `alias sudo=` before copy-pasting commands that use `sudo`. This will ensure that the commands work as intended.

14. Next, edit the **cascade-hsm-bridge** configuration file to point it to the Thales Luna Cloud HSM PKCS#11 module, and to listen on all network IPv4 interfaces inside the Docker container:

```
$ sed -i -e 's|^lib_path =.\+|lib_path = "/usr/local/dpodclient/libs/64/
↪libCryptoki2.so"|' /etc/cascade-hsm-bridge/config.toml
$ sed -i -e 's|addr = .\+|addr = "0.0.0.0"|' /etc/cascade-hsm-bridge/config.toml
```

15. Now run **cascade-hsm-bridge** and send its logs to the terminal so that we can easily verify that it loads the Thales PKCS#11 module correctly.

```
$ cascade-hsm-bridge -c /etc/cascade-hsm-bridge/config.toml --stderr
$ cat /tmp/cascade-hsm-bridge.log
[2025-10-03T20:48:37] [INFO] Loading and initializing PKCS#11 library /usr/local/
↪dpodclient/libs/64/libCryptoki2.so
[2025-10-03T20:48:37] [INFO] Loaded SafeNet PKCS#11 library v10.9 supporting␣
↪Cryptoki v2.20: Chrystoki
[2025-10-03T20:48:37] [WARN] Generating self-signed server identity certificate
[2025-10-03T20:48:37] [INFO] Listening on 127.0.0.1:5696`
```

Here we can see that the PKCS#11 module has been loaded correctly.

This does NOT show that **cascade-hsm-bridge** is able to connect to the Luna Cloud HSM, but the `pkcs11-tool -O` command we used above proved that the PKCS#11 module is able to connect and so **cascade-hsm-bridge** can as well. To demonstrate that, however, you will need to setup Cascade to use this running instance of **kmip2pksc11**.

1.14.4 Using cascade-hsm-bridge to connect Cascade to the Thales HSM

To learn how to use the **cascade-hsm-bridge** instance that you just setup with Cascade, visit the *Hardware Security Modules (HSMs)* page, but skip to the *"Using cascade-hsm-bridge with Cascade"* section as we have already setup **cascade-hsm-bridge** on port 5659.

If you have Cascade setup, the command to add the Thales HSM is:

```
$ cascade hsm add thales 127.0.0.1 \
    --insecure \
    --username MyPartition \
    --password <THE PASSWORD YOU CHOSE IN STEP 11 ABOVE>
[2025-10-03T21:43:43.486Z] INFO cascade::units::http_server: Writing to KMIP server file
↪'./kmip/thales
Added KMIP server 'cascade-hsm-bridge 0.1.0-alpha using PKCS#11 token with label␣
↪MyPartition in slot Net Token Slot via library libCryptoki2.so'.
```

Note that the username is the PKCS#11 slot label, and the password is the password you chose in step 11 above when setting up the Luna Cloud HSM.

We can see from the output that Cascade made an initial test connection to the Thales HSM via **cascade-hsm-bridge** to query its identification strings.

For an example of how to associate the HSM with a Cascade policy and use it to sign a zone see the *Integrating with SoftHSMv2* page.

1.15 Integrating with a SmartCard-HSM

Note

The instructions on this page are for an Debian 12 host and assume that Cascade has already been installed using our DEB package.

Note

The instructions on this page assume you will be using the Smart Card exclusively for the task at hand and might wipe the content of the SC.

> SmartCard-HSM is a lightweight Hardware Security Module in a Smart Card, MicroSD or USB form factors and protect your RSA and ECC keys in hardware. They are accessible through a PKCS #11 interface. You can use them to experiment with PKCS #11 without having to purchase an expensive HSM. (Hint: do not expect wonders in terms of performance.)
>
> —https://www.smartcard-hsm.com

1.15.1 Install the Prerequisites

```
# apt install -y opensc opensc-pkcs11
```

1.15.2 Identify the Card Reader

Your card reader may well be of a different vendor and type.

```
# opensc-tool -l
# Detected readers (pcsc)
Nr.  Card  Features  Name
0    Yes             Identive CLOUD 2700 R Smart Card Reader [CCID Interface]
```

1.15.3 Initialize the Smartcard

The card is configured with the SO PIN and user PIN as per the vendor. You can initialize (i.e. fully reset) the card and change these using **sc-hsm-tool** which is provided in the **opensc** package:

```
# sc-hsm-tool --initialize --so-pin 0123012301230123 --pin 123456 --label NL-smartcard
Using reader with a card: Identive CLOUD 2700 R Smart Card Reader [CCID Interface]
```

1.15.4 Show the card's slot information

We display a list of available slots and take note of our slot ID (`0`) to which our card label (`NL-smartcard`) is associated:

```
# pkcs11-tool --list-slots
Available slots:
Slot 0 (0x0): Identive CLOUD 2700 R Smart Card Reader [CCID Interface] (536...
  token label        : NL-smartcard (UserPIN)
  token manufacturer : www.CardContact.de
  token model        : PKCS#15 emulated
  token flags        : login required, rng, token initialized, PIN initialized
  hardware version   : 24.13
  firmware version   : 4.1
  serial num         : DECC1206715
  pin min/max        : 6/15
```

1.15.5 List the Smart Card's mechanisms

```
# pkcs11-tool --module opensc-pkcs11.so --list-mechanisms
Using slot 0 with a present token (0x0)
Supported mechanisms:
  SHA-1, digest
  SHA224, digest
  SHA256, digest
  SHA384, digest
  SHA512, digest
  MD5, digest
  RIPEMD160, digest
  GOSTR3411, digest
  ECDSA, keySize={192,521}, hw, sign, verify, EC F_P, EC parameters, EC OID, EC␣
↪uncompressed
  ECDSA-SHA384, keySize={192,521}, sign, verify
 ...
```

1.15.6 Configure cascade-hsm-bridge

cascade-hsm-bridge needs to know where to find the OpenSC PKCS#11 module. As PKCS#11 modules are loaded into a host application, any access to resources needed by the PKCS#11 module must be granted to the host application.

```
# sed -i -e 's|^lib_path = .\+|lib_path = "/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so"|' /
↪etc/cascade-hsm-bridge/config.toml
# systemctl start cascade-hsm-bridge
```

1.15.7 Create a Cascade Policy that uses your HSM

Create a Cascade policy called `smartcard` and set it to use the HSM called `cascade-hsm-bridge` we configured earlier.

```
# cascade template policy | tee /etc/cascade/policies/smartcard.toml
# sed -i -e 's|^#hsm-server-id = .\+|hsm-server-id = "cascade-hsm-bridge"|' /etc/cascade/
↪policies/smartcard.toml
```

Start the Cascade daemon:

```
# systemctl start cascaded
# cascade policy reload
Policies reloaded:
- smartcard added
```

Configure a HSM in Cascade called `cascade-hsm-bridge` that will connect to the locally running **cascade-hsm-bridge** daemon. The `username` is the slot identifier we found our card in earlier, and the `password` is the user PIN configured for the card. (`username` can also be the full token label which in our above example is the complete string `NL-smartcard (UserPIN)`.)

```
# cascade hsm add --insecure --username 0 --password "123456" cascade-hsm-bridge 127.0.0.
↪1
Added KMIP server 'cascade-hsm-bridge 0.1.0-alpha using PKCS#11 token with label NL-
↪smartcard (UserPIN) in slot Identive CLOUD 2700 R Smart Card Reader [CCID Interface]␣
↪(536... via library opensc-pkcs11.so'.
```

1.15.8 Sign a Test Zone with SmartCard-HSM

Create a test zone to load and sign and ensure the Cascade daemon has access to it:

```
# mkdir /etc/cascade/zones
# cat > /etc/cascade/zones/example.net << EOF
example.net.    3600    IN      SOA     ns.example.net. username.example.net. 1 86400␣
↪7200 2419200 300
example.net.            IN      NS      ns
ns                      IN      A       192.0.2.1
EOF
# chown -R cascade: /etc/cascade/zones
```

Add our test zone to Cascade and associate the policy that we created with the zone:

```
# cascade zone add --source /etc/cascade/zones/example.net --policy smartcard example.net
Added zone example.net
```

Check that the zone has been signed, and print out additional information which includes the identifiers of the signing keys that were used:

```
# cascade zone status example.net --detailed
Status report for zone 'example.net' using policy 'smartcard'
✓ Waited for a new version of the example.net zone
✓ Loaded version 1
  Loaded at 2025-10-09T14:58:11+00:00 (26s ago)
  Loaded 196 B and 3 records from the filesystem in 0 seconds
✓ Auto approving signing of version 1, no checks enabled in policy.
✓ Approval received to sign version 1, signing requested
✓ Signed version 1 as version 2025100901
  Signing requested at 2025-10-09T14:58:11+00:00 (26s ago)
  Signing started at 2025-10-09T14:58:11+00:00 (25s ago)
  Signing finished at 2025-10-09T14:58:11+00:00 (25s ago)
  Collected 3 records in 0s, sorted in 0s
  Generated 5 NSEC(3) records in 0s
  Generated 5 signatures in 0s (5 sig/s)
  Inserted signatures in 0s (5 sig/s)
  Took 0s in total, using 2 threads
  Current action: Finished
✓ Auto approving publication of version 2025100901, no checks enabled in policy.
✓ Published version 2025100901
  Published zone available on 127.0.0.1:4543
DNSSEC keys:
  KSK tagged 15202:
    Reference: kmip://cascade-hsm-bridge/keys/CE8E308B232C890B54066E6D3CF85802FB6B27F8_
↪pub?algorithm=13&flags=257
    Actively used for signing
  ZSK tagged 43092:
    Reference: kmip://cascade-hsm-bridge/keys/870D4E7E7A1C89A14D3A8FD14BDC953D249093D9_
↪pub?algorithm=13&flags=256
    Actively used for signing
   ...
```

1.15.9 Inspect the SmartCard HSM

Use the **pkcs11-tool** program from the **opensc** package installed earlier to list objects on the SmartCard-HSM. Initially the card will likely be empty, but after Cascade has created some keys you should see the objects on the card.

```
# pkcs11-tool --module opensc-pkcs11.so --list-objects
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:  ␣
↪044104084d2c0a3b1645ef07e898526d2cac0a44d127703209dcd98d484c14dafb7cfa035d7bc903a6b695ecfd830c610be39
  EC_PARAMS:  06082a8648ce3d030107
  label:      example.net-15202-ksk-pri
  ID:         ce8e308b232c890b54066e6d3cf85802fb6b27f8
  Usage:      verify
  Access:     none
```

Note how the key tag or key IDs created by Cascade match the `labels` on the Smart Card: `<zone name>-<key tag>-<key type>-pri`.

End.

— Contributed by Jan-Piet Mens

1.16 Integrating with a Nitrokey NetHSM

> **Note**
>
> The instructions on this page are for a Debian 12 host and assume that Cascade has already been installed using our DEB package, and that Docker is installed and can run the NetHSM container.

> **Note**
>
> The instructions on this page assume you will be using the NetHSM exclusively for the task at hand and for testing. Using the provided test image is in no way suggested for production. Please study the product documentation to find out how to do so.

> **Note**
>
> Most instructions on this page assume you will be working as a normal, i.e. non-`root`, user.

