

# **OpenDNSSEC Documentation v1.4**

**Released April 2013**

1. OpenDNSSEC Documentation Home	3
1.1 New in OpenDNSSEC 1.4	4
1.2 Getting Started	5
1.3 Overview of OpenDNSSEC	7
1.3.1 Components	9
1.3.2 Files	13
1.3.3 Key States	16
1.3.4 Key Rollovers	17
1.3.5 Quick guides	19
1.4 Installation	22
1.4.1 Dependencies	25
1.4.2 Hardware Security Modules	28
1.4.3 Migrating from earlier versions of OpenDNSSEC	28
1.4.4 Migrating between supported database backends	29
1.5 Configuration	30
1.5.1 Configuration files	31
1.5.1.1 Date Time durations	32
1.5.1.2 conf.xml	33
1.5.1.3 kasp.xml	42
1.5.1.4 zonelist.xml	51
1.5.1.5 addns.xml	54
1.5.1.6 signconf.xml	59
1.5.1.7 Migrating zone fetcher to DNS adapters	65
1.5.2 Zone content	69
1.5.3 Migrating to OpenDNSSEC	70
1.5.4 External Auditing of Zones	73
1.5.5 Plugins	73
1.6 Running OpenDNSSEC	75
1.6.1 Key Management	79
1.6.2 Zone Management	83
1.6.3 Logging	85
1.6.4 High availability	86
1.7 Command Utilities	88
1.7.1 ods-ksmutil	94
1.8 Getting Help	107
1.8.1 Troubleshooting	108
1.8.2 Frequently Asked Questions	114
1.8.3 Getting Support	116
1.8.4 Reporting bugs	117
1.8.4.1 How to get information from a segmentation fault	118
1.9 Reference Material	120
1.9.1 Downloads	120

# OpenDNSSEC Documentation Home

## Welcome

### About

The **OpenDNSSEC documentation** gives information on how to install, configure, and run [OpenDNSSEC](#). There might still remain some questions, so we try to reflect them in our a growing list of [frequently asked questions](#).

Remember that you also need an HSM, which uses the PKCS#11 interface. We do provide the [SoftHSM](#), a software-only implementation of an HSM. Read the [HSM Buyer's Guide](#) for more information and consult the [list of HSM vendors](#).

The latest version of OpenDNSSEC is 1.4

***[See what is new in 1.4](#) - note that due to changes in the database schema [a migration](#) is required when upgrading to 1.4 from earlier versions of OpenDNSSEC.***

### Scope

The goal of OpenDNSSEC is to have a complete DNSSEC zone signing system which maintains stability and security of signed domains. DNSSEC adds many cryptographic concerns to DNS; OpenDNSSEC automates those to allow current DNS administrators to adopt DNSSEC. This document provides DNS administrators with the necessary information to get the system up and running with a basic configuration.

## OpenDNSSEC Documentation

### [Getting Started](#)

### [Overview of OpenDNSSEC](#)

### [Installation](#)

### [Configuration files](#)

### [Running OpenDNSSEC](#)

### [Command Utilities](#)

### [Troubleshooting](#)

### [Reporting bugs](#)

### [Getting Support](#)

## Downloads

A PDF version is available on our [Downloads](#) page, along with instructions on how to export selected pages.


## New in OpenDNSSEC 1.4

- [Major enhancements](#)
  - [DNS Adapters](#)
  - [PIN Storage](#)
  - [Auditor is deprecated](#)
- [Minor enhancements](#)
  - [ods-ksmutil: one step 'key backup' is deprecated](#)
  - [ods-ksmutil/enforcer enhancements](#)
  - [Signer enhancements](#)
  - [Database](#)
- [Versioning and Support Policy](#)
- [Bug fixes](#)
- [Notes](#)

### Major enhancements

#### DNS Adapters

OpenDNSSEC now supports both input and output adapters for AXFR and IXFR in addition to file transfer.

 Migration required:

- The [zonefetch.xml](#) file has been replaced by [addns.xml](#) to support this enhancement.
- Also changes to the KASP database mean that a database [migration is required](#) to upgrade to 1.4 from earlier versions of OpenDNSSEC.

#### PIN Storage

The HSM PIN can now be omitted from the conf.xml file and entered via the '[ods-hsmutil login](#)' command instead for increased security.

#### Auditor is deprecated

The auditor is no longer supported in 1.4. This greatly reduced the dependencies of OpenDNSSEC, namely it no longer depends on Ruby. Alternative validation tools are described [here](#).

#### Minor enhancements

(Some enhancements are also available in later 1.3 releases - see the 1.3 release NEWS file)

## ods-ksmutil: one step 'key backup' is deprecated

The command

```
ods-ksmutil backup done
```

is deprecated - for more details see [ods-ksmutil backup](#)

## ods-ksmutil/enforcer enhancements

- ods-ksmutil key list: key size, algorithm and next key state are included in output when -v flag is used
- ods-ksmutil rollover list: more information displayed on the KSKs waiting for the ds-seen command when the -v flag is used
- ods-ksmutil key generate: now displays how many keys will be generated and presents the user with the opportunity to stop the operation.
- [Optionally include CKA\\_ID in output of the DelegationSignerSubmitCommand](#)

## Signer enhancements

- Allow for Classless IN-ADDR.ARPA names (RFC 2317).

## Database

- [Provide script \(enforcer/utlils/convert\\_database.pl\) to convert database backend between different supported formats.](#)
- [Updated advice on SQLite usage.](#)

## Versioning and Support Policy

The versioning scheme used for releases and the release maintenance policy have both been updated as of 1.4. Please see the [Release Management Process](#) for details.

## Bug fixes

A full list of bug fixes and issue numbers can be found in the 1.4 release NEWS file.

## Notes

The 'Multi-threaded enforcer' feature (which was available in earlier beta versions of 1.4) was removed from the 1.4 release due to issues with the implementation. Note that the 2.0 release will deliver significant performance improvements for running with many zones.

# Getting Started

## How the site is organised

- [Overview](#): There is an Overview of OpenDNSSEC section with high level descriptions of how

OpenDNSSEC works, diagrams of workflows and common activities.

- **Documentation:** The other pages on this wiki site server as a detailed reference manual describing installing, configuring and running OpenDNSSEC.
- **Reference:** The Reference Material section includes highly recommended in depth training material and user contributed presentations on existing deployments of OpenDNSSEC.
- **Resources:** Resources such as FAQ and reporting BUGS are also available. There is also a user mailing list.

## Overview

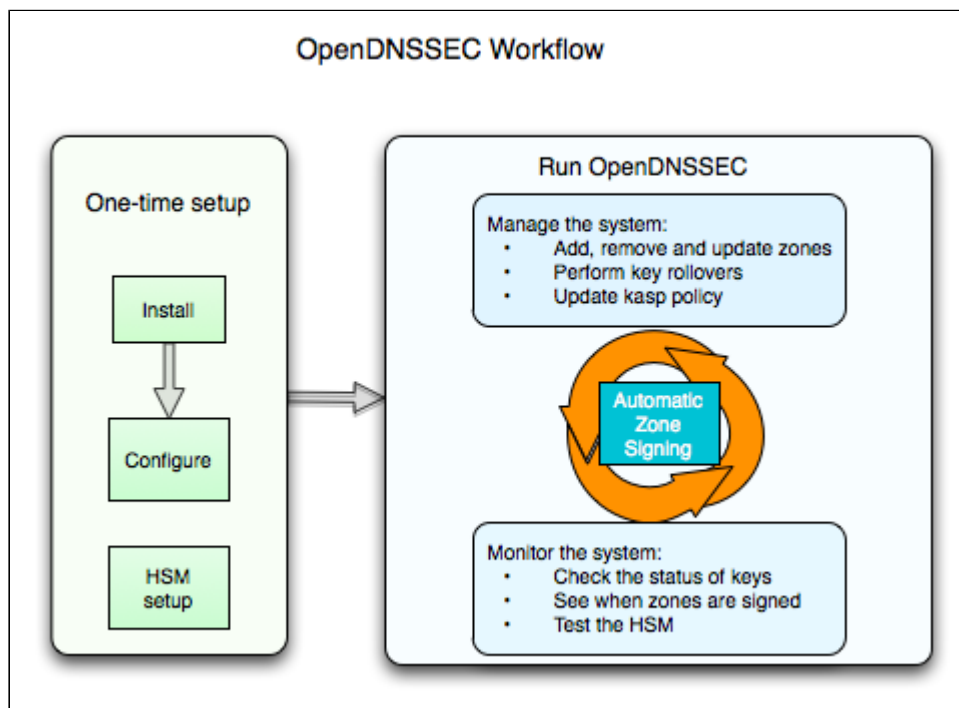
- **i** Once installed and configured OpenDNSSEC will automate key management and zone signing activities.

### OpenDNSSEC

Read an [overview of how OpenDNSSEC](#) works. This details the components, diagrams of workflows and common activities.