> NetHSM is an open hardware security module created and distributed by Nitrokey. It's a secure store for cryptographic keys powered by open source, which enables people to verify exactly how it works. The HSM is accessible through a REST interface, and a PKCS #11 shared object interfaces that with Cascade's **`cascade-hsm-bridge`**. The software is also provided as an OCI image (used here for demonstration purposes) which can be used with Docker or Podman for experimentation without having to purchase the device proper.
>
> —https://www.nitrokey.com/products/nethsm

1.16.1 Launch the NetHSM container

We launch the container to be removed on exit which means all settings and keys will be wiped when the container is stopped.

```
$ docker run --rm -ti -p 127.0.0.1:8443:8443 docker.io/nitrokey/nethsm:testing
```

1.16.2 Install the Prerequisites

```
$ sudo apt install -y opensc opensc-pkcs11 pipx
$ pipx install pynitrokey    # for the nitropy utility
$ pipx ensurepath
$ source ~/.bashrc
$ export NETPKCS="/usr/lib/x86_64-linux-gnu/nethsm-pkcs11.so"
```

We download and install the NetHSM PKCS#11 driver to a somewhat shorter name; `$NETPKCS` will help us keep command-lines short in this documentation.

```
$ wget https://github.com/Nitrokey/nethsm-pkcs11/releases/download/v1.7.2/nethsm-pkcs11-
↪v1.7.2-x86_64-linux-glibc.so
$ sudo install nethsm-pkcs11-v1.7.2-x86_64-linux-glibc.so $NETPKCS
```

1.16.3 Configure the NetHSM PKCS#11 driver

We then configure the driver, in the file `/etc/nitrokey/p11nethsm.conf`, paying attention to the URI, usernames, and passwords. as per NetHSM's PKCS#11 Setup documentation

```
enable_set_attribute_value: false
log_level: Debug
syslog_facility: "user"

slots:
  - label: LocalHSM                        # Name your NetHSM however you want
    description: Local HSM (docker)        # Optional description
    operator:
      username: "jj01"
      password: "blab123456"
    administrator:
      username: "admin18"
      password: "secret9999"

    instances:
      - url: "https://127.0.0.1:8443/api/v1"   # URL to reach the server
        max_idle_connections: 10
        danger_insecure_cert: false
    retries:
      count: 3
      delay_seconds: 1
    tcp_keepalive:
      time_seconds: 600
      interval_seconds: 60
      retries: 3
    connections_max_idle_duration: 1800
    timeout_seconds: 10
```

1.16.4 Provision the HSM

Configure access to the NetHSM, ensuring IP address and port number match those of the container running the HSM. We show passwords in clear below so as to be able to demonstrate where they are later used. Please and obviously don't use these. As we haven't configured a TLS key and certificate for the device we disable TLS verification on the connection (not recommended).

```
$ export NETHSM_HOST=127.0.0.1:8443

$ nitropy nethsm --no-verify-tls provision
Command line tool to interact with Nitrokey devices 0.11.2
Unlock passphrase: nlnetlabs001
Repeat for confirmation:
Admin passphrase: lecascadeur
Repeat for confirmation:
Warning: The unlock passphrase cannot be reset without knowing the current value...
NetHSM 127.0.0.1:8443 provisioned
```

1.16.5 Configure NetHSM's TLS certificate

These steps are optional and probably not worth doing for the ephemeral HSM test container which will have its data destroyed when it's stopped. You must then remember to add `--no-verify-tls` to all subsequent **nitropy** commands. However, when running on a productive NetHSM, and since we are actually talking to an HSM we should endeavour to communicate securely.

Generate a certificate signing request on the NetHSM. (Until our certificate is added to the NetHSM we must disable TLS certificate verification.)

```
$ nitropy nethsm --no-verify-tls csr \
      --api \
      --country="NL" \
      --state-or-province="North Holland" \
      --locality="Amsterdam" \
      --organization="NLnet Labs" \
      --organizational-unit="Cascade" \
      --common-name="nethsm.example.net" \
      --email-address="info@example.net"
Command line tool to interact with Nitrokey devices 0.11.2
[auth] User name for NetHSM 127.0.0.1:8443: admin
[auth] Password for user admin on NetHSM 127.0.0.1:8443: lecascadeur
-----BEGIN CERTIFICATE REQUEST-----
MIIBjDCCATECAQAwgZ4xGzAZBgNVBAMMEm5ldGhzbS5leGFtcGxlLm5ldDELMAkG
...
aRRIYefZ9EB/6NoULVJjTQ==
-----END CERTIFICATE REQUEST-----
```

Have the certificate signed by a Certification Authority (CA) and verify the certificate is what we expect. (Not shown here, but our CA has added SANs for the IP address(es) of the device.)

```
$ openssl x509 -in nethsm1.crt -noout -subject
subject= /CN=nethsm.example.net/C=NL/L=Amsterdam/ST=North Holland/O=NLnet Labs/
↪OU=Cascade/emailAddress=info@example.net
```

Overwrite the device's self-signed certificate with that which we received from the CA, in PEM format.

```
$ nitropy nethsm --no-verify-tls  set-certificate --api /tmp/nethsm1.crt
Command line tool to interact with Nitrokey devices 0.11.2
[auth] User name for NetHSM 127.0.0.1:8443: admin
[auth] Password for user admin on NetHSM 127.0.0.1:8443: lecascadeur
Updated the API certificate for NetHSM 127.0.0.1:8443
```

Verify a connection to the NetHSM can be validated by our CA certificate by specifying the `--ca-certs` option to **nitropy**

```
$ nitropy nethsm --ca-certs ca.crt info
Command line tool to interact with Nitrokey devices 0.11.2
Host:    127.0.0.1:8443
Vendor:  Nitrokey GmbH
Product: NetHSM
```

Install our CA certificate on the system. This certificate bundle (store) will typically be used by programs on our host.

```
$ sudo mkdir /usr/local/share/ca-certificates/nethsm-ca
$ sudo install -m444 ca.crt /usr/local/share/ca-certificates/nethsm-ca
$ sudo update-ca-certificates
```

Sadly Python uses a distinct certificate store, and becasue **nitropy** is written in Python, we determine which file Python will search for certificates and add ours to that. (If **nitropy** was install with **pipx**, the path to **python3** will likely be `~/.local/pipx/venvs/pynitrokey/bin/python`.)

```
$ python3
>>> import certifi
>>> print(certifi.where())
/etc/ssl/certs/ca-certificates.crt

$ sudo tee -a /etc/ssl/certs/ca-certificates.crt < ca.crt
```

We can now access the NetHSM with a verified TLS connection and need neither disable verification (`--no-verify-tls`) nor always use the `--ca-certs` option.

```
$ nitropy nethsm info
Command line tool to interact with Nitrokey devices 0.11.2
Host:    127.0.0.1:8443
Vendor:  Nitrokey GmbH
Product: NetHSM
```

1.16.6 Add a dedicated user

We add a dedicated user with which Cascade's **cascade-hsm-bridge** will connect to and interact with the NetHSM. (It is possible to have other programs use the same HSM with distinct usernames.)

```
$ nitropy nethsm add-user \
    --real-name "Jane Jolie" \
    --role Operator \
    --user-id jj01 \
    --passphrase blab123456
 Command line tool to interact with Nitrokey devices 0.11.2
 [auth] User name for NetHSM 127.0.0.1:8443: admin
 [auth] Password for user admin on NetHSM 127.0.0.1:8443: lecascadeur
 User jj01 added to NetHSM 127.0.0.1:8443
```

1.16.7 Verify NetHSM is accessible and show its slots

The NetHSM should now be configured and we can attempt to access it via its PKCS#11 interface. (If logging has been configured for the driver, the following command will cause logs to be written.)

```
$ pkcs11-tool --module $NETPKCS --show-info
Cryptoki version 3.1
Manufacturer     Nitrokey
Library          Nitrokey NetHsm PKCS#11 library (ver 1.7)
Using slot 0 with a present token (0x0)

$ pkcs11-tool --module $NETPKCS --list-slots
Available slots:
Slot 0 (0x0): NetHSM
```

(continues on next page)

(continued from previous page)

```
  token label        : LocalHSM
  token manufacturer : Nitrokey GmbH
  token model        : NetHSM
  token flags        : rng, token initialized, PIN initialized
  hardware version   : 0.1
  firmware version   : 3.1
  serial num         : 0000000000
  pin min/max        : 0/0
Slot 1 (0x1): NetHSM
  token label        : LocalHSM
  token manufacturer : Nitrokey GmbH
  token model        : NetHSM
  token flags        : rng, token initialized, PIN initialized
  hardware version   : 0.1
  firmware version   : 3.1
  serial num         : 0000000000
  pin min/max        : 0/0
```

1.16.8 List the HSM's mechanisms

```
$ pkcs11-tool --module $NETPKCS --list-mechanisms
Using slot 0 with a present token (0x0)
Supported mechanisms:
  AES-CBC, keySize={128,256}, hw, encrypt, decrypt, generate
  RSA-X-509, keySize={1024,8192}, hw, decrypt
  RSA-PKCS, keySize={1024,8192}, hw, decrypt, sign, generate_key_pair
  SHA1-RSA-PKCS, keySize={1024,8192}, hw, decrypt, sign, generate_key_pair
  SHA224-RSA-PKCS, keySize={1024,8192}, hw, decrypt, sign, generate_key_pair
  SHA256-RSA-PKCS, keySize={1024,8192}, hw, decrypt, sign, generate_key_pair
 ...
```

1.16.9 Configure `cascade-hsm-bridge`

`cascade-hsm-bridge` needs to know where to find the NetHSM PKCS#11 module. As PKCS#11 modules are loaded into a host application, any access to resources needed by the PKCS#11 module must be granted to the host application.

```
# sed -i -e 's|^lib_path = .\+|lib_path = "/usr/lib/x86_64-linux-gnu/nethsm-pkcs11.so"|' /
↪etc/cascade-hsm-bridge/config.toml
# systemctl start cascade-hsm-bridge
```

1.16.10 Create a Cascade Policy that uses your HSM

Create a Cascade policy called `nethsm` and set it to use the HSM called `cascade-hsm-bridge` we configured earlier.

```
# cascade template policy | tee /etc/cascade/policies/nethsm.toml
# sed -i -e 's|^#hsm-server-id = .\+|hsm-server-id = "cascade-hsm-bridge"|' /etc/cascade/
↪policies/nethsm.toml
```

Start the Cascade daemon:

```
# systemctl start cascaded
# cascade policy reload
Policies reloaded:
- nethsm added
```

Configure a HSM in Cascade called `cascade-hsm-bridge` that will connect to the locally running **`cascade-hsm-bridge`** daemon. The `username` is the slot identifier we found in our NetHSM earlier, and the `password` anything – it isn't actually used here, as the username/password with which we'll connect to the NetHSM has been configured in `p11nethsm.conf` above.

```
# cascade hsm add --insecure --username 0 --password "123456" cascade-hsm-bridge 127.0.0.
↪1
Added KMIP server 'cascade-hsm-bridge 0.1.0-alpha using PKCS#11 token with label␣
↪LocalHSM in slot NetHSM via library nethsm-pkcs11.so'
```

1.16.11 Sign a Test Zone with NetHSM

Create a test zone to load and sign and ensure the Cascade daemon has access to it:

```
# mkdir /etc/cascade/zones
# cat > /etc/cascade/zones/example.net << EOF
example.net.    3600    IN      SOA     ns.example.net. username.example.net. 1 86400␣
↪7200 2419200 300
example.net.            IN      NS      ns
ns                      IN      A       192.0.2.1
EOF
# chown -R cascade: /etc/cascade/zones
```

Add our test zone to Cascade and associate the policy that we created with the zone:

```
# cascade zone add --source /etc/cascade/zones/example.net --policy nethsm example.net
Added zone example.net
```

Check that the zone has been signed, and print out additional information which includes the identifiers of the signing keys that were used:

```
$ cascade zone status example.net
Status report for zone 'example.net' using policy 'nethsm'
✓ Waited for a new version of the example.net zone
✓ Loaded version 3
  Loaded at 2025-11-22T14:42:20+00:00 (1h 11m 36s ago)
  Loaded 196 B and 3 records from the filesystem in 0 seconds
✓ Auto approving signing of version 3, no checks enabled in policy.
✓ Approval received to sign version 3, signing requested
✓ Signed version 3 as version 1763826146
  Signing requested at 2025-11-22T15:42:26+00:00 (11m 30s ago)
  Signing started at 2025-11-22T15:42:26+00:00 (11m 30s ago)
  Signing finished at 2025-11-22T15:42:26+00:00 (11m 30s ago)
  Collected 3 records in 0s, sorted in 0s
  Generated 2 NSEC(3) records in 0s
  Generated 5 signatures in 0s (5 sig/s)
  Inserted signatures in 0s (5 sig/s)
  Took 0s in total, using 2 threads
```