### Hardware Security Modules

Read about [HSM](#) and the OpenDNSSEC implementation of a soft HSM - [SoftHSM](#).



## Documentation

## ✓ Installation

Describes how you [install OpenDNSSEC](#) from source. It also gives some hints on how you can choose your hardware set-up to match your zone signing requirements.

## Configuration

The next step after installation is to [configure OpenDNSSEC](#). Remember that you also need an HSM with the PKCS#11 API.

## Running OpenDNSSEC

With the software installed and configured, it is time to [start using the system](#). Adding/removing zones, rolling keys, etc.

## Command utilities

The software package comes with a number of [command line tools](#).

## Reference Material

[Available here.](#)

## Resources

### i Troubleshooting

This chapter describes some [troubleshooting tips and experiences](#).

### FAQ

This chapter describes some [frequently asked questions](#) from the users.

### Reporting bugs

If you encounter any bugs: [Reporting bugs](#)

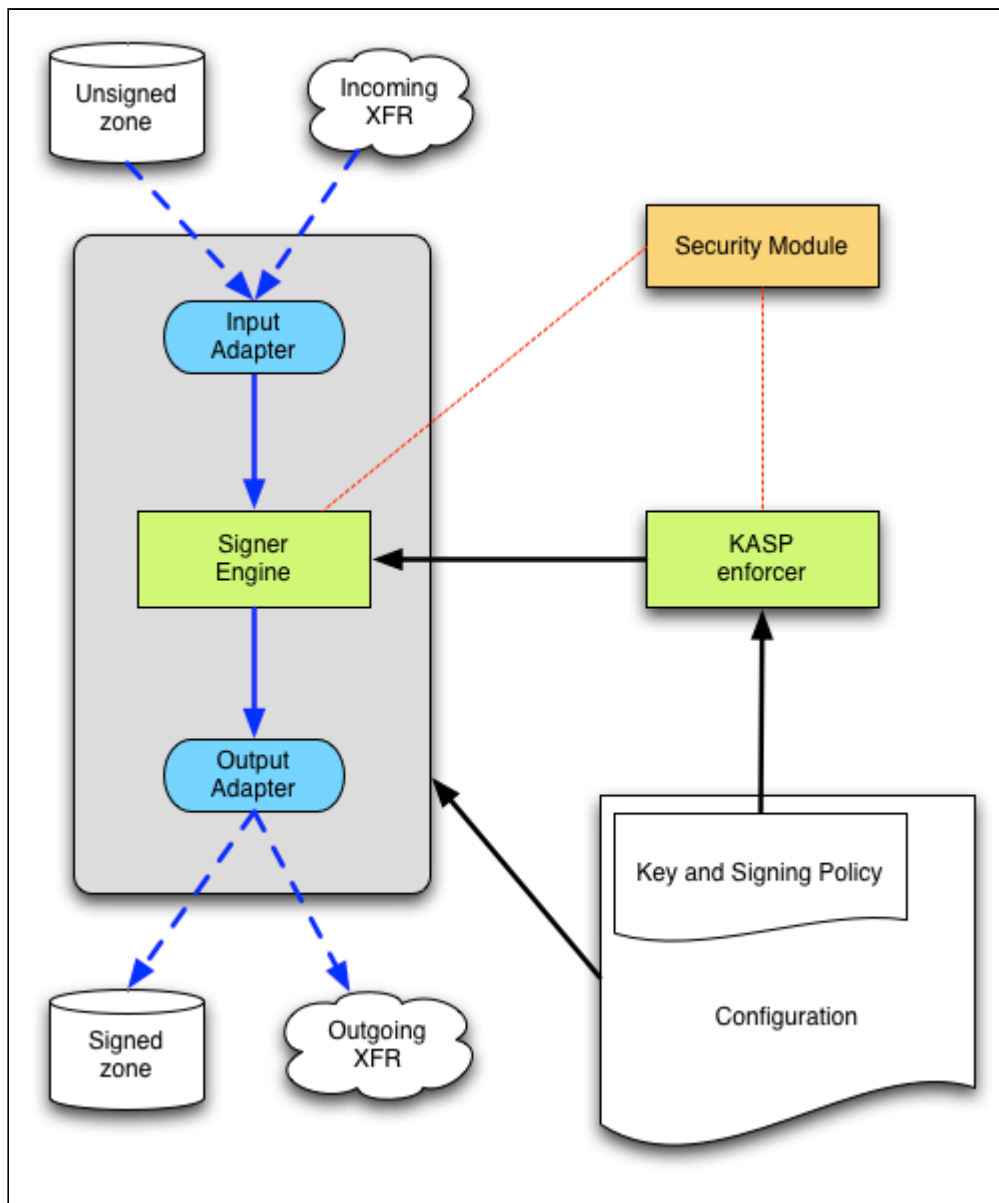
### Mailing list

<https://lists.opendnssec.org/mailman/listinfo/opendnssec-user>

## Overview of OpenDNSSEC

OpenDNSSEC takes in unsigned zones, adds the signatures and other records for DNSSEC and passes the zones on to the authoritative name servers for that zones.

It does this according to a Key and Signing Policy (KASP) that describes how an organisation wants their DNSSEC configured.



### On this Page

- [What does OpenDNSSEC do automatically?](#)
- [What can be done manually?](#)
- [What must be done manually?](#)
- [What are the key components of OpenDNSSEC?](#)

## What does OpenDNSSEC do automatically?

Once [installed](#), [configured](#) and running OpenDNSSEC will do the following:

- Receive unsigned zones from file or through XFR.
- Key Rollover: Generate, publish and retire keys held in an HSM according to policy. See the full key lifecycle in the [Key States](#) guide.
- Signing and re-signing of zones according to policy, including the reuse of signatures.



- Provide signed zones to file or to name servers via XFR.

## What can be done manually?

- Zones can be added, updated and removed.
- Keys can be backup and exported or managed manually.
- Manual key rollovers can be performed to cater for emergencies.
- The Key and Signing policy can be updated.

See the [Running OpenDNSSEC](#) guide for more details

## What must be done manually?

[Uploading the Trust Anchor](#) to the parent and notifying OpenDNSSEC that this has been done is a manual operation.

## What are the key components of OpenDNSSEC?

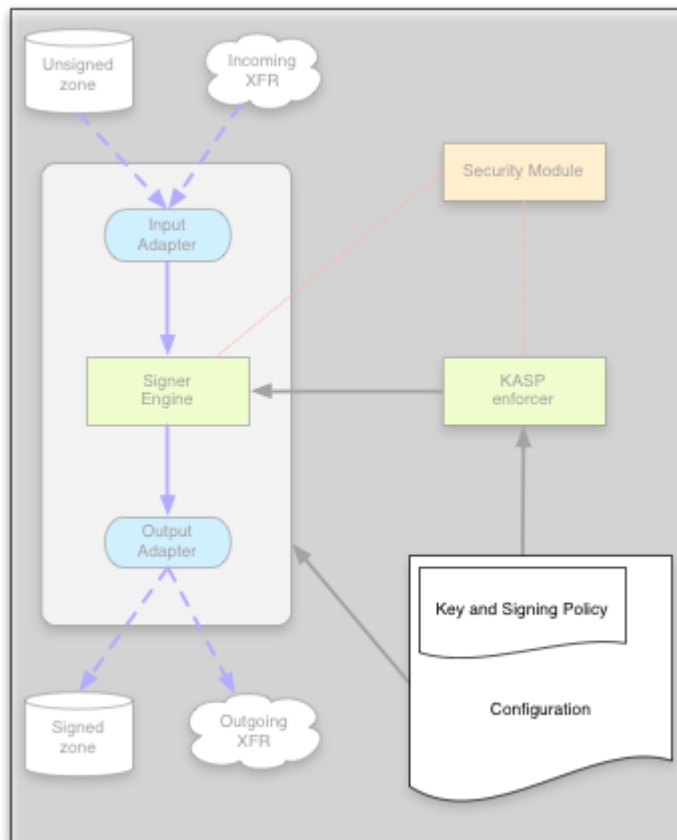
- **Configuration including:**
  - *KASP* - is the set of user defined policies to be used for signing and maintaining zones managed by this system.
- **Enforcer** - is responsible for enforcing the policy by managing the keys and orchestrating zone signing.
- **Signer** - is responsible for performing zone signing according to the instructions from the Enforcer. Is also responsible for ensuring that the zone is secure i.e. it will validate correctly.
  - *Input/Output Adapters* - are responsible for managing the input and output of zones via the specified mechanism (file, AXFR/IXFR).
- A [HSM](#) is also required for key management and storage.

## Components

- [Configuration](#)
  - [KASP](#)
- [Enforcer](#)
- [Signer](#)
  - [Engine](#)
  - [Adapters](#)

## Configuration

### *KASP*

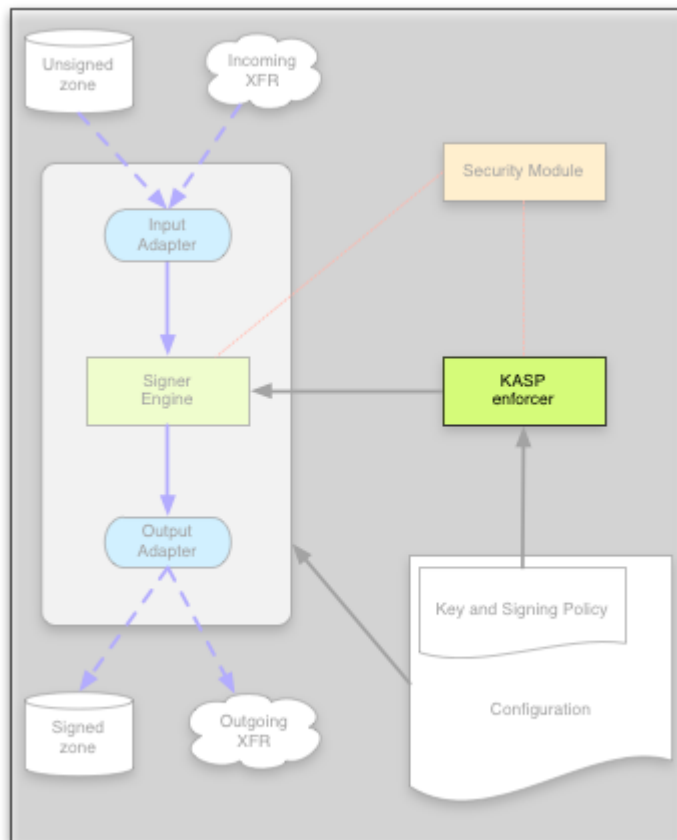


The KASP is one of the configuration files for OpenDNSSEC (more on configuration files [here](#)). It specifies a Key And Signing Policy which controls the following aspects of DNSSEC

- key strength
- key algorithm
- key lifetime
- signature lifetime
- NSEC vs NSEC3
- etc.

There can be one or many policies and these can be associated with different zones for fine control of DNSSEC.

## Enforcer

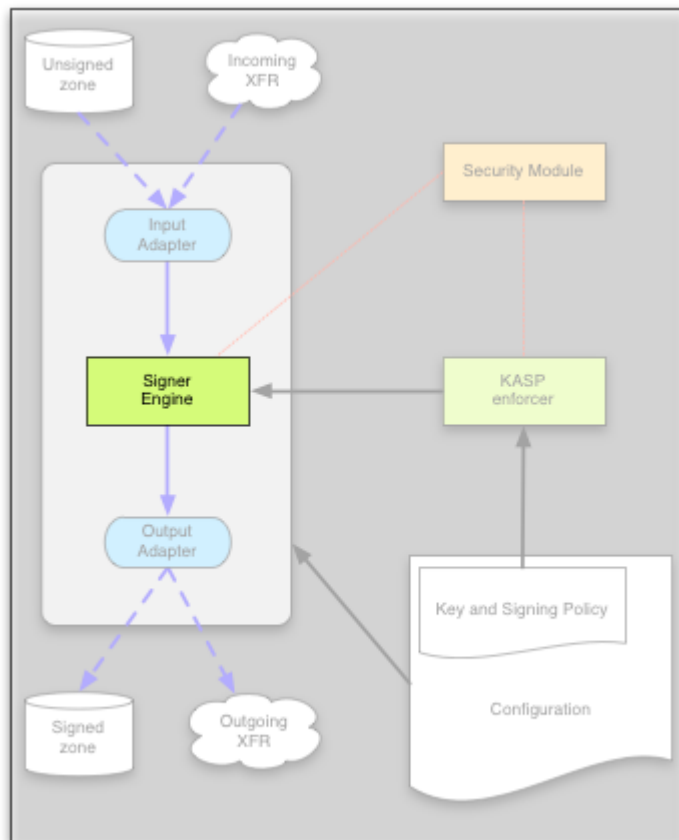


The KASP enforcer (also known as just "the enforcer") is responsible for the management of keys. It runs as a daemon and wakes periodically to check if key states need updating. It also has a command interface (ods-ksmutil) to provide information on key and zones states. It does the following tasks

- manages key creation using the HSM
- manages which zones are associated with which policies
- manages the key states and transition
- manages key rollovers; chooses which keys to use to sign the zone
- manages key backup

## Signer

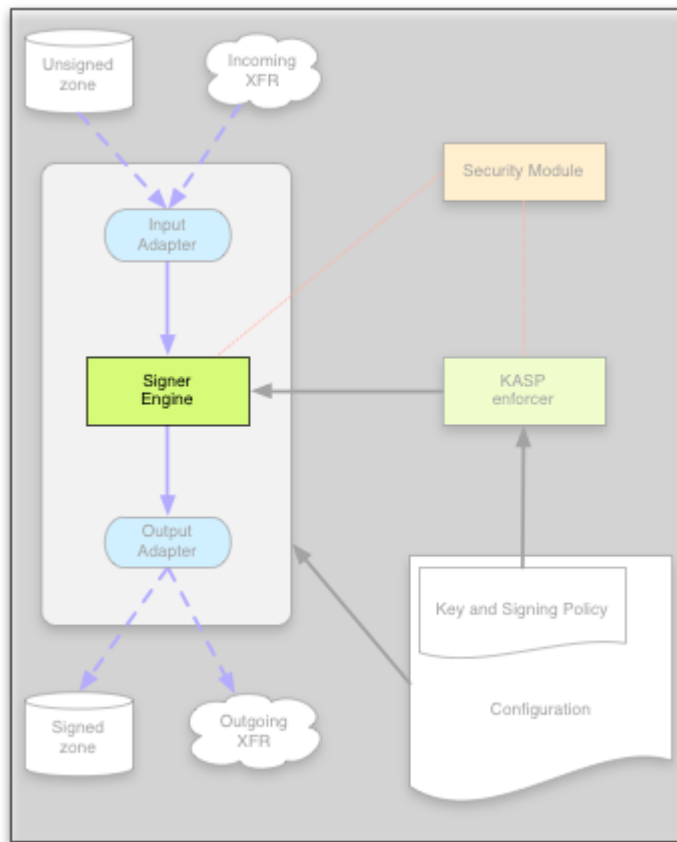
### *Engine*



The Signer Engine (also known as just "the signer") is responsible for actually performing the zone signing. It runs as a daemon and wakes periodically to check if the zones need updating. It also has a command interface (ods-signer) to manually control zone signing. It consumes information generated by the enforcer and unsigned zones and then generates zones signed with the specified keys:

- it can reuse signatures that are not too old
- it can spread signature expiration time over time (jitter)
- it maintains the NSEC/NSEC3 chain

### **Adapters**

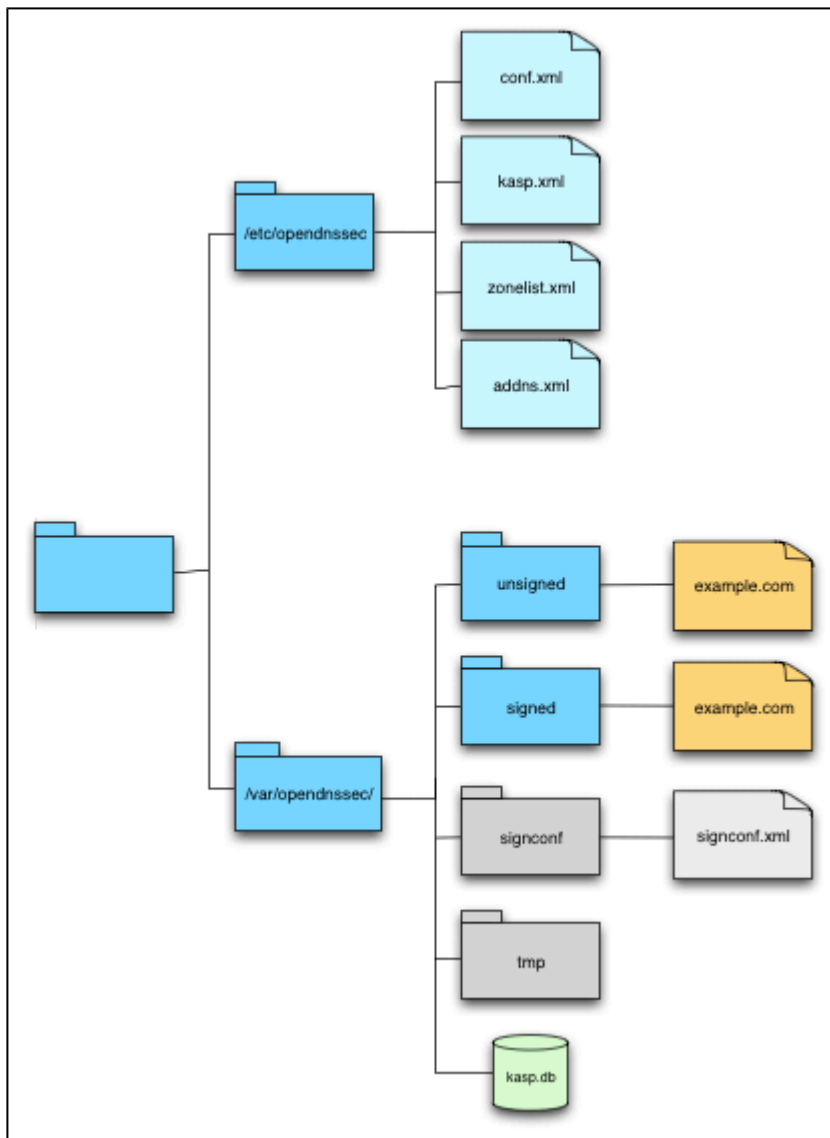


The Adaptors are responsible for obtaining the unsigned zone and distributing the signed zone. Currently supported mechanisms are (for both input and output):

- File: the zone files are held on disk
- AXFR: the zone files are obtained/distributed via AXFR
- IXFR: the zone files are obtained/distributed via IXFR where this is supported

## Files

This page describes the default setup of OpenDNSSEC.