(continues on next page)

(continued from previous page)

```
  Current action: Finished
✓ Waited for approval to publish version 1763826146
✓ Published version 1763826146
  Published zone available on 127.0.0.1:4543

$ dig @127.0.0.1 -p 4543 example.net DNSKEY +nocrypto +norec +noedns
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11653
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; ANSWER SECTION:
example.net.         3600    IN      DNSKEY  257 3 13 [key id = 31203]

;; Query time: 0 msec
;; SERVER: 127.0.0.1#4543(127.0.0.1) (UDP)
;; WHEN: Sat Nov 22 16:56:00 CET 2025
;; MSG SIZE  rcvd: 131
```

1.16.12 Inspect the keys directly on the NetHSM

Use the **nitropy** program installed earlier to list objects on the NetHSM. We see an object created by Cascade on the device.

```
$ nitropy nethsm -u jj01 -p blab123456 list-keys
Command line tool to interact with Nitrokey devices 0.11.2
Keys on NetHSM 127.0.0.1:8443:

Key ID                                    Type    Mechanisms      Operations      Tags
----------------------------------------  ------- --------------- ----------      ----
3fccaf83ff24bde4e0d3ee036acad36dec76bd8a  EC_P256 ECDSA_Signature 16
```

The **pkcs11-tool** can list the objects it sees via the PKCS#11 interface.

```
$ pkcs11-tool --module $NETPKCS --list-objects
Using slot 0 with a present token (0x0)
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:  ␣
↪04410454a0f26046607a1606788bf116ad348125948d7da55dfe581f3c7e8cefb2b57bdce49a5884fad0d86a20b7e3e63f726
  EC_PARAMS:  06082a8648ce3d030107
  label:      3fccaf83ff24bde4e0d3ee036acad36dec76bd8a
  ID:         ␣
↪3366636361663833666632346264653465306433656530333661636164333664656337366264386 1
  Usage:      none
  Access:     none
Private Key Object; EC
  label:      3fccaf83ff24bde4e0d3ee036acad36dec76bd8a
  ID:         ␣
↪3366636361663833666632346264653465306433656530333661636164333664656337366264386 1
  Usage:      sign, derive
  Access:     sensitive, always sensitive, never extractable
  Allowed mechanisms: ECDSA
```

1.16.13 Key labels

In order to determine which key on the NetHSM belongs to a zone, we can use **cascade** to output detailed information about the zone status.

```
$ cascade zone status example.net --detailed
Status report for zone 'example.net' using policy 'csk13-hsm'
✓ Waited for a new version of the example.net zone
✓ Loaded version 3
...
 key kmip://cascade-hsm-bridge/keys/3FCCAF83FF24BDE4E0D3EE036ACAD36DEC76BD8A_pub?
↪algorithm=13&flags=257 does not expire. No validity period is configured for the key␣
↪type
```

The lowercased value of *3FCCAF83FF24BDE4E0D3EE036ACAD36DEC76BD8A* is the label of the key as reported by the above tools.

1.16.14 Final notes

It doesn't appear to be possible to determine which key on the NetHSM corresponds to which zone. Contrary to keys generated on the *SmartCard-HSM*, the key labels are random names. Changing the value of `enable_set_attribute_value` in `p11nethsm.conf` doesn't seem to make a difference.

End.

— Contributed by Jan-Piet Mens

1.17 Frequently Asked Questions

1.17.1 Design and Architecture

Why did you build this project in Rust?

Because Cascade is written in the Rust programming language, it is significantly less likely to crash or suffer from memory safety issues. Rust also makes it easier to leverage the higher core count of modern computers through "fearless concurrency".

See also

Rust Programming Language website
An overview of Rust's feature set.

Do I need separate database software to run Cascade?

No, the Cascade pipeline runs as a single binary and no additional database software is required. Cascade stores its state in on-disk files in JSON format, by default at various locations under a single parent directory. As such, state is human-readable and easily backed up.

See also

Architecture
An overview of Cascade's design.

Do I need to use a HSM to run Cascade?

No, Cascade does not require a Hardware Security Module (HSM) to operate. While it is common practice to secure cryptographic key material using an HSM, not all operators use an HSM. Cascade is able to use OpenSSL and/or ring software cryptography to generate signing keys and to cryptographically sign DNS RRset data, storing the generated keys in on-disk files.

Why is the default policy KSK/ZSK and not CSK?

Key rolls should be automatic and frequent. Frequent key rolls help to ensure that they become normal operational practice and not an exception. Key rolls should be automated as much as possible to avoid mistakes. Unfortunately, the standard for updating DS records (CDS, **RFC 8078**) is not widely implemented so in many cases a KSK roll has to have a manual component, namely submitting and updating the DS record at the parent.

These factors favor the KSK/ZSK split because it makes frequent ZSK rolls possible and KSK rolls can be performed less frequently.

Finally, KSK and ZSK key rolls are less complex than CSK rolls. Some people use a CSK and never roll the key, which avoids the key roll complexity but leads to a lack of operational practice when a situation arises that a key roll is needed.

Why are the default key lifetimes the way they are?

A ZSK roll requires re-signing the entire zone. For bigger zones, this should not be done too often to keep the overhead of the key roll low. Once a month seems a good compromise.

A KSK roll requires updating the DS RRset in the parent zone. For this reason, rolling a KSK once a year is reasonable.

See also

Hardware Security Modules (HSMs)
Hardware Security Modules (HSMs).

1.17.2 Installing ang Building

Can I build Cascade with LibreSSL?

No, OpenSSL 3.x is required as these versions fully support Edwards-curve Digital Security Algorithm (EdDSA) keys and signatures using the Ed448 curve (DNSSEC algorithm 16). By contrast, LibreSSL does not yet have support for Ed448.

Ed448 was standardized for use with DNSSEC in February 2017 (**RFC 8080**) and has been a RECOMMENDED algorithm since June 2019 (**RFC 8624**).

See also

Install OpenSSL
Installing OpenSSL on common distributions, such as Debian, Ubuntu and Red Hat Enterprise Linux.

1.18 Known Limitations

Important

Cascade is a hidden signer. As such, it is *not* a complete authoritative DNS server. Cascade will not reply with the AA or AD flag set, nor can it reply to DNSSEC queries. Instead, Cascade is intended to be used with a proper secondary serving the signed zones to actual clients.

1.18.1 Expectations for the Beta Release

Tip

This page details what you can expect from Cascade in its beta form. Our goal is to gather operator feedback. Please *reach out* to us.

- The included functionality should work correctly for simple scenarios with correct inputs when running on setups (O/S, HSM) that we have tested on.
- Handling of incorrect inputs, edge cases, more complex scenarios, non-default policy settings, and so on *may be incomplete or incorrect*. Please *report any bugs you find*
- The user experience is a *work-in-progress*. The goal of Cascade is not only to be a correctly functioning DNSSEC signer which makes it easy to do the right thing and hard to do the wrong thing, it should also be obvious how to use it and be clear what the system did, is doing now and will do in the future. But we're not there yet, we have more ideas but *we'd love to hear yours too*.

1.18.2 Policy Edits Require Explicit Reload

Users may expect that edits to policy files will take effect if Cascade is restarted, however this is not the case. Cascade deliberately does not reload the policy files until explicitly told to do so via `cascade policy reload`.

This design ensures that a restart doesn't suddenly cause unexpected changes in behaviour, e.g. config file edits that were made but never actually used and then forgotten about.

1.18.3 Differences to OpenDNSSEC

Improvements

- An HSM is not required.
- More suited to containerized usage:
  - Supports stdout/stderr logging as well as syslog.
  - Single daemon per image.

- Rust.
- Observability (Still a Work-In-Progress).
- No XML.
- No database.
- No file based communication between daemons.
- Finer grained control over and insight into key states.
- Jitter is replaced with a deterministic algorithm for updating signatures.

Missing features

The beta release of Cascade is missing some of the features provided by OpenDNSSEC that will be added in a future release:

- File output.
- Holding keys for use until a backup flag is set.
- Sharing of keys between zones.
- Passthrough mode.
- Prefix based access control.
- CAA record support.
- Terminology differences, Cascade does not use the term "omnipresent" for example.

1.18.4 Other known limitations

- No NOTIFY retry support.
- No NOTIFY "Notify Set" (RFC 1996) discovery.
- No KMIP batching support.
- No DNS UPDATE support.
- HSM algorithm support is limited to RSASHA256 and ECDSAP256SHA256.
- Changing a policy to use an HSM will not affect existing zones.
- Memory usage can be improved.

1.19 Glossary

> **Note**
>
> This page lists DNS-related terms used in the Cascade documentation, sourced from RFC 9499.

Apex (Zone)
: The point in the tree at an owner of an SOA and corresponding authoritative NS RRset. This is also called the "zone apex". RFC 4033 defines it as "the name at the child's side of a zone cut". The "apex" can usefully be thought of as a data-theoretic description of a tree structure, and "origin" is the name of the same concept when it is implemented in zone files. The distinction is not always maintained in use, however, and one can find uses that conflict subtly with this definition. RFC 1034 uses the term "top node of the zone" as a synonym of

"apex", but that term is not widely used. These days, the first sense of "origin" (above) and "apex" are often used interchangeably.

Bogus (DNSSEC State)
RFC 4033 Section 5 says: "The validating resolver has a trust anchor and a secure delegation indicating that subsidiary data is signed, but the response fails to validate for some reason: missing signatures, expired signatures, signatures with unsupported algorithms, data missing that the relevant NSEC RR says should be present, and so forth."

RFC 4035 Section 4.3 says: "An RRset for which the resolver believes that it ought to be able to establish a chain of trust but for which it is unable to do so, either due to signatures that for some reason fail to validate or due to missing data that the relevant DNSSEC RRs indicate should be present. This case may indicate an attack but may also indicate a configuration error or some form of data corruption."

Child (Zone)
"The entity on record that has the delegation of the domain from the Parent." (Quoted from RFC 7344 Section 1.1)

Combined signing key (CSK)
"In cases where the differentiation between the KSK and ZSK is not made, i.e., where keys have the role of both KSK and ZSK, we talk about a Single-Type Signing Scheme." (Quoted from RFC 6781 Section 3.1) This is sometimes called a "combined signing key" or "CSK". It is operational practice, not protocol, that determines whether a particular key is a ZSK, a KSK, or a CSK.

DNSSEC Policy (DP)
A statement that "sets forth the security requirements and standards to be implemented for a DNSSEC-signed zone." (Quoted from RFC 6841 Section 2)

DNSSEC Practice Statement (DPS)
"A practices disclosure document that may support and be a supplemental document to the DNSSEC Policy (if such exists), and it states how the management of a given zone implements procedures and controls at a high level." (Quoted from RFC 6841 Section 2)

Hardware security module (HSM)
A specialized piece of hardware that is used to create keys for signatures and to sign messages without ever disclosing the private key. In DNSSEC, HSMs are often used to hold the private keys for KSKs and ZSKs and to create the signatures used in RRSIG records at periodic intervals.

Indeterminate (DNSSEC State)
RFC 4033 Section 5 says: "There is no trust anchor that would indicate that a specific portion of the tree is secure. This is the default operation mode."

RFC 4035 Section 4.3 says: "An RRset for which the resolver is not able to determine whether the RRset should be signed, as the resolver is not able to obtain the necessary DNSSEC RRs. This can occur when the security-aware resolver is not able to contact security-aware name servers for the relevant zones."

Insecure (DNSSEC State)
RFC 4033 Section 5 says: "The validating resolver has a trust anchor, a chain of trust, and, at some delegation point, signed proof of the non- existence of a DS record. This indicates that subsequent branches in the tree are provably insecure. A validating resolver may have a local policy to mark parts of the domain space as insecure."

RFC 4035 Section 4.3 says: "An RRset for which the resolver knows that it has no chain of signed DNSKEY and DS RRs from any trusted starting point to the RRset. This can occur when the target RRset lies in an unsigned zone or in a descendent [sic] of an unsigned zone. In this case, the RRset may or may not be signed, but the resolver will not be able to verify the signature."

Insecure delegation
"A signed name containing a delegation (NS RRset), but lacking a DS RRset, signifying a delegation to an unsigned subzone." (Quoted from RFC 4956 Section 2)

Key signing key (KSK)
DNSSEC keys that "only sign the apex DNSKEY RRset in a zone." (Quoted from RFC 6781 Section 3.1)

NSEC
"The NSEC record allows a security-aware resolver to authenticate a negative reply for either name or type non-existence with the same mechanisms used to authenticate other DNS replies." (Quoted from RFC 4033 Section 3.2 In short, an NSEC record provides authenticated denial of existence.

"The NSEC resource record lists two separate things: the next owner name (in the canonical ordering of the zone) that contains authoritative data or a delegation point NS RRset, and the set of RR types present at the NSEC RR's owner name." (Quoted from RFC 4034 Section 4.

NSEC3
Like the NSEC record, the NSEC3 record also provides authenticated denial of existence; however, NSEC3 records mitigate zone enumeration and support Opt-Out. NSEC3 resource records require associated NSEC3PARAM resource records. NSEC3 and NSEC3PARAM resource records are defined in RFC 5155.

Note that RFC 6840 says that RFC 5155 "is now considered part of the DNS Security Document Family as described by RFC 4033 Section 10". This means that some of the definitions from earlier RFCs that only talk about NSEC records should probably be considered to be talking about both NSEC and NSEC3.

Online signing
RFC 4470 defines "on-line signing" (note the hyphen) as "generating and signing these records on demand", where "these" was defined as NSEC records. The current definition expands that to generating and signing RRSIG, NSEC, and NSEC3 records on demand.

Opt-out
"The Opt-Out Flag indicates whether this NSEC3 RR may cover unsigned delegations." (Quoted from RFC 5155 Section 3.1.2.1) Opt-out tackles the high costs of securing a delegation to an insecure zone. When using Opt-Out, names that are an insecure delegation (and empty non-terminals that are only derived from insecure delegations) don't require an NSEC3 record or its corresponding RRSIG records. Opt-Out NSEC3 records are not able to prove or deny the existence of the insecure delegations. (Adapted from RFC 7129 Section 5.1)

Origin (Zone)
There are two different uses for this term:

1. "The domain name that appears at the top of a zone (just below the cut that separates the zone from its parent)... The name of the zone is the same as the name of the domain at the zone's origin." (Quoted from RFC 2181 Section 6) These days, this sense of "origin" and "apex" (defined below) are often used interchangeably.
2. The domain name within which a given relative domain name appears in zone files. Generally seen in the context of "$ORIGIN", which is a control entry defined in RFC 1035 Section 5.1, as part of the master file format. For example, if the $ORIGIN is set to `example.org.`, then a master file line for "www" is in fact an entry for `www.example.org.`.

Parent (Zone)
"The domain in which the Child is registered." (Quoted from RFC 7344 Section 1.1) Earlier, "parent name server" was defined in RFC 0882 as "the name server that has authority over the place in the domain name space that will hold the new domain". (Note that RFC 0882 was obsoleted by RFC 1034 and RFC 1035.) RFC 819 also has some description of the relationship between parents and children.

Primary server
"Any authoritative server configured to be the source of zone transfer for one or more [secondary] servers." (Quoted from RFC 1996 Section 2.1) Or, more specifically, RFC 2136 calls it "an authoritative server configured to be the source of AXFR or IXFR data for one or more [secondary] servers". Primary servers are also discussed in RFC 1034. Although early DNS RFCs such as RFC 1996 referred to this as a "master", the current common usage has shifted to "primary".

Recursive resolver
A resolver that acts in recursive mode. In general, a recursive resolver is expected to cache the answers it receives (which would make it a full-service resolver), but some recursive resolvers might not cache.

RFC 4697 tried to differentiate between a recursive resolver and an iterative resolver.

Resource Record Set (RRset)
A set of resource records "with the same label, class and type, but with different data" (according to RFC 2181 Section 5). Also written as "RRSet" in some documents. As a clarification, "same label" in this definition means "same owner name". In addition, RFC 2181 states that "the TTLs of all RRs in an RRSet must be the same".

Note that RRSIG resource records do not match this definition. RFC 4035 says:

> "An RRset MAY have multiple RRSIG RRs associated with it. Note that as RRSIG RRs are closely tied to the RRsets whose signatures they contain, RRSIG RRs, unlike all other DNS RR types, do not form RRsets. In particular, the *TTL* values among RRSIG RRs with a common owner name do not follow the RRset rules described in RFC 2181."

Secondary server
"An authoritative server which uses zone transfer to retrieve the zone." (Quoted from RFC 1996 Section 2.1) Secondary servers are also discussed in RFC 1034. RFC 2182 describes secondary servers in more detail. Although early DNS RFCs such as RFC 1996 referred to this as a "slave", the current common usage has shifted to calling it a "secondary".

Secure (DNSSEC State)
RFC 4033 Section 5 says: "The validating resolver has a trust anchor, has a chain of trust, and is able to verify all the signatures in the response."

RFC 4035 Section 4.3 says: "An RRset for which the resolver is able to build a chain of signed DNSKEY and DS RRs from a trusted security anchor to the RRset. In this case, the RRset should be signed and is subject to signature validation, as described above."

Secure Entry Point (SEP)
A flag in the DNSKEY RDATA that "can be used to distinguish between keys that are intended to be used as the secure entry point into the zone when building chains of trust, i.e., they are (to be) pointed to by parental DS RRs or configured as a trust anchor.... Therefore, it is suggested that the SEP flag be set on keys that are used as KSKs and not on keys that are used as ZSKs, while in those cases where a distinction between a KSK and ZSK is not made (i.e., for a Single-Type Signing Scheme), it is suggested that the SEP flag be set on all keys." (Quoted from RFC 6781 Section 3.2.3) Note that the SEP flag is only a hint, and its presence or absence may not be used to disqualify a given DNSKEY RR from use as a KSK or ZSK during validation.

The original definition of SEPs was in RFC 3757. That definition clearly indicated that the SEP was a key, not just a bit in the key. The abstract of RFC 3757 says: "With the Delegation Signer (DS) resource record (RR), the concept of a public key acting as a secure entry point (SEP) has been introduced. During exchanges of public keys with the parent there is a need to differentiate SEP keys from other public keys in the Domain Name System KEY (DNSKEY) resource record set. A flag bit in the DNSKEY RR is defined to indicate that DNSKEY is to be used as a SEP." That definition of the SEP as a key was made obsolete by RFC 4034, and the definition from RFC 6781 is consistent with RFC 4034.

Signed zone
"A zone whose RRsets are signed and that contains properly constructed DNSKEY, Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records." (Quoted from RFC 4033 Section 2) It has been noted in other contexts that the zone itself is not really signed, but all the relevant RRsets in the zone are signed. Nevertheless, if a zone that should be signed contains any RRsets that are not signed (or opted out), those RRsets will be treated as bogus, so the whole zone needs to be handled in some way.

It should also be noted that, since the publication of RFC 6840, NSEC records are no longer required for signed zones: a signed zone might include NSEC3 records instead. RFC 7129 provides additional background commentary and some context for the NSEC and NSEC3 mechanisms used by DNSSEC to provide authenticated

denial- of-existence responses. NSEC and NSEC3 are described below.

Trust anchor
"A configured DNSKEY RR or DS RR hash of a DNSKEY RR. A validating security-aware resolver uses this public key or hash as a starting point for building the authentication chain to a signed DNS response. In general, a validating resolver will have to obtain the initial values of its trust anchors via some secure or trusted means outside the DNS protocol." (Quoted from **RFC 4033 Section 2**)

TTL
The maximum "time to live" of a resource record. "A TTL value is an unsigned number, with a minimum value of 0, and a maximum value of 1. That is, a maximum of 2^31 - 1. When transmitted, this value shall be encoded in the less significant 31 bits of the 32 bit TTL field, with the most significant, or sign, bit set to zero." (Quoted from **RFC 2181 Section 8**) Note that **RFC 1035** erroneously stated that this is a signed integer; that was fixed by **RFC 2181**.

The TTL "specifies the time interval that the resource record may be cached before the source of the information should again be consulted." (Quoted from **RFC 1035 Section 3.2.1**) **RFC 1035 Section 4.1.3** states "the time interval (in seconds) that the resource record may be cached before it should be discarded". Despite being defined for a resource record, the TTL of every resource record in an RRset is required to be the same (**RFC 2181 Section 5.2**).

The reason that the TTL is the maximum time to live is that a cache operator might decide to shorten the time to live for operational purposes, for example, if there is a policy to disallow TTL values over a certain number. Some servers are known to ignore the TTL on some RRsets (such as when the authoritative data has a very short TTL) even though this is against the advice in **RFC 1035**. An RRset can be flushed from the cache before the end of the TTL interval, at which point, the value of the TTL becomes unknown because the RRset with which it was associated no longer exists.

There is also the concept of a "default TTL" for a zone, which can be a configuration parameter in the server software. This is often expressed by a default for the entire server, and a default for a zone using the $TTL directive in a zone file. The `$TTL` directive was added to the master file format by **RFC 2308**.

Unsigned zone
RFC 4033 Section 2 defines this as "a zone that is not signed". **RFC 4035 Section 2** defines this as a "zone that does not include these records [properly constructed DNSKEY, Resource Record Signature (RRSIG), Next Secure (NSEC), and (optionally) DS records] according to the rules in this section…" There is an important note at the end of **RFC 4035 Section 5.2** that defines an additional situation in which a zone is considered unsigned: "If the resolver does not support any of the algorithms listed in an authenticated DS RRset, then the resolver will not be able to verify the authentication path to the child zone. In this case, the resolver SHOULD treat the child zone as if it were unsigned."

Validating resolver
A security-aware recursive name server, security-aware resolver, or security-aware stub resolver that is applying at least one of the definitions of validation (above) as appropriate to the resolution context. For the same reason that the generic term "resolver" is sometimes ambiguous and needs to be evaluated in context, "validating resolver" is a context-sensitive term.

Validation
Validation, in the context of DNSSEC, refers to one of the following:

- Checking the validity of DNSSEC signatures,
- Checking the validity of DNS responses, such as those including authenticated denial of existence, or
- Building an authentication chain from a trust anchor to a DNS response or individual DNS RRsets in a response.

The first two definitions above consider only the validity of individual DNSSEC components, such as the RRSIG validity or NSEC proof validity. The third definition considers the components of the entire DNSSEC authen-

tication chain; thus, it requires "configured knowledge of at least one authenticated DNSKEY or DS RR" (as described in **RFC 4035 Section 5**).

RFC 4033 Section 2, says that a "Validating Security-Aware Stub Resolver… performs signature validation" and uses a trust anchor "as a starting point for building the authentication chain to a signed DNS response"; thus, it uses the first and third definitions above. The process of validating an RRSIG resource record is described in **RFC 4035 Section 5.3**.

RFC 5155 refers to validating responses throughout the document in the context of hashed authenticated denial of existence; this uses the second definition above.