OpenDNSSEC manages various information on disk which includes the following

- Configuration files (blue): xml files that specify the settings for the system
- Zone files (orange): unsigned and signed copies of the zones
- Working files/directories (grey): temporary files used by the system to exchange information between components and track internal state information
- Database (green): a database backend used by the enforcer to track key and zone status (assumes a SQLite database backend)

## Configuration files

Default location: `/etc/opendnssec`

Name	Information
conf.xml	<ul style="list-style-type: none"> <li>• Repository list (i.e. HSM configuration)</li> <li>• Common (logging, file locations)</li> <li>• Enforcer (database location, wake interval, etc.)</li> <li>• Signer (working directory, worker threads, etc.)</li> </ul>
kasp.xml	<p>Policy specifications:</p> <ul style="list-style-type: none"> <li>• Signature specifications (resign, validity, jitter, etc.)</li> <li>• Authenticated denial of existence <ul style="list-style-type: none"> <li>• NSEC or NSEC3, resalt, hash, etc.</li> </ul> </li> <li>• Key data <ul style="list-style-type: none"> <li>• Shared keys, TTL, etc.</li> <li>• KSK specifics (algorithm, lifetime, etc.)</li> <li>• ZSK specifics (algorithm, lifetime, etc.)</li> </ul> </li> <li>• Zone data <ul style="list-style-type: none"> <li>• Propagation delay, SOA parameters</li> </ul> </li> <li>• Parent zone information <ul style="list-style-type: none"> <li>• Propagation delay, SOA and DS parameters</li> </ul> </li> </ul>
zonelist.xml	<p>Zone specifications:</p> <ul style="list-style-type: none"> <li>• Zone name</li> <li>• Policy</li> <li>• Adapters</li> </ul>
addns.xml	<p>Adapter specifications:</p> <ul style="list-style-type: none"> <li>• TSIG data (name, algorithm, secret)</li> <li>• Inbound <ul style="list-style-type: none"> <li>• Transfer request configuration</li> <li>• Allow notify configuration</li> </ul> </li> <li>• Outbound <ul style="list-style-type: none"> <li>• Provide transfer configuration</li> <li>• Notify configuration</li> </ul> </li> </ul>

## Zone files

Default locations:

- /var/opendnssec/signed
- /var/opendnssec/unsigned

Signed and unsigned files managed by OpenDNSSEC

## Working files

Default locations:

- /var/opendnssec/signconf - temporary files used to exchange information between the enforcer and signer components. These files should not be edited by users but are useful for debugging
- /var/opendnssec/tmp - temporary working directory used to hold state information

## Key States

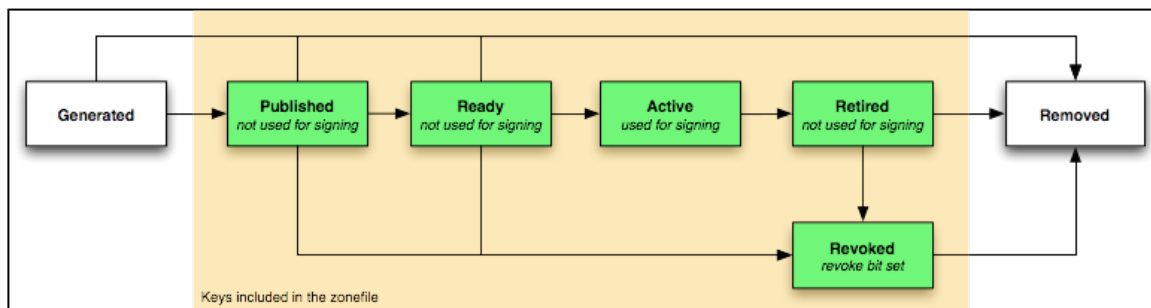
Most of the key states are as described in the DNSSEC key timing draft: <http://tools.ietf.org/html/draft-morris-dns-op-dnssec-key-timing-02>

Also see: <http://tools.ietf.org/html/draft-mekking-dnsop-dnssec-key-timing-bis-02>

The diagram below shows the states described in this draft.

### On this Page

- [Generate](#)
- [Publish](#)
- [Ready](#)
- [Active](#)
- [Retired](#)
- [Dead](#)
- [Standby KSK states](#)
  - [DSSUB](#)
  - [DSPUBLISH](#)
  - [DSREADY](#)
  - [KEYPUBLISH](#)



Below is a brief describe of what the states mean:

### Generate

Keys in the generate state have been created and stored but not used yet.

### Publish

Keys in the publish state have been published in the zone, but are not yet considered safe to use. (i.e. They have not been in the zone long enough to have propagated through the system.)



## Ready

Keys in the ready state have been published long enough that we could safely start to use them.

## Active

Keys in the active state are those that are in use.


## Retired


Keys in the retire state have been in use but have been replaced by a successor, they are post-published while signatures generated with them might still be in the system.

## Dead

Keys in the dead state have been retired long enough for them to be safely removed from the zone.

## Standby KSK states

 The standby keys are experimental. We are currently not supporting offline HSM, which is needed to get the security level needed to fulfill the idea behind standby keys. Will be fixed in a future version.

 For standby KSKs there are some extra states which replace Published and Ready. This is because the standby keys are not introduced into the zone until they are needed. Instead their DS record is submitted to the parent - see [Uploading a Trust Anchor](#). (The idea is that if the key is needed in an emergency the shortest timescale that it can be used in is the publication through the child system.)

## DSSUB

The DS has possibly been submitted (if it happened automatically) but in any case we are waiting for the ds-seen command.

## DSPUBLISH

The ds-seen command has been given, and we are now waiting for the various propagation delays and safety margins to pass.

## DSREADY

The DS record is now considered safe to use, so the standby key is ready.

## KEYPUBLISH

We have been asked to use the standby key so we have published it in the zone. Once the key has propagated through the system it will move into the active state.

## Key Rollovers

In theory a variety of key rollover mechanisms are possible and are described in detail in: <http://tools.ietf.org/html/draft-morris-dnsop-dnssec-key-timing-02>

Also see: <http://tools.ietf.org/html/draft-mekking-dnsop-dnssec-key-timing-bis-02>

A summary is given below:

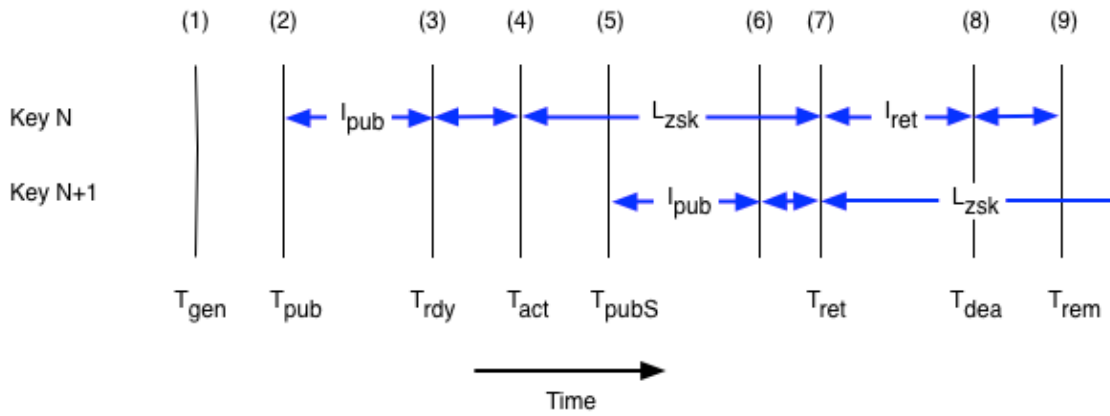
ZSK Method	KSK Method	Description
Pre-Publication	N/A	Publish DNSKEY before the RRSIG
Double-Signature	Double-signature	Publish DNSKEY and RRSIG at the same time. For a KSK, this happens before the DS is published
Double RR-sig	N/A	Publish RRSIG before the DNSKEY
N/A	Double-DS	Publish DS before DNSKEY
N/A	Double-RRset	Publish DNSKEY and DS in parallel.

OpenDNSSEC currently supports the following mechanisms:

- ZSK: Pre-Publication
- KSK: Double-Signature

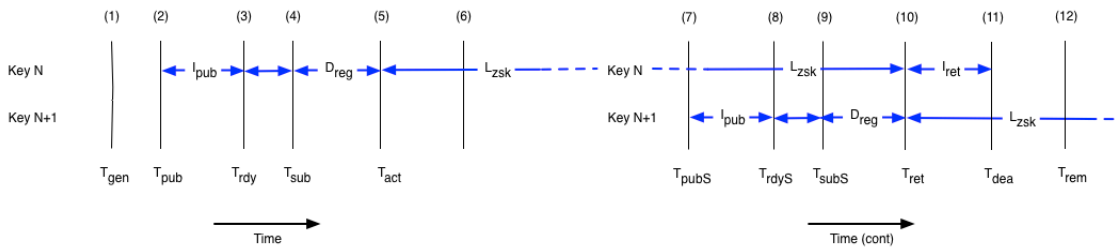
Future versions of OpenDNSSEC will support additional mechanisms.

### **ZSK rollovers: Pre-Publication**



- **First key:**  $I_{pub} = D_{prp} + \min(TTL_{soa}, SOA_{min})$
- **Future keys:**  $I_{pub} = D_{prp} + TTL_{key}$
- $T_{pubS} \leq T_{act} + L_{zsk} - I_{pub}$
- $I_{ret} = D_{sgn} + D_{prp} + TTL_{sig}$

### KSK rollovers: Double-Signature



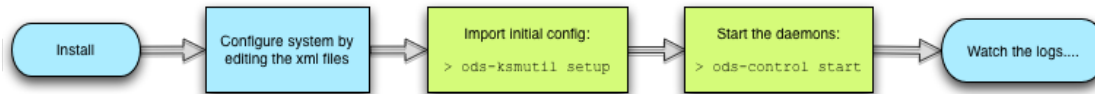
- $I_{pub} = D_{prp} + TTL_{key}$
- $T_{pubS} \leq T_{act} + L_{ksk} - D_{reg} - I_{pub}$
- $I_{ret} = D_{prp} + TTL_{ds}$

### Quick guides

- [Start OpenDNSSEC for the first time](#)
- [Pre-generate keys](#)
- [Backup keys](#)
- [Publish a DS](#)
  - [Manual export](#)
  - [Automatic export](#)
- [Add a zone](#)
  - [Using ksmutil](#)
  - [By importing zonelist.xml](#)

- [Manually update a zone file and re-sign](#)
- [Automatically reload a signed zone file to a nameserver](#)
- [Migrate a zone to OpenDNSSEC](#)
- [Backup](#)

## Start OpenDNSSEC for the first time



Also see: [Running OpenDNSSEC](#)

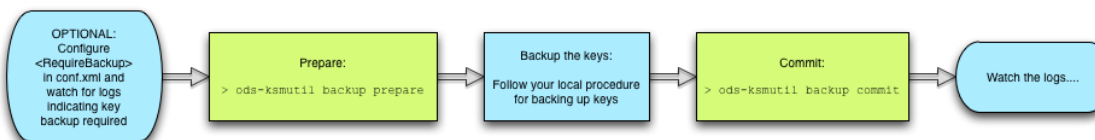
## Pre-generate keys

OpenDNSSEC will generate keys as needed, however if you wish to pre-generate a pool of keys for use later use the following command:

```
> ods-ksmutil key generate --policy
my_policy --interval P6M
```

Also see: [Key Management](#)

## Backup keys



Also see: [Key Management](#)

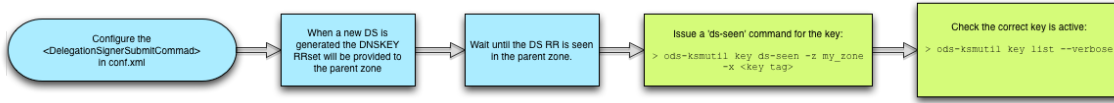
## Publish a DS

### Manual export



Also see: [Running OpenDNSSEC](#) and [Key Management](#)

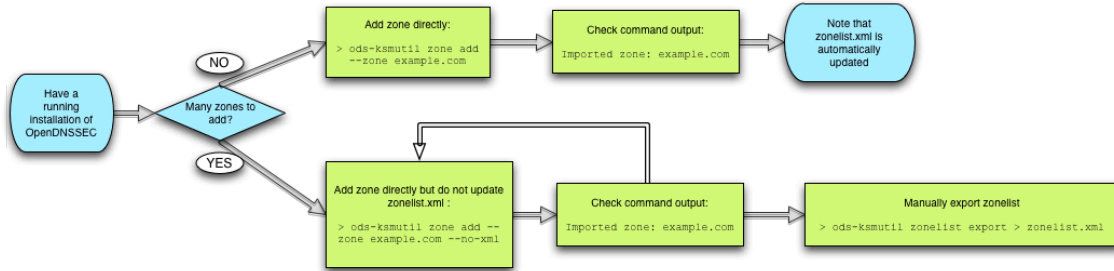
### Automatic export



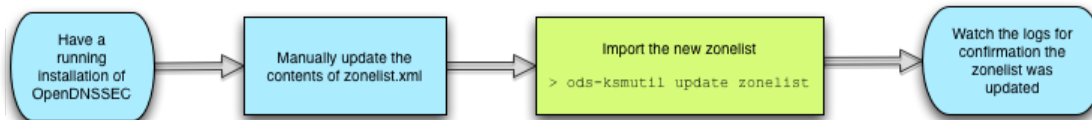
Also see: [conf.xml](#)

## Add a zone

### Using ksmutil

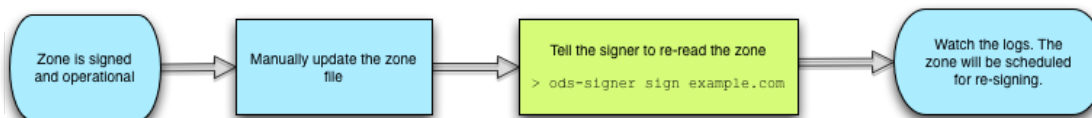


### By importing zonelist.xml



Also see: [Zone Management](#)

## Manually update a zone file and re-sign



Also see: [Zone Management](#)

## Automatically reload a signed zone file to a nameserver

Configure the <NotifyCommand> option in the [conf.xml](#) file, for example:

```

<NotifyCommand>rndc reload
%zone</NotifyCommand>
  
```

## Migrate a zone to OpenDNSSEC

See: [Migrating to OpenDNSSEC](#)

### Backup

The kasp database can be backed up with the following command:

```
ods-ksmutil database backup [--output  
<output>]
```

## Installation

Before you start to use OpenDNSSEC in your production environment you must first decide which hardware you going to run on.

When you have a good system to run on, then it is time to install the software that OpenDNSSEC depends on, and finally installing OpenDNSSEC.

### On this Page

- [Hardware set-up](#)
- [Platform support](#)
- [Database](#)
- [Dependencies](#)
- [Pre-built Binaries](#)
- [Obtaining the Source Code](#)
- [Building & Installing](#)
- [Post-installation](#)

### Hardware set-up

Here are some short recommendations if you are planning to use OpenDNSSEC with many zones or a single large zone, and where the speed of signing is important.

- **CPU:** OpenDNSSEC is multi-threaded when it concerns the handling of multiple zone. But it is not currently multi-threaded in the handling of a single zone. So a multi-core machine will not give any benefits if you plan to only run a single large zone.

✓ For handling a single large zone it is therefore more important to go with a CPU that is fast rather than a CPU with many cores.

- **Hardisk:** The OpenDNSSEC signer engine makes backup files to recover your zone data with no loss. A backup file will use up one time the size of the signed zone on the HDD. If you use an inbound DNS adapter, the XFR is also stored on disk. If you use an outbound DNS adapter, a journal file is required. OpenDNSSEC will maintain three outbound changes on disk.
- **Memory:** The zones are also stored in memory. The amount of memory is quite excessive, due to the use

of Idns.

## Platform support

OpenDNSSEC 1.4 has been tested on the following platforms:

- Debian 6.0.3 amd64
- Ubuntu Server 10.04.3 amd64
- Ubuntu Server 10.04.3 i386
- Ubuntu Server 12.04.01 amd64
- Red Hat Enterprise Linux 6.2 amd64
- CentOS 6.2 i386
- Scientific Linux 6.1 amd64
- OpenSUSE 12.1 amd64
- OpenSUSE 12.1 i386
- Solaris 11 11/11
- FreeBSD 9 amd64
- FreeBSD 9 i386
- OpenBSD 5.0 amd64
- NetBSD 5.1 amd64
- SUSE Linux Enterprise Server SP2 amd64

## Database

MySQL is recommended as the primary database backend for use with OpenDNSSEC in production environments. SQLite is supported but it is recommended that this is used only for testing. There are two reasons for this. Firstly SQLite does not scale well as the number of zones requiring signing grows. Secondly the current implementation of SQLite in OpenDNSSEC can be open to locking issues in certain circumstances.