The term "authentication" is used interchangeably with "validation", in the sense of the third definition above. **RFC 4033 Section 2**, describes the chain linking trust anchor to DNS data as the "authentication chain". A response is considered to be authentic if "all RRsets in the Answer and Authority sections of the response [are considered] to be authentic" (Quoted from **RFC 4035**) DNS data or responses deemed to be authentic or validated have a security status of "secure" (**RFC 4035 Section 4.3**; **RFC 4033 Section 5**). "Authenticating both DNS keys and data is a matter of local policy, which may extend or even override the [DNSSEC] protocol extensions…" (Quoted from **RFC 4033 Section 3.1**).

The term "verification", when used, is usually a synonym for "validation".

Zone
: "Authoritative information is organized into units called ZONEs, and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone." (Quoted from **RFC 1034 Section 2.4**)

Zone enumeration
: "The practice of discovering the full content of a zone via successive queries." (Quoted from **RFC 5155 Section 1.3**) This is also sometimes called "zone walking". Zone enumeration is different from zone content guessing where the guesser uses a large dictionary of possible labels and sends successive queries for them, or matches the contents of NSEC3 records against such a dictionary.

Zone signing key (ZSK)
: "DNSSEC keys that can be used to sign all the RRsets in a zone that require signatures, other than the apex DNSKEY RRset." (Quoted from **RFC 6781 Section 3.1**) Also note that a ZSK is sometimes used to sign the apex DNSKEY RRset.

Zone transfer
: The act of a client requesting a copy of a zone and an authoritative server sending the needed information. There are two common standard ways to do zone transfers: the AXFR ("Authoritative Transfer") mechanism to copy the full zone (described in **RFC 5936**, and the IXFR ("Incremental Transfer") mechanism to copy only parts of the zone that have changed (described in **RFC 1995**). Many systems use non-standard methods for zone transfers outside the DNS protocol.

1.20 Cascade CLI

1.20.1 Synopsis

cascade `[GLOBAL OPTIONS]` `<COMMAND>`

1.20.2 Description

cascade is the CLI to the *Cascade Daemon*.

1.20.3 Global Options

-s, --server <IP:PORT>
: The cascade server instance to connect to [default: 127.0.0.1:4539].

--log-level <LEVEL>
: The minimum severity of messages to log [default: warning] [possible values: trace, debug, info, warning, error, critical].

-h, --help
: Print the help text (short summary with -h, long help with --help).

-V, --version
: Print version.

1.20.4 Commands

***cascade-health*(1)**
: Check the health of Cascade.

***cascade-zone*(1)**
: Manage zones.

***cascade-policy*(1)**
: Manage policies.

***cascade-keyset*(1)**
: Execute manual key roll or key removal commands.

***cascade-tsig*(1)**
: Manage TSIG keys.

***cascade-hsm*(1)**
: Manage HSMs.

***cascade-debug*(1)**
: Debug / troubleshoot Cascade.

***cascade-template*(1)**
: Print example config or policy files.

1.20.5 See Also

https://cascade.docs.nlnetlabs.nl
: Cascade online documentation

cascaded(1)
: *Cascade Daemon*

cascaded-config.toml(5)
: *Configuration File Format*

cascaded-policy.toml(5)
: *Policy File Format*

1.21 Cascade Daemon

1.21.1 Synopsis

cascaded [OPTIONS]

1.21.2 Description

cascaded is the daemon process of Cascade, a friendly DNSSEC signing solution.

For more information about Cascade, please refer to the Cascade documentation at https://cascade.docs.nlnetlabs.nl.

1.21.3 Options

--check-config
: Check the configuration and exit with code 0 if the configuration is valid, or code 1 if the configuration is invalid.

--state <PATH>
: The global state file to use.

-c, --config <PATH>
: The configuration file to load. Defaults to `/etc/cascade/config.toml`.

--log-level <LEVEL>
: The minimum severity of messages to log [possible values: trace, debug, info, warning, error, critical].

  Defaults to `info`, unless set in the config file.

-l, --log <TARGET>
: Where logs should be written to [possible values: stdout, stderr, file:<PATH>, syslog].

Changed in version 0.1.0-alpha2: Added types `stdout` and `stderr`. Type `file` with values `/dev/stdout` and `/dev/stderr` can still be used but may not work properly in some cases, e.g. when running under systemd.

-d, --daemonize
: Whether Cascade should fork on startup. This option changes the working directory to the root directory and as such influences where files are looked for. Use absolute path names in configuration to avoid ambiguities.

-h, --help
: Print the help text (short summary with `-h`, long help with `--help`).

-V, --version
: Print version.

1.21.4 Files

/etc/cascade/config.toml
: Default Cascade config file

/etc/cascade/policies
: Default policies directory

/var/lib/cascade/zone-state
: Default zone state directory

/var/lib/cascade/tsig-keys.db
: Default file for stored TSIG keys

/var/lib/cascade/keys
Default directory for on-disk zone keys

/var/lib/cascade/kmip/credentials.db
Default file for KMIP credentials

/var/lib/cascade/kmip
Default directory for KMIP state files

1.21.5 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

1.22 Configuration File Format

Cascade uses the TOML format for its configuration file. A template can be generated using `cascade template config`. The provided values to the options below are the default values and are serving as a hint to the option's format.

Note

All changes to the configuration file require running `cascade config reload` for them to take effect. Currently, most options additionally require a restart of the server.

1.22.1 Example

```
version = "v1"
policy-dir = "/etc/cascade/policies"
zone-state-dir = "/var/lib/cascade/zone-state"
tsig-store-path = "/var/lib/cascade/tsig-keys.db"
kmip-credentials-store-path = "/var/lib/cascade/kmip/credentials.db"
keys-dir = "/var/lib/cascade/keys"
kmip-server-state-dir = "/var/lib/cascade/kmip"
dnst-binary-path = "dnst"

[daemon]
log-level = "info"
log-target = { type = "syslog" }
daemonize = true
pid-file = "/var/run/cascade.pid"
identity = "cascade:cascade"

[remote-control]
servers = ["127.0.0.1:4539", "[::1]:4539"]
```

(continues on next page)

(continued from previous page)

```
[loader]

[loader.review]
servers = ["127.0.0.1:4540", "[::1]:4540"]

[signer]
[signer.review]
servers = ["127.0.0.1:4541", "[::1]:4541"]

[key-manager]

[server]
servers = ["127.0.0.1:4542", "[::1]:4542"]
```

1.22.2 Options

Global Options

version = "v1"

The configuration file version. (REQUIRED)

This is the only required option. All other settings, and their defaults, are associated with this version number. More versions may be added in the future and Cascade may drop support for older versions over time.

- v1: This format.

policy-dir = "/etc/cascade/policies"

The directory storing zone policies.

Zone policies are user-managed files configuring groups of zones. You can modify them as you like, then ask Cascade to reload them with `cascade policy reload`.

zone-state-dir = "/var/lib/cascade/zone-state"

The directory storing per-zone state files.

Cascade maintains an internal state file for every known zone here. These files should not be modified manually, but they can be backed up and restored in the event of filesystem corruption.

tsig-store-path = "/var/lib/cascade/tsig-keys.db"

The file storing TSIG key secrets.

This is an internal state file containing sensitive cryptographic material. It should not be modified manually, but it can be backed up and restored in the event of filesystem corruption. Carefully consider its security.

Note: This setting is not used at present as the alpha version of Cascade does not yet support TSIG keys.

kmip-credentials-store-path = "/var/lib/cascade/kmip/credentials.db"

The file storing KMIP credentials.

This is an internal state file containing sensitive KMIP server login credentials. It should not be modified manually, but it can be backed up and restored in the event of filesystem corruption. Carefully consider its security.

keys-dir = "/var/lib/cascade/keys"

The directory storing rollover states and on-disk DNSSEC keys.

For every zone, the state of its DNSSEC keys (which keys are used, on-going rollovers, etc.) are stored here. If on-disk keys are used to sign zones, they are stored also here.

The organization of this directory (file names and file formats) constitutes internal implementation details. It should not be modified manually, but it can be backed up and restored in the event of filesystem corruption. Carefully consider its security.

kmip-server-state-dir = "/var/lib/cascade/kmip"

The directory containing KMIP server state.

Information about known KMIP servers is stored in this directory.

The organization of this directory (file names and file formats) constitutes internal implementation details. It should not be modified manually, but it can be backed up and restored in the event of filesystem corruption.

dnst-binary-path = "dnst"

The path to the dnst binary Cascade should use.

Cascade relies on a separate tool called `dnst` (https://github.com/NLnetLabs/dnst) to perform DNSSEC key management. You can specify an absolute path here, or just `dnst` if it is in $PATH.

Settings relevant to any daemon program.

The `[daemon]` section.

log-level = "info"

The minimum severity of the messages logged by the daemon.

Messages at or above the specified severity level will be logged. The following levels are defined:

- `trace`: A function or variable was interacted with, for debugging.
- `debug`: Something occurred that may be relevant to debugging.
- `info`: Things are proceeding as expected.
- `warning`: Something does not appear to be correct.
- `error`: Something went wrong (but Cascade can recover).
- `critical`: Something went wrong and Cascade can't function at all.

log-target = { type = "stdout" }

log-target = { type = "stderr" }

log-target = { type = "syslog" }

log-target = { type = "file", **path** = "cascaded.log" }

The location the daemon writes logs to.

- type `file`: Logs are appended line-by-line to the specified file path.

  If it is a terminal, ANSI escape codes may be used to style the output.

- type `stdout`: Logs are written to stdout. (The default)

  If it is a terminal, ANSI escape codes may be used to style the output.

- type `stderr`: Logs are written to stderr.

  If it is a terminal, ANSI escape codes may be used to style the output.

- type `syslog`: Logs are written to the UNIX syslog.

  This option is only supported on UNIX systems.

Changed in version 0.1.0-alpha2: Added types `stdout` and `stderr` which should be used instead of `file` values `/dev/stdout` and `/dev/stderr` which do not work properly in some cases, e.g. when running under systemd.

> **Note**
>
> When using systemd, `syslog` and `stdout` are the most reliable options. Systemd environments are often heavily isolated, making file-based logging difficult.

`daemonize` = `false`

Whether to apply internal daemonization.

'Daemonization' involves several steps:

- Forking the process to disconnect it from the terminal
- Tracking the new process' PID (by storing it in a file)
- Binding privileged ports (below 1024) as configured
- Dropping administrator privileges

These features may be provided by an external system service manager, such as systemd. If no such service manager is being used, Cascade can provide such features itself, by setting this option to `true`. This will also enable the `pid-file` and `identity` settings (although they remain optional).

If this option is set to `true`, the server changes its working directory to the root directory and as such influences where files are looked for. Use absolute path names in configuration to avoid ambiguities. Additionally, it will redirect stdout and stderr to `/dev/null` and you need to choose `syslog` or `file` as the *`log-target`*.

`pid-file` = `"/var/run/cascade.pid"`

The path to a PID file to maintain, if any.

If specified, Cascade will maintain a PID file at this location; it will be a simple plain-text file containing the PID number of the daemon process. This option is only supported if `daemonize` is true.

`identity` = `"cascade:cascade"`

An identity (user and group) to assume after startup.

Cascade will assume the specified identity after initialization. Note that this will fail if Cascade is started without administrator privileges. This option is only supported if `daemonize` is `true`.

The identity can be specified as `<user>:<group>` or just `<user>`; in the latter case, the identically named group will be used. Numeric IDs are not supported; only names can be used.

> **Note**
>
> When using systemd, you should rely on its 'User=' and 'Group=' options instead. See <https://www.freedesktop.org/software/systemd/man/latest/systemd.exec.html#User=>.

How Cascade is controlled.

The `[remote-control]` section.

`servers` = `["127.0.0.1:4539", "[::1]:4539"]`

Where to serve Cascade's HTTP API.

The HTTP API can be used to monitor and control Cascade. The addresses refer to TCP sockets that will be listened on for HTTP requests. At the moment, security mechanisms like TLS are not supported.