## Dependencies

OpenDNSSEC depends on a number of open-source packages, all of which must be installed on your system for OpenDNSSEC to build successfully.

The [installation of dependencies guide](#) shows which packages are required and how to download/install them.

 **You also need a [Hardware Security Module](#).**

Choose from any vendor that uses the [PKCS#11](#) interface. Or the software-only implementation of an HSM called SoftHSM created by the OpenDNSSEC project. Follow these instructions on [how to install SoftHSM](#).

## Pre-built Binaries

You can find information about packages for your operating system here: <http://www.opendnssec.org/download/packages/>

## Obtaining the Source Code

The latest version of OpenDNSSEC can be found as a tarball on <http://www.opendnssec.org>

The development (unstable) version of OpenDNSSEC is available from the Subversion repository and can be obtained using the following command:

```
svn co
http://svn.opendnssec.org/trunk/OpenDNSSEC
OpenDNSSEC
```

## Building & Installing

1. If you downloaded the tarball then first untar it:

```
tar -xzf opendnssec-<VERSION>.tar.gz
cd OpenDNSSEC
```

or if you are working from the repository:

```
cd OpenDNSSEC
sh autogen.sh
```

2. Then it is time to configure the build scripts:

```
./configure
```

You may also need some other options to configure.

```
--enable-timeshift      For debugging purposes
--with-database-backend Select database backend (sqlite3|mysql)
(default sqlite)
```

Use the following command to find out which other options that are available:

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

The configure script defaults to `--prefix=/usr/local`, `--sysconfdir=/etc`, and `--localstatedir=/var`


3. Once configured, build OpenDNSSEC using:



```
make
```

... and install using ...

```
sudo make install
```

 If the build fails it might be because of a missing software dependency. Please read the error messages carefully.

## Post-installation

Depending on operating system, there may be a few additional steps required after installation.

**Linux Users** Linux users need to rebuild the dynamic linker caches. To do this, issue the command:

```
sudo ldconfig [library-path [library-path  
...]]
```

If OpenDNSSEC or any of the pre-requisites were installed in non-standard directories, the list of library paths should be specified as arguments on the command line.

## Dependencies


The following sections list the prerequisite software required for OpenDNSSEC.

### On this Page

- [Software versions](#)
  - [ldns](#)
  - [libxml2](#)
  - [MySQL](#)
  - [SQLite](#)

## Software versions

The following sections list the prerequisite software required for OpenDNSSEC. For Ubuntu users, the name of the package (where relevant) is listed.

 The version of the package available from the Ubuntu download sites may not be compatible with OpenDNSSEC; in that case, the latest version of the package should be obtained (and built if required).

Users of operating systems with different software packaging should consult the appropriate documentation.

Implicit in these sections is the assumption that the operating system has the following languages installed: C, C++. If any are absent, consult the documentation for your operating system.

In all cases, a location from where to get a copy of the package source code and build instructions for the package are given.

- ✓ As there are some dependencies between the prerequisite components, they should be installed in the order listed here.

Note, where no version number is specified any fairly recent distribution (e.g. Ubuntu 10.04) will have a new enough version of the software in its standard repositories. Also note that these are minimum version numbers, so they provide API calls that we use; there may be bug fixes in later versions which are useful.

Software	min. version for 1.4.0	min. version for trunk
ldns*	1.6.12	1.6.12
libxml2, libxml2-dev, libxml2-utils	2.6.16	2.6.16
java	not required	
sqlite3, libsqlite3, libsqlite3-dev	3.3.9	3.3.9
(mysql-client, libmysqlclient15, libmysqlclient15-dev)	5.0.3	5.0.3

\* Note that due to issues found in ldns version 1.3.11 (and later) of OpenDNSSEC does not support version 1.6.14 or 1.6.15 of ldns

## ldns

ldns is a DNS programming library used in the signer component.

### Ubuntu Users

Make sure the the packages "libldns-x.z.y" (where "x.y.z" is the ldns version number) and "libldns-dev" are installed on your system.

### Installing from Source Distribution

Download a copy of ldns from <http://www.nlnetlabs.nl/downloads/ldns>.

When downloaded and unpacked, "cd" into the directory into which you have unpacked the tar file and issue the following commands:

```
./configure (--disable-gost)
make
sudo make install
```

This installs the `ldns` library in `/usr/local/lib`. If you require the software to be installed elsewhere, add the switch `--prefix=<location>` to the `./configure` command.

## libxml2

`libxml2` is a C-library for handling XML. It is used in all parts of OpenDNSSEC.

### Ubuntu Users

Install the packages `"libxml2"`, `"libxml2-dev"`, and `"libxml2-utils"`.

### Installing from Source Distribution

Download a copy of `libxml2` from <http://xmlsoft.org/downloads.html>.

When downloaded and unpacked, `cd` into the directory into which you have unpacked the tar file and issue the following commands:

```
./configure
make
sudo make install
```

This installs the `libxml2` library in `/usr/local/lib`. If you require the software to be installed elsewhere, add the switch `--prefix=<location>` to the `./configure` command.

## MySQL

 MySQL is the recommended database for production environments.

### Ubuntu Users

Install the packages `"mysql-client"`, `"libmysqlclient15"`, `"libmysqlclient15-dev"`.

### Installing from Source Distribution

Download it from <http://dev.mysql.com/downloads/mysql>

At this site there are links for various different systems, or to the source code if you want to build the code yourself. Full documentation is also available from the download page.

## SQLite

SQLite is a cut-down SQL database system, used by the KASP component of OpenDNSSEC.

### Ubuntu Users

Install the packages "sqlite3" and "libsqlite3-dev".

### Installing from Source Distribution

Download a copy of SQLite from <http://www.sqlite.org/download.html>.

When downloaded and unpacked, "cd" into the directory into which you have unpacked the tar file and issue the following commands:

```
./configure
make
sudo make install
```

This installs sqlite in `/usr/local/bin`. If you require the software to be installed elsewhere, add the switch `--prefix=<location>` to the `./configure` command.

## Hardware Security Modules

### Documentation on Hardware Security Modules (HSMs).

Remember that to run OpenDNSSEC you also need an HSM, which uses the [PKCS#11](#) interface. We do provide the [SoftHSM](#), a software-only implementation of an HSM. Read the [HSM Buyer's Guide](#) for more information and consult the [list of HSM vendors](#).

- [SoftHSM Documentation](#)
- [HSM Buyer's Guide](#)
- [HSM Vendors](#)

## Migrating from earlier versions of OpenDNSSEC

Version 1.4 has some kasp database changes to allow for an update to the zonelist.xml schema (these changes support flexibility in the input and output adapters).


 Also see the MIGRATION file in the source code.

This means that in order to use version 1.4 of OpenDNSSEC with a database created with an earlier version of OpenDNSSEC there are two options:


1. **If you do NOT need to retain the key information for the system:**  
Wipe and recreate your kasp database (run `ods-ksmutl setup`) which *will lose all of the current state for the system*.
2. **If you DO need to retain the key information for the system:**
  - Assuming that you are currently using v1.3; run the sql statements against your existing database (depending on which database is used) given in:
    - `enforcer/utills/migrate_adapters_1.mysql` or

- enforcer/utls/migrate\_adapters\_1.sqlite3
- If you are using v1.1 and wish to maintain your existing keys then you will first need to run one of:
  - enforcer/utls/migrate\_keyshare\_mysql.pl or
  - enforcer/utls/migrate\_keyshare\_sqlite3.pl

depending on your database, before running the above.

 Although these scripts have been tested it is recommended to make a backup of your database prior to running them.

## Migrating between supported database backends


 See the [database](#) section of the installation guide for guidance on the difference between the supported backends.

### Migration script

In 1.4 a script has been provided to enable users to migrate between the supported backends for a given version of OpenDNSSEC. The script is located here:

```
enforcer/utls/convert_database.pl
```

In 1.4 the supported backends are SQLite and MySQL.

 You must create the MySQL database and database user manually before converting to MySQL.

If users wish to convert to another OpenDNSSEC version you need to migrate existing database to that version first - see [Migrating from earlier versions of OpenDNSSEC](#).

### Examples

*Convert from SQLite to MySQL:*

```
enforcer/utils/convert_database.pl --from
dbi:SQLite:dbname=/var/opendnssec/kasp.db
--to dbi:mysql:database=kasp;host=localhost
--to-username <username> --to-password
<password>
```

*Convert from MySQL to SQLite:*

```
enforcer/utils/convert_database.pl --from
dbi:mysql:database=kasp;host=localhost
--from-username <username> --from-password
<password> --to
dbi:SQLite:dbname=/var/opendnssec/kasp.db
```

 Also see the MIGRATION file in the source code.

## Configuration

Configuring OpenDNSSEC has a number of aspects:

### Configuration Files

These can be customised for each installation, although the default configuration installs good default values. The individual files and tools to check the configuration files are described [here](#).

### Keys

Configure your [HSM](#) as required.

### Zones

Check the supported [zone formats](#).

### External Auditing of Zones

The auditor component of OpenDNSSEC was removed in 1.4. The [Zone Auditing](#) page describes alternative validation tools and configurations.

## Migration

If you are migrating to OpenDNSSEC read our [Migration guide](#).

## Configuration files

The default configuration installs good default values for anyone who just wants to sign their domains with DNSSEC. There are four configuration files for the basic OpenDNSSEC installation. You have

- *conf.xml* which is the overall configuration of the system,
- *kasp.xml* which contains the policy of signing,
- *zonelist.xml* where you list all the zones that you are going to sign,
- *addns.xml* (per zone, optional) for zone transfers.

Click on the filenames below to see details of the file contents.

### On this Page

- [Date/time durations](#)
- [Files](#)
  - [conf.xml](#)
  - [kasp.xml](#)
  - [zonelist.xml](#)
  - [addns.xml](#)
- [Signer configuration](#)
- [Checking your configuration files](#)

## Date/time durations

 Please read this description of [how date/time durations are used](#) in the configuration files.

## Files

### [conf.xml](#)

The overall configuration of OpenDNSSEC is defined by the contents of the file */etc/opendnssec/conf.xml*. In this configuration file you specify logging facilities (only syslog is supported now), system paths, key repositories, privileges, and the database where all key and zone information is stored.

### [kasp.xml](#)

*kasp.xml* - found by default in */etc/opendnssec* - is the file that defines policies used to sign zones. KASP stands for "Key and Signature Policy", and each policy details

- security parameters used for signing zones
- timing parameters used for signing zones

You can have any number of policies and refer to the proper one by name in for example the *zonelist.xml* configuration file.

### [zonelist.xml](#)

The *zonelist.xml* file is used when first setting up the system, but also used by the *ods-signerd* when signing

zones. For each zone, it contains a *Zone* tag with information about

- the zone's DNS name
- the policy from *kasp.xml* used to sign the zone
- how to obtain the zone
- how to publish the zone

### [addns.xml](#)

OpenDNSSEC can sign zone files on disk, but can also receive and server zone transfers (both AXFR and IXFR). If you configure a listener in *conf.xml*, the Signer Engine will kick off a DNS handler that will listen to queries, NOTIFY messages from the master and zone transfer requests from secondaries.

Information in this file details

- where to fetch zone data from
- protection mechanisms to be used

### Signer configuration

There are also xml files for each of the zones that the user wants to sign, but those are only used for communication between the Enforcer and the Signer Engine. And they are created automatically by the Enforcer. The location of these files can be found in *zonelist.xml*.

Read more [details about Signer configuration](#)

### Checking your configuration files

The OpenDNSSEC XML configuration files (*conf.xml* and *kasp.xml*) offer the user many options to customise the OpenDNSSEC signing system. Not all possible configuration texts are meaningful however.

A tool ([ods-kaspcheck](#)) is provided to check that the configuration files (*conf.xml* and *kasp.xml*) are semantically sane and contain no inconsistencies.

- ✓ It is advisable to use this tool to check your configuration before starting to use OpenDNSSEC.

### Date Time durations

This section explains how timing is described in all of the configuration files.

All date/time durations in the configuration files are specified as defined by [ISO 8601](#).

- i Durations are represented by the format P[n]Y[n]M[n]DT[n]H[n]M[n]S. In these representations, the [n] is replaced by the value for each of the date and time elements that follow the [n]. Leading zeros are not required. The capital letters 'P', 'Y', 'M', 'D', 'T', 'H', 'M', and 'S' are designators for each of the date and time elements and are not replaced.

- P is the duration designator (historically called "period") placed at the start of the duration representation.
- Y is the year designator that follows the value for the number of years.
- M is the month designator that follows the value for the number of months.
- D is the day designator that follows the value for the number of days.
- T is the time designator that precedes the time components of the representation.



- H is the hour designator that follows the value for the number of hours.
- M is the minute designator that follows the value for the number of minutes.
- S is the second designator that follows the value for the number of seconds.

For example, "P3Y6M4DT12H30M5S" represents a duration of "three years, six months, four days, twelve hours, thirty minutes, and five seconds". Date and time elements including their designator may be omitted if their value is zero, and lower order elements may also be omitted for reduced precision. For example, "P23DT23H" and "P4Y" are both acceptable duration representations.

**⚠ Exception:** A year or month vary in duration depending on the current date. For OpenDNSSEC, we assume fixed values

- One month is assumed to be 31 days.
- One year is assumed to be 365 days.

## conf.xml

The overall configuration of OpenDNSSEC is defined by the contents of the file `/etc/opendnssec/conf.xml`. In this configuration file you specify logging facilities (only syslog is supported now), system paths, key repositories, privileges, and the database where all key and zone information is stored.

Date/time durations are as specified [here](#).

### On this Page

- [The elements of the conf.xml file](#)
  - [Preamble](#)
  - [Configuration](#)
  - [Repositories](#)
  - [Common](#)
  - [Enforcer](#)
  - [Signer Configuration](#)
  - [Auditor Configuration](#)

The elements of the conf.xml file

## Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: conf.xml.in 1143 2009-06-24
12:10:40Z jakob $ -->
```

Each XML file starts with a standard element "`<?xml...>`". As with any XML file, comments are included between the delimiters "`<!--`"

and "`-->`".

## Configuration

```
<Configuration>
```

The enclosing element of the XML file is the element `<Configuration>` which, with the closing element `</Configuration>`, brackets the configuration information.

## Repositories

```
...  
<RepositoryList>
```

OpenDNSSEC stores its keys in "repositories". There must be at least one repository, although the system can be configured to run with multiple repositories.

- i** There are a number of reasons for running with multiple repositories, including:
- Temporarily running with multiple repositories whilst a switch is made from one repository to another, e.g. when replacing hardware.
  - The chosen type of repository has a limit on the number of keys that can be stored, and multiple repositories are needed to handle the chosen number of keys.
  - Different types of repository are needed for different security levels, e.g. a key-signing key may require a higher level of security than a zone-signing key.

In practice, all repositories are "HSM"s (hardware security modules) or an emulation of one.

### SoftHSM

The software emulation of an HSM - SoftHSM - must be installed separately but is part of the OpenDNSSEC project, and its definition is shown below. The element names will be explained later.

```

...
  <Repository name="SoftHSM">

<Module>/usr/local/lib/libsoftism.so</Modu
le>
  <TokenLabel>OpenDNSSEC</TokenLabel>
  <PIN>1234</PIN>
</Repository>

```

### HSM

As indicated, any number of repositories can be specified in the configuration file:

```

<!--
  <Repository name="sca6000">
    <Module>/usr/lib/libpkcs11.so</Module>
    <TokenLabel>Sun Metaslot</TokenLabel>
    <PIN>test:1234</PIN>
    <Capacity>1000</Capacity>
    <RequireBackup/>
    <SkipPublicKey/>
  </Repository>
-->

```

(The example above is commented out by the XML comment delimiters.)

- *<Repository>* - the definition of a repository is bracketed by the *<Repository>* and *</Repository>* elements. The name attribute must be supplied and must be unique. It is this name that is used in the [kasp.xml](#) file to identify which repository holds the keys. This field is limited to 30 characters.
- *<Module>* identifies the dynamic-link library that controls the repository. Each type of HSM will have its own library.
- *<TokenLabel>* identifies the "token" within the HSM that is being used - essentially a form of sub-repository. The token label is also used where there are two repositories of the same type, in that each repository should contain a different token label sub-repository. OpenDNSSEC will automatically go

to the right HSM based on this. This field is limited to 32 characters.

- `<PIN>` is an optional element containing the password to the HSM. OpenDNSSEC have this stored *en-claire* in the configuration file. If the PIN is not present in the configuration file, then it must be entered by using the command `ods-hsmutil login`.
- `<Capacity>` indicates the maximum number of keys the HSM can store. It is an optional element - if there is no (realistic) limit to the number of keys, remove it.
- `<RequireBackup>` is an optional element that specifies that keys from this repository may not be used until they are backed up. If backup has been done, then use `ods-ksmutil` to notify OpenDNSSEC about this. The backup notification is needed for OpenDNSSEC to be able to complete a key rollover. (If the backup is not done then the old key will remain in use.)
- `<SkipPublicKey>` is an optional element which specifies that the public key objects should not be stored or handled in the HSM. The public key is needed in order to create the DNSKEY RR. In theory, the public part of the key is also available in the private key object. However, the PKCS#11 API does not require the HSM to behave in this way. We have not seen a HSM where we cannot do this, but you should remove this flag if you are having any problem with it. The benefit of adding this flag is that you save space in your HSM, because you are only storing the private key object.