These sockets may be bound by systemd and passed into Cascade. If systemd does not provide them, Cascade will bind them itself (and will do so before dropping privileges, if that is enabled).

How zones are loaded.

The `[loader]` section. (This only includes the `[loader.review]` section below, for now).

How loaded zones are reviewed.

The `[loader.review]` section.

`servers` = `["127.0.0.1:4540", "[::1]:4540"]`

Where to serve loaded zones for review.

A DNS server will be bound to these addresses, and will serve the contents of all loaded zones. This can be used to verify the consistency of these zones.

Unless explicitly specified (e.g. `udp://localhost:4540`), each address will be served over UDP and TCP. An empty array will disable serving entirely.

These sockets may be bound by systemd and passed into Cascade. If systemd does not provide them, Cascade will bind them itself (and will do so before dropping privileges, if that is enabled).

How zones are signed.

The `[signer]` section. (This only includes the `[signer.review]` section below, for now).

How signed zones are reviewed.

The `[signer.review]` section.

`servers` = `["127.0.0.1:4541", "[::1]:4541"]`

Where to serve signed zones for review.

A DNS server will be bound to these addresses, and will serve the contents of all signed (but not necessarily published) zones. This can be used to check the correctness of the signer.

Unless explicitly specified (e.g. `udp://localhost:4541`), each address will be served over UDP and TCP. An empty array will disable serving entirely.

These sockets may be bound by systemd and passed into Cascade. If systemd does not provide them, Cascade will bind them itself (and will do so before dropping privileges, if that is enabled).

DNSSEC key management.

The `[key-manager]` section. (Currently without options)

How zones are published.

The `[server]` section.

`servers` = `["127.0.0.1:4542", "[::1]:4542"]`

Where to serve published zones.

A DNS server will be bound to these addresses, and will serve the contents of all published zones. This is the final output from Cascade.

Unless explicitly specified (e.g. `udp://localhost:4542`), each address will be served over UDP and TCP. At least one address must be specified.

These sockets may be bound by systemd and passed into Cascade. If systemd does not provide them, Cascade will bind them itself (and will do so before dropping privileges, if that is enabled).

1.22.3 Files

/etc/cascade/config.toml
Default Cascade config file

1.22.4 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

1.23 Policy File Format

A policy is a collection of settings that apply to a group of zones known to Cascade. Policy controls how Cascade operates on those zones, e.g. how they are signed. This page describes all possible settings and their defaults. You can generate a template with these default values using `cascade template policy`.

Policy files are managed by the user, and are stored at a configurable path (by default, `/etc/cascade/policies`). You can add, modify, and remove policy files, then update Cascade with `cascade policy reload`. Note that:

- Cascade maintains an internal copy of all policies, and will use this until `cascade policy reload` is used. If reloading fails, Cascade will continue to use its existing internal copy. It won't reload policies if it restarts.
- Policies cannot be removed if they are attached to zones; those zones need to be deleted or shifted to a different policy first. If you remove a used policy and reload policies in Cascade, it will fail and continue to use its internal copy of the policy.
- Only policy files stored in the configured policy directory and having a `.toml` extension will be loaded by Cascade.`

> **Note**
>
> In the current alpha release, changes to some policy options (e.g. review hook) also require a server restart in addition to running `cascade policy reload` to take effect.

1.23.1 Example

```
version = "v1"

[loader]
[loader.review]
mode = "off"

[key-manager]
ksk.validity = "365d"
zsk.validity = "30d"
```

(continues on next page)

(continued from previous page)

```
csk.validity = "365d"
ksk.auto-start = true
zsk.auto-start = true
csk.auto-start = true
algorithm.auto-start = true
ksk.auto-report = true
zsk.auto-report = true
csk.auto-report = true
algorithm.auto-report = true
ksk.auto-expire = true
zsk.auto-expire = true
csk.auto-expire = true
algorithm.auto-expire = true
ksk.auto-done = true
zsk.auto-done = true
csk.auto-done = true
algorithm.auto-done = true
ds-algorithm = "SHA256"
auto-remove = true
auto-remove-delay = "7d"
publication-nameservers = []

[key-manager.records]
ttl = "1h"
dnskey.signature-inception-offset = "1d"
cds.signature-inception-offset = "1d"
dnskey.signature-lifetime = "2w"
cds.signature-lifetime = "2w"
dnskey.signature-remain-time = "1w"
cds.signature-remain-time = "1w"

[key-manager.generation]
use-csk = false
algorithm = "ECDSAP256SHA256"

[signer]
serial-policy = "date-counter"
signature-inception-offset = "1d"
signature-lifetime = "2w"
signature-remain-time = "1w"
signature-refresh-interval = "12h"
key-roll-time = "24h"

[signer.denial]
type = "nsec"

[signer.review]
mode = "off"

[server.outbound]
send-notify-to = []
```

1.23.2 Options

Global Options

version = "v1"

The policy file version. (REQUIRED)

This is the only required option. All other settings, and their defaults, are associated with this version number. More versions may be added in the future and Cascade may drop support for older versions over time.

- v1: This format.

How zones are loaded.

The [loader] section.

How loaded zones are reviewed.

The [loader.review] section.

Review offers an opportunity to perform external checks on the zone contents loaded by Cascade.

mode = "off"

The mode for loader review.

This can be one of the following values:

- "off" will disable review
- "manual" will enable manual review via the CLI.
- "script" will enable automatic review via a hook. The hook field must be specified in this case.

The default value is "off".

hook = ""

A hook for reviewing a loaded zone. This is a path to an executable.

This command string will be executed in the user's shell when a new version of a zone is loaded. At the moment, it will only be run if required is true.

It will receive the following information via environment variables:

- CASCADE_ZONE: The name of the zone, formatted without a trailing dot.
- CASCADE_SERIAL: The serial number of the zone (decimal integer).
- CASCADE_SERVER: The combined address and port where Cascade is serving the zone for review, formatted as <ip-addr>:<port>.
- CASCADE_SERVER_IP: Just the address of the above server.
- CASCADE_SERVER_PORT: Just the port of the above server.

Added in version 0.1.0-alpha2: CASCADE_SERVER_IP and CASCADE_SERVER_PORT.

The command will be called from an unspecified directory, and it must be accessible to Cascade (i.e. after it has dropped privileges). Its exit code will determine whether the zone is approved or not.

on-reject = "discard"

What to do when a zone is rejected by review.

This field can only be specified if mode is either "manual" or "script" and can have one of the following values:

- "discard" will discard the rejected zone and go back to an idle state
- "halt" will halt the pipeline until an operator either resets the pipeline or overrides the rejection.

The default value is "discard".

DNSSEC key management.

The [key-manager] section.

ksk.validity = "365d"

zsk.validity = "30d"

csk.validity = "365d"

How long keys are considered valid for.

If a key has been used for longer than this time, it is considered expired, and (if enabled) it will automatically be rolled over to a new key. Even if automatic rollovers are not enabled, the key will be reported as expired. This is a soft condition – DNSSEC does not have a concept of key expiry, and it will not break DNSSEC validation, but it is usually important to the security of the zone.

Independent validity times are set for KSKs, ZSKs, and CSKs. An integer value will be interpreted as seconds, "forever" means keys never expire, and a time string such as "365d" will be interpreted as 365 days. Supported suffixes include s, m, h, d and w.

ksk.auto-start = true

zsk.auto-start = true

csk.auto-start = true

algorithm.auto-start = true

Whether to automatically start key rollovers.

If this is enabled, Cascade will automatically start rolling over keys when they expire (as per validity). When using this setting, validity must not be set to "forever".

The first step in a rollover will be to generate new keys to replace old ones. By disabling this setting, the user can manually control how new keys are generated, and when rollovers happen.

ksk.auto-report = true

zsk.auto-report = true

csk.auto-report = true

algorithm.auto-report = true

Whether to automatically check for record propagation.

If this is enabled, Cascade will automatically contact public DNS servers to detect when new records (e.g. DNSKEY) are visible globally. It is necessary to wait until some records are visible globally to progress key rollovers. If this is disabled, the user will have to inform Cascade when these conditions are reached manually.

For this setting to work, Cascade must have network access to the zone's public nameservers and the parent zone's public nameservers.

ksk.auto-expire = true

zsk.auto-expire = true

csk.auto-expire = true

algorithm.auto-expire = true

Whether to automatically wait for cache expiry.

If this is enabled, Cascade will automatically progress through key rollover steps that need to wait for downstream users' DNS caches to expire. It will estimate the right amount of time to wait based on record TTLs.

ksk.auto-done = true

zsk.auto-done = true

csk.auto-done = true

algorithm.auto-done = true

Whether to automatically check for rollover completion.

Like `auto-report`, if this setting is enabled, Cascade will automatically contact public DNS servers to detect when new records are visible globally. `auto-done` specifically affects automatic checks for the last step of key rollovers, and is independent from `auto-report`.

For this setting to work, Cascade must have network access to the zone's public nameservers and the parent zone's public nameservers.

ds-algorithm = "SHA-256"

The hash algorithm used by the parent zones' DS records.

Supported options:

- `SHA-256`: SHA-256.
- `SHA-384`: SHA-384.

auto-remove = true

Whether to automatically remove expired keys.

If this option is set, expired keys will be removed automatically (by deleting the files for on-disk keys or removing it from the HSM).

auto-remove-delay = "7d"

Delay after which expired keys will be removed when auto-remove is true.

An integer value is interpreted as seconds. A string is interpreted as time string with a number followed by a unit (i.e. "s", "m", "h", "d", or "w").

publication-nameservers = []

The set of nameservers to use when checking for RRSIG propagation during a key roll.

Each nameserver must be specified as a string in the form:

```
"<IP>[:<PORT>][^<TSIG_KEY_NAME>]"
```

If a TSIG key name is specified, a key by that name must exist in the Cascade TSIG key store and will be used to authenticate communication with the nameserver.

If no nameservers are specified, the nameserver specified by the SOA MNAME field will be checked.

The management of DNS records by the key manager.

The `[key-manager.records]` section.

The key manager generates and signs several records (DNSKEY and CDS). This section controls its behaviour towards them.

ttl = "1h"

The TTL for the generated records.

dnskey.signature-inception-offset = "1d"

cds.signature-inception-offset = "1d"

The offset for generated signature inceptions.

Record signatures have a fixed inception time, from when they are considered valid. An imprecise computer clock could cause signatures to be considered invalid, because their inception point appears to be some time in the future. To prevent such cases, this setting allows the inception time to be offset into the past.

Independent offsets can be set for each type of record. An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`). Inception times will be calculated as `now - offset` at the time of signing.

dnskey.signature-lifetime = "2w"

cds.signature-lifetime = "2w"

The lifetime of generated signatures.

Record signatures have a fixed lifetime, after which they are considered invalid. To keep the zone valid, the signatures should be regenerated before they expire; see `signature-remain-time` to control regeneration time.

Independent lifetimes can be set for each type of record. An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

dnskey.signature-remain-time = "1w"

cds.signature-remain-time = "1w"

The amount of time remaining before expiry when signatures will be regenerated.

In order to prevent a zone's signatures from appearing invalid, they have to be regenerated before they expire. That hard limit is set by `signature-lifetime` above. This setting controls how long before expiry signatures will be regenerated; it must be less than the `signature-lifetime` setting.

Independent waiting times can be set for each type of record. An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

How keys are generated.

The `[key-manager.generation]` section.

hsm-server-id = ""

The HSM server to use.

If this is set, the named HSM server (which must be configured via `cascade hsm add`) will be used for generating new DNSSEC keys.

See https://cascade.docs.nlnetlabs.nl/en/latest/hsms.html for more information.

use-csk = false

Whether to generate CSKs, instead of separate ZSKs and KSKs.

A CSK (Combined Signing Key) takes the role of both ZSK and KSK for a zone, unlike the standard practice of using separate keys for ZSK and KSK. This setting does not affect how DNSSEC validation is performed, only procedures for key rollovers.

If this is enabled, Cascade will generate CSKs for the associated zones.

algorithm = "ECDSAP256SHA256"

The cryptographic algorithm (and parameters) for generated keys.

DNSSEC supports various cryptographic algorithms for signatures; one must be selected, and for some algorithms, additional parameters are also necessary. The same algorithm and parameters will be applied to the ZSK and KSK.

- `RSASHA256[:<bits>]`, algorithm 8, with a public key size of `<bits>` (default 2048) bits.
- `RSASHA512[:<bits>]`, algorithm 10, with a public key size of `<bits>` (default 2048) bits.
- `ECDSAP256SHA256`, algorithm 13.
- `ECDSAP384SHA384`, algorithm 14.
- `ED25519`, algorithm 15.
- `ED448`, algorithm 16.

There are additional algorithms, but many are now considered insecure, and it is recommended or mandated to avoid them. In addition, RSA keys smaller than 2048 bits are not recommended.

> **Note**
>
> At the moment, only RSASHA256 and ECDSAP256SHA256 work with HSMs. Other algorithms cannot be used with HSMs, and will cause generation failures.

How zones are signed.

The `[signer]` section.

Note that certain records (e.g. DNSKEY and CDS records at the apex of the zone) are signed by the key manager, rather than the zone signer; see the `[key-manager.records]` section for configuring the signing of those records.

serial-policy = "date-counter"

How SOA serial numbers are generated for signed zones.

Supported options:

- `keep`: use the same serial number as the unsigned zone.
- `counter`: increment the serial number every time.
- `unix-time`: use the current Unix time, in seconds.
- `date-counter`: format the number as `<YYYY><MM><DD><xx>` in decimal. `<xx>` is a simple counter to allow up to 100 versions per day.

signature-inception-offset = "1d"

The offset for generated signature inceptions.

Record signatures have a fixed inception time, from when they are considered valid. An imprecise computer clock could cause signatures to be considered invalid, because their inception point appears to be some time in the future. To prevent such cases, this setting allows the inception time to be offset into the past.

An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`). Inception times will be calculated as `now - offset` at the time of signing.

signature-lifetime = "2w"

The lifetime of generated signatures.

Record signatures have a fixed lifetime, after which they are considered invalid. To keep the zone valid, the signatures should be regenerated before they expire; see `signature-remain-time` to control regeneration time.

An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

signature-remain-time = "1w"

The amount of time remaining before expiry when signatures will be regenerated.

In order to prevent a zone's signatures from appearing invalid, they have to be regenerated before they expire. That hard limit is set by `signature-lifetime` above. This setting controls how long before expiry signatures will be regenerated; it must be less than the `signature-lifetime` setting.

An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

signature-refresh-interval = "12h"

Refresh period to prevent signatures from expiring. Each period, Cascade will refresh some number of signatures. This way the work to refresh all signatures is spread out over time. The effective lifetime of a signature is signature-lifetime - signature-remain-time. Each period roughly a fraction of all signatures that is equal to signature-refresh-interval divided by the effective signature lifetime will be refreshed.

signature-refresh-interval should be a lot smaller than signature-remain-time to make sure that signatures are refreshed in time. If this is not the case then in extreme cases, signatures could expire.

An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

key-roll-time = "24h"

To avoid resigning the entire zone at once during a ZSK or CSK roll, generating signatures with the new key can be spread out over time. New signatures are generated at intervals controlled by signature-refresh-interval.

An integer value is interpreted as seconds. A string is interpreted as a time string consisting of a number followed by a unit (i.e. `s`, `m`, `h`, `d`, or `w`).

How denial-of-existence records are generated.

The `[signer.denial]` section.

type = "nsec"

The type of denial-of-existence records to generate.

Supported options:

- `nsec`: Use NSEC records (**RFC 4034**).
- `nsec3`: Use NSEC3 records (**RFC 5155**).

opt-out = false

(Only set when using NSEC3)

Whether to skip NSEC3 records for unsigned delegations.

This enables the NSEC3 Opt-Out flag, and skips delegations to unsigned zones when generating NSEC3 records. This affects the security of the zone, so be careful if you wish to enable it.

How signed zones are reviewed.

The `[signer.review]` section.

`[signer.review]`

How signed zones are reviewed.

`required` = `false`

Whether review is required.

If this is `true`, a signed version of a zone will not be published until it is approved. If it is `false`, signed zones will be published immediately. At the moment, the review hook will only be run if this is set to true.

`cmd-hook` = `""`

A hook for reviewing a signed zone. This is a path to an executable.

This command string will be executed in the user's shell when a new version of a zone is signed. At the moment, it will only be run if `required` is true.

It will receive the following information via environment variables:

- `CASCADE_ZONE`: The name of the zone, formatted without a trailing dot.
- `CASCADE_SERIAL`: The serial number of the signed zone (decimal integer).
- `CASCADE_SERVER`: The combined address and port where Cascade is serving the zone for review, formatted as `<ip-addr>:<port>`.
- `CASCADE_SERVER_IP`: Just the address of the above server.
- `CASCADE_SERVER_PORT`: Just the port of the above server.

The command will be called from an unspecified directory, and it must be accessible to Cascade (i.e. after it has dropped privileges). Its exit code will determine whether the zone is approved or not.

How published zones are served.

The `[server.outbound]` section.

`send-notify-to` = `[]`

The set of nameservers to which NOTIFY messages should be sent.

If no nameservers are specified, no NOTIFY messages will be sent.

Each nameserver must be specified as a string in the form:

"<IP>[:<PORT>][^<TSIG_KEY_NAME>]"

If a TSIG key name is specified, a key by that name must exist in the Cascade TSIG key store and will be used to authenticate communication with the nameserver.

`provide-xfr-to` = `[]`

The set of nameservers to provide zone transfers to.

If no nameservers are specified, zone transfers will be provided to any nameserver.

Each nameserver must be specified as a string in the form:

"<IP>[^<TSIG_KEY_NAME>]"

1.23.3 Files

/etc/cascade/config.toml
Default Cascade config file

/etc/cascade/policies
Default policies directory

1.23.4 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

1.24 cascade debug

Added in version 0.1.0-beta1.

1.24.1 Synopsis

cascade [GLOBAL OPTIONS] debug <COMMAND>

cascade [GLOBAL OPTIONS] debug *change-logging* [OPTIONS]

1.24.2 Description

Debug / troubleshoot Cascade. The sub-commands here are tools for analyzing Cascade when lower-level problems occur. It should be combined with analysis of Cascade's log files.

1.24.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.24.4 Commands

change-logging

Change how Cascade logs information.

The location where logs are written to cannot be changed; but the information being logged can be changed.

1.24.5 Options for debug change-logging

-l, --level <LEVEL>

Change the log level. Possible values: trace, debug, info, warning, error, critical

--trace-targets <TARGETS>

Select internal Cascade modules to selectively log trace-level information for. The names of such modules can be found in the log files. All other modules will continue using the log level.

-h, --help
Print the help text (short summary with -h, long help with --help).

1.24.6 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

1.25 cascade health

Added in version 0.1.0-alpha2.

1.25.1 Synopsis

cascade [GLOBAL OPTIONS] health

1.25.2 Description

Check the health of Cascade.

Exits with code zero if Cascade is healthy, non-zero otherwise.

1.25.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.25.4 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

1.26 cascade hsm

1.26.1 Synopsis

cascade [GLOBAL OPTIONS] hsm <COMMAND>

cascade [GLOBAL OPTIONS] hsm *add* <SERVER_ID> <IP_HOST_OR_FQDN>

cascade [GLOBAL OPTIONS] hsm *show* <SERVER_ID>

cascade [GLOBAL OPTIONS] hsm *list*

1.26.2 Description

Manage the configuration of Hardware Security Modules (HSMs) in Cascade

1.26.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.26.4 Commands

add

Add a KMIP server to use for key generation & signing.

Note: There are no commands to remove or modify KMIP servers yet.

show

Get the details of an existing KMIP server.

list

List all configured KMIP servers.

1.26.5 Arguments for `hsm show`

<SERVER_ID>

The identifier of the KMIP server to show information about.

1.26.6 `hsm add`

Add a KMIP server to use for key generation & signing instead of using Ring/OpenSSL based key generation.

1.26.7 Arguments for `hsm add`

<SERVER_ID>

An identifier to refer to the KMIP server by.

This identifier is used with other `cascade` commands and Cascade policy files. The identifier serves several purposes:

1. To make it easy at a glance to recognize which KMIP server a given key was created on, by allowing operators to assign a meaningful name to the server instead of whatever identity strings the server associates with itself or by using hostnames or IP addresses as identifiers.

2. To refer to additional configuration elsewhere to avoid including sensitive and/or verbose KMIP server credential or TLS client certificate/key authentication data in each key identifier, and which would be repeated in every key created on the same server.

3. To allow the actual location of the server and/or its access credentials to be rotated without affecting key idenifiers, e.g. if a server is assigned a new IP address or if access credentials change.

<IP_HOST_OR_FQDN>

The hostname or IP address of the KMIP server.

1.26.8 Options for `hsm add`

-h, --help

Print the help text (short summary with `-h`, long help with `--help`).

Server:

--port <PORT>

TCP port to connect to the KMIP server on.

[default: 5696]

Client Credentials:

--username <USERNAME>

Optional username to authenticate to the KMIP server as.

Note: When using the Cascade `cascade-hsm-bridge` tool the username set here will be used as the label of the PKCS#11 token to login to.

--password <PASSWORD>

Optional password to authenticate to the KMIP server with.

Note: When using the Cascdee `cascade-hsm-bridge` tool the password set here will be used as the PKCS#11 PIN to login with.

Client Certificate Authentication:

--client-cert <CLIENT_CERT_PATH>

Optional path to a TLS certificate to authenticate to the KMIP server with. The file will be read and sent to the server.

--client-key <CLIENT_KEY_PATH>

Optional path to a private key for client certificate authentication. THe file will be read and sent to the server.

The private key is needed to be able to prove to the KMIP server that you are the owner of the provided TLS client certificate.

Server Certificate Verification:

--insecure

Whether to accept the KMIP server TLS certificate without verifying it.

Use this option when your KMIP server uses a self-signed TLS certificate, e.g. in a test environment.

--server-cert <SERVER_CERT_PATH>

Optional path to a TLS PEM certificate for the server.

--ca-cert <CA_CERT_PATH>

Optional path to a TLS PEM certificate for a Certificate Authority.

Client Limits:

--connect-timeout <CONNECT_TIMEOUT>

TCP connect timeout.

[default: 3s]

--read-timeout <READ_TIMEOUT>

TCP response read timeout.

[default: 30s]

--write-timeout <WRITE_TIMEOUT>

TCP request write timeout.

[default: 3s]

--max-response-bytes <MAX_RESPONSE_BYTES>

Maximum KMIP response size to accept (in bytes).

[default: 8192]

Key Labels:

--key-label-prefix <KEY_LABEL_PREFIX>

Optional user supplied key label prefix.

Can be used to denote the s/w that created the key, and/or to indicate which installation/environment it belongs to, e.g. dev, test, prod, etc.

--key-label-max-bytes <KEY_LABEL_MAX_BYTES>

Maximum label length (in bytes) permitted by the HSM. Key labels longer than this will be truncated to fit.

[default: 32]

1.26.9 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascade-hsm-bridge(1)
KMIP to PKCS#11 bridge documentation

1.27 cascade keyset

1.27.1 Synopsis

cascade [GLOBAL OPTIONS] keyset <ZONE> ksk|zsk|csk|algorithm [OPTIONS] <COMMAND>

cascade [GLOBAL OPTIONS] keyset <ZONE> *remove-key* [OPTIONS] <KEY>

cascade [GLOBAL OPTIONS] keyset <ZONE> *get* [RR]

1.27.2 Description

Execute manual key roll or key removal commands.

1.27.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.27.4 Commands

ksk

Command for KSK rolls.

zsk

Command for ZSK rolls.

csk

Command for CSK rolls.

algorithm

Command for algorithm rolls.

remove-key

Remove a key from the key set.

get

Get the key or keys for a zone as DS, DNSKEY, or CDS RRsets.

1.27.5 Key roll commands for `ksk|zsk|csk|algorithm`

start-roll

Start a key roll.

propagation1-complete <TTL>

Inform keyset that the changed RRsets and signatures have propagated.

TTL is the maximum TTL of the zone.

cache-expired1

Inform keyset that enough time has passed that caches should have expired.

propagation2-complete <TTL>

Inform keyset that the changed RRsets and signatures have propagated.

TTL is the maximum TTL of the zone.

cache-expired2

Inform keyset that enough time has passed that caches should have expired.

roll-done

Report that the final changes have propagated and the roll is done

1.27.6 Arguments for `keyset remove-key`

<KEY>

The key to remove. This is the key's URI as reported by `cascade zone status`.

1.27.7 Options for `keyset remove-key`

`--force`
Force a key to be removed even if the key is not stale.

`--continue`
Continue when removing the underlying keys fails.

1.27.8 Arguments for `keyset get`

Added in version 0.1.0-beta1.

`[RR]`
The RRset to print. `ds`, `dnskey`, or `cds`.

The CDS RRset includes the CDNSKEY RRset and signatures.

> **Note**
>
> The DS and CDS RRset is only available during the appropriate step of a key roll. So, if the output is empty, check the zone's key roll status to see if it may still be waiting for propagation of e.g. the new DNSKEY. If you need the DS RRset even if cascade is still waiting for propagation, you can use `cascade keyset <zone> get dnskey | dnst key2ds -n /dev/stdin`.

1.27.9 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

dnst-keyset(1)
Further documentation of the key roll commands (and more)

1.28 cascade policy

1.28.1 Synopsis

`cascade` `[GLOBAL OPTIONS] policy <COMMAND>`

`cascade` `[GLOBAL OPTIONS] policy list`

`cascade` `[GLOBAL OPTIONS] policy show <NAME>`

`cascade` `[GLOBAL OPTIONS] policy reload`

1.28.2 Description

Manage Cascade's policies.

1.28.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.28.4 Commands

list
: List registered policies.

show
: Show the settings contained in a policy.

reload
: Reload all the policies from the files.

1.28.5 See Also

https://cascade.docs.nlnetlabs.nl
: Cascade online documentation

cascade(1)
: *Cascade CLI*

cascaded(1)
: *Cascade Daemon*

cascaded-config.toml(5)
: *Configuration File Format*

cascaded-policy.toml(5)
: *Policy File Format*

1.29 cascade status

Added in version 0.1.0-alpha2.

1.29.1 Synopsis

cascade [GLOBAL OPTIONS] status

1.29.2 Description

Displays an at-a-glance status report for Cascade indicating what it is currently doing and noting any issues that require operator action.

1.29.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.29.4 See Also

https://cascade.docs.nlnetlabs.nl
: Cascade online documentation

cascade(1)
: *Cascade CLI*

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

1.30 cascade template

1.30.1 Synopsis

cascade [GLOBAL OPTIONS] template <COMMAND>

1.30.2 Description

Print example config or policy files.

1.30.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.30.4 Commands

config
Generate a config template.

policy
Generate a policy template.

1.30.5 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

1.31 cascade tsig

Added in version 0.1.0-beta1.

1.31.1 Synopsis

cascade [GLOBAL OPTIONS] tsig <COMMAND>

cascade [GLOBAL OPTIONS] tsig *add* <TSIG_KEY_NAME> <ALGORITHM> <SECRET>

cascade [GLOBAL OPTIONS] tsig *list*

cascade [GLOBAL OPTIONS] tsig *remove* <TSIG_KEY_NAME>

1.31.2 Description

Manage **RFC 8945** (TSIG) keys for authenticating zone transfer (AXFR, IXFR) and related messages (SOA and NOTIFY).

> **Tip**
>
> Cascade isn't currently able to generate TSIG keys itself. One way to generate a TSIG key is to use the tsig-keygen tool from the ISC BIND project.

1.31.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.31.4 Commands

add

Add a new TSIG key.

Incoming DNS messages that are TSIG signed will be rejected if the key used to sign the message is not registered with Cascade.

list

List registered TSIG keys.

remove

Remove a TSIG key.

> **Note**
>
> Returns an error if the key does not exist in the TSIG key store, or if the key is still referenced by other configuration.

1.31.5 Arguments for `tsig add`

<TSIG_KEY_NAME>

[<ALGORITHM>]:<TSIG_KEY_NAME>:<SECRET>

The name of the TSIG key to add, or a complete TSIG key specification.

TSIG key names must be valid domain names.

A complete TSIG key specification consists of an optional algorithm (default `hmac-sha256`), a key name and the secret key material. When a complete TSIG key specification is supplied, supplying the <ALGORITHM> and <SECRET> arguments as well will result in an error.

Secret key material must be the correct length for the specified algorithm and must be encoded using the RFC 4648 Base64 encoding.

> **Warning**
>
> Secret key material supplied via a command-line argument may be visible to other processes running on the same computer as the Cascade CLI.

<ALGORITHM>

The TSIG algorithm of the specified TSIG key. Can be one of: `hmac-sha1`, `hmac-sha256`, `hmac-sha384` or `hmac-sha512`.

<SECRET>

RFC 4648 Base64 encoded secret key material. The number of bytes prior to encoding must be correct for the specified <ALGORITHM>.

Can also be a path to a file containing the Base64 encoded secret material.

> **Note**
>
> Secret key material supplied via a command-line argument may be visible to other processes running on the same computer as the Cascade CLI. Consider supplying a file name instead.

1.31.6 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

1.32 cascade zone

1.32.1 Synopsis

cascade [GLOBAL OPTIONS] zone <COMMAND>

cascade [GLOBAL OPTIONS] zone *add* [OPTIONS] --source <SOURCE> --policy <POLICY> <NAME>

cascade [GLOBAL OPTIONS] zone *remove* <NAME>

cascade [GLOBAL OPTIONS] zone *list*

cascade [GLOBAL OPTIONS] zone *reload* <NAME>

cascade [GLOBAL OPTIONS] zone *approve* <--unsigned|--signed> <NAME> <SERIAL>

cascade [GLOBAL OPTIONS] zone *reject* <--unsigned|--signed> <NAME> <SERIAL>

cascade [GLOBAL OPTIONS] zone *override* <--unsigned|--signed> <NAME>

cascade [GLOBAL OPTIONS] zone *status* [--detailed] <NAME>

cascade [GLOBAL OPTIONS] zone *reset* <NAME>

cascade [GLOBAL OPTIONS] zone *history* <NAME>

cascade [GLOBAL OPTIONS] zone maintenance <enable|disable> <NAME>

1.32.2 Description

Manage Cascade's zones.

1.32.3 Global Options

See *Cascade CLI* for information about global options supported by every CLI command.

1.32.4 Commands

add

Add a new zone.

remove

Remove a zone.

> **Note**
>
> Once removed, downstream servers will no longer be able to fetch the zone!

list

List registered zones.

reload

Reload a zone.

approve

Approve a zone being reviewed.

reject

Reject a zone being reviewed.

override

Override a previous rejection of a zone review.

status

Get the status of a single zone.

reset

Reset the pipeline for a zone to get it out of a halted state.

history

Get the history of a single zone.

1.32.5 Options for `zone add`

`--source` `<IP>[:<PORT>][^<TSIG_KEY_NAME>]`

The zone source can be the IP address of an upstream nameserver (with or without port, defaults to port 53) or the path to a zone file locally available to the `cascaded` daemon.

When specifying an upstream nameserver you may also optionally specify the name of an **RFC 8945** TSIG key that should be used to authenticate communication with the upstream.

Zones sourced from an upstream nameserver will be automatically updated if a new version is detected via a SOA query, either based on the zone's SOA record timers, or in response to an **RFC 1996** NOTIFY message from the upstream.

Zones can also be manualy updated via **`cascade`** *`reload`*.

For zones that have already been retrieved at least once via AXFR, subsequent refreshes will attempt to use IXFR and fallback to AXFR if IXFR is not available.

> **Note**
>
> When running **`cascade`** `zone add` from a different host than where the Cascade daemon is running, make sure that the source (whether filesystem path or IP address) is reachable by the Cascade daemon.

> **Note**
>
> If using a TSIG key the key must first be added to Cascade via **`cascade`** `tsig add`.

`--policy` `<POLICY>`

Policy to use for this zone.

Note: At present to use a HSM with a zone the HSM must exist and be configured in the policy used by the zone when the zone is added. It is not possible to change it later in this alpha version of Cascade.

`--import-public-key` `<IMPORT_PUBLIC_KEY>`

Import a public key to be included in the DNSKEY RRset.

This needs to be a file path accessible by the Cascade daemon.

`--import-ksk-file` `<IMPORT_KSK_FILE>`

Import a key pair as a KSK.

The file path needs to be the public key file of the KSK. The private key file name is derived from the public key file. Key files are not actually copied from the specified paths and must remain accessible to the server.

`--import-zsk-file` `<IMPORT_ZSK_FILE>`

Import a key pair as a ZSK.

The file path needs to be the public key file of the ZSK. The private key file name is derived from the public key file. Key files are not actually copied from the specified paths and must remain accessible to the server.

`--import-csk-file` `<IMPORT_CSK_FILE>`

Import a key pair as a CSK.

The file path needs to be the public key file of the CSK. The private key file name is derived from the public key file. Key files are not actually copied from the specified paths and must remain accessible to the server.

--import-ksk-kmip <server> <public_id> <private_id> <algorithm> <flags>
Import a KSK from an HSM.

--import-zsk-kmip <server> <public_id> <private_id> <algorithm> <flags>
Import a ZSK from an HSM.

--import-csk-kmip <server> <public_id> <private_id> <algorithm> <flags>
Import a CSK from an HSM.

-h, --help
Print the help text (short summary with -h, long help with --help).

<NAME>
The name of the zone to add.

1.32.6 Options for zone remove

<NAME>
The name of the zone to remove.

1.32.7 Options for zone reload

<NAME>
The name of the zone to reload.

1.32.8 Options for zone approve

<--unsigned|--signed>
Whether the zone to approve is at the unsigned or signed review stage.

<NAME>
The name of the zone to approve.

<SERIAL>
The serial number of the zone to approve.

1.32.9 Options for zone reject

<--unsigned|--signed>
Whether the zone to reject is at the unsigned or signed review stage.

<NAME>
The name of the zone to reject.

<SERIAL>
The serial number of the zone to reject.

1.32.10 Options for zone override

Added in version 0.1.0-beta1.

<--unsigned|--signed>
Whether the zone to override is at the unsigned or signed review stage.

<NAME>
The name of the zone to override.

1.32.11 Options for `zone status`

`--detailed`

Print detailed information about the zone, including a zone's DNSSEC key identifiers in use, as well as the new DNSKEY records during key rolls.

`<NAME>`

The name of the zone to report the status of.

1.32.12 Options for `zone reset`

Added in version 0.1.0-beta1.

`<NAME>`

The name of the zone to reset the pipeline of.

1.32.13 Options for `zone maintenance`

`<enable|disable>`

Whether maintenance mode should be enabled or disabled.

`<NAME>`

The name of the zone to toggle maintenance mode of.

1.32.14 See Also

https://cascade.docs.nlnetlabs.nl
Cascade online documentation

cascade(1)
Cascade CLI

cascaded(1)
Cascade Daemon

cascaded-config.toml(5)
Configuration File Format

cascaded-policy.toml(5)
Policy File Format

INDEX

Symbols

A

B

C

D

G

H

I

K

L

M

S

T

U

V

Z