```
...  
</RepositoryList>
```

The list of repositories ends with the closing tag.

## Common

These are the configuration that is shared among the components of OpenDNSSEC.

```
...
<Common>
  <Logging>
    <Verbosity>3</Verbosity>

<Syslog><Facility>local0</Facility></Syslog>
</Logging>

<PolicyFile>/etc/opendnssec/kasp.xml</PolicyFile>

<ZoneListFile>/etc/opendnssec/zonelist.xml
</ZoneListFile>
</Common>
```

All components of OpenDNSSEC logs errors, warnings and progress information. This is configurable and defined in the <Logging> element. Currently, the only syslog() feature configurable via the OpenDNSSEC configuration file is the facility name under which messages are logged. This can be any of the names listed in the operating system's syslog header file (usually /usr/include/sys/syslog.h, but the location is system dependent). <Verbosity> controls the level of logging, 0 will disable logging and 3 (default level) will provide informational log messages. You can set it higher to get debug log messages.

- ✓ Although any facility listed there can be used, it is recommended that one of the "local" facilities (usually "local0" through "local7") be used.

Then you also have pointers to where the policy and zone list files can be found. There are also an optional element where you specify the path of the zone fetch configuration used for inbound AXFR.

## Enforcer

The KASP enforcer component of OpenDNSSEC - which deals with key rollover and key generation - has its own section in the configuration file:

```

...
<Enforcer>
<!--
  <Privileges>
    <User>opendnssec</User>
    <Group>opendnssec</Group>
  </Privileges>
-->

<!--
  <WorkerThreads>4</WorkerThreads>
-->

<Datastore><SQLite>/var/opendnssec/kasp.db
</SQLite></Datastore>
  <Interval>PT3600S</Interval>
  <!-- <ManualKeyGeneration/> -->
  <!--
<RolloverNotification>P14D</RolloverNotifi
cation> -->
  <!--
<DelegationSignerSubmitCommand>/usr/local/
sbin/simple-dnskey-mailer.sh</DelegationSi
gnerSubmitCommand> -->
</Enforcer>

```

The section is bracketed by the `<Enforcer>` .. `</Enforcer>` tags.

- `<Privileges>` The Enforcer can drop its privileges if specified.
- `<WorkerThreads>` This option is not supported in the stable 1.4 release. If it used it is silently ignored.

Multiple worker threads were supported in early beta versions of 1.4 and this option was left in the conf.xml file for compatibility. It will be removed in 2.0.

- `<Datastore>` The database used by the Enforcer is specified by the `<Datastore>` tag. OpenDNSSEC supports SQLite and MySQL (MySQL is the recommended choice for production environments), the choice being indicated by one of two mutually exclusive tags:

#### SQLite

If SQLite is the database in use, the `<Datastore>` tag must contain a single `<SQLite>` tag which specifies the database file in use (as shown above).

#### MySql

If MySQL is in use, then the `<Datastore>` contains a single `<MySQL>` tag, which in turn contains elements that describe the database connection. The following XML elements shows this:

```

...
<Datastore>
  <MySQL>
    <Host port="1213">dnssec-db</Host>
    <Database>KASP</Database>
    <Username>kaspuser</Username>
    <Password>abc123</Password>
  </MySQL>
</Datastore>

```

- `<Host>` is the name of the system on which the database resides. It is optional - if omitted, the database is assumed to run on the same system as OpenDNSSEC. The "port" attribute gives the port to which the MySQL connection is made. It too is optional, and defaults to 3306 if omitted.
- `<Database>` is the name of the database holding the KASP Enforcer data.
- `<Username>` is the username needed to connect to the database.
- `<Password>` is the password associated with the username.

#### Other Enforcer Parameters

- `<Interval>` is how often the Enforcer runs to check whether keys are coming up for expiry and should be rolled. The more frequently this is run, the closer will the usage of keys reflect the policy set for it. However, if the key lifetimes are in the order of months, an `<Interval>` of the order of a day to a week is sufficient.
- `<ManualKeyGeneration/>` will disable the automatic key generation in the Enforcer. The user have to generate the keys itself with the `ods-ksmutil key generate` command.
- `<RolloverNotification>` specifies how long before a KSK rollover the Enforcer should start logging messages about the future rollover.
- `<DelegationSignerSubmitCommand>` Configure the `<DelegationSignerSubmitCommand>` if you want to

have a program/script receiving the new KSK during a key rollover.

- ✓ This will make it possible to create a fully automatic KSK rollover, where OpenDNSSEC feed your program/script on *stdin* with the current set of DNSKEYs that we want to have in the parent as DS RRs.

*Key Identifier:* If the command defined here ends with "`--cka_id`" then that will be stripped off the command and "`<CKA_ID>`" will be added to the output.

An example script for the `<DelegationSignerSubmitCommand>` command is available: which is a simple mail script (from 1.4 the `epclient` is no longer supported). Remember that the `ods-ksmutil key ds-seen` must be given in order to complete the rollover. This should only be done when the new DS RRs are available on the parents public nameservers.

## Signer Configuration

The Signer Engine of OpenDNSSEC - the part that constructs signature records to include in to the zone file - also has its own section in the configuration file:



```
...
<Signer>
<!--
  <Privileges>
    <User>opendnssec</User>
    <Group>opendnssec</Group>
  </Privileges>
-->

<WorkingDirectory>/var/opendnssec/tmp</WorkingDirectory>
  <WorkerThreads>4</WorkerThreads>
  <SignerThreads>4</SignerThreads>

<!--
  <Listener>
    <Interface><Port>53</Port></Interface>
  </Listener>
-->

<!--

<NotifyCommand>/usr/local/bin/my_nameserver_reload_command</NotifyCommand>
-->

</Signer>
```

Delimited by the `<Signer>` .. `</Signer>` tags, the components are:

- `<Privileges>` the Signer Engine can drop its privileges if specified.
- `<WorkingDirectory>` defines the location in which the Signer Engine will create temporary files.
- `<WorkerThreads>` specify the number of workers. One worker can handle one zone a time. When it is finished with the zone it takes the next one in queue.
- `<SignerThreads>` specify the number of threads that are dedicated to signing RRsets. Usually set to the number of parallel operations your HSM can handle. If the element is omitted from the configuration, the number of signer threads is equal to the number of workers.
- `<Listener>` is an element that is needed when you use DNS adapters. Within this element, you can specify a number of interfaces the Signer Engine has to listen to for DNS traffic, such as queries, NOTIFY messages and zone transfer requests. `<Interface>` has two optional elements: either `<IPv4>` or `<IPv6>`, and `<Port>`. If no IP address is given, the Signer Engine defaults to both local IPv4 and IPv6 address. The default port is 53.
- `<NotifyCommand>` optional element that will tell the Signer Engine to call this command when the zone has been signed. Will expand the following variables: `%zone` (the name of the zone that was signed) and `%zonefile` (the filename of the signed zone).

## Auditor Configuration

The Auditor has been removed from OpenDNSSEC since 1.4.0.

```
</Configuration>
```

As there are no more elements, the `</Configuration>` tag closes the file.

### kasp.xml

**i** KASP stands for "Key and Signature Policy"

kasp.xml (found by default in `/etc/opendnssec`) is the file that defines policies used to sign zones. Each policy comprises a series of parameters that define the way the zone is signed. This section explains the parameters by referring to the example kasp.xml file supplied with the OpenDNSSEC distribution.

Date/time durations are as specified [here](#).

#### On this Page

- [Elements of the kasp.xml file](#)
  - [Preamble](#)
  - [Policy Description](#)
  - [Signatures](#)
  - [Authenticated Denial of Existence](#)
  - [Key Information](#)
  - [Zone Information](#)
  - [Parent Zone Information](#)

### Elements of the kasp.xml file

## Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: kasp.xml.in 1154 2009-06-24
13:02:25Z jakob $ -->
```

Each XML file starts with a standard element "<?xml...". As with any XML file, comments are included between the delimiters "<!--" and "-->".

## Policy Description

```
<KASP>
```

The enclosing element of the XML file is the element <KASP> which, with the closing element </KASP>, brackets one or more policies.

```
...
<Policy name="default">
```

Each policy is included in the <Policy>...</Policy> elements. Each policy has a "name" attribute giving the name of the policy. The name is used to link a policy and the zones signed using it; each policy must have a unique name.

The policy named "default" is special, as it is associated with all zones that do not have a policy explicitly associated with them.

```
...
<Description>A default policy that will
amaze you and your friends</Description>
```

A policy can have a description associated with it. Unlike XML comments, the description can be understood by programs and may be used to document the policy, e.g. a future GUI may display a list of policies along with their description and ask you to select one for editing.

## Signatures

The next section of the file is the Signatures section, which lists the parameters for the signatures created using the policy.

```

...
<Signatures>
  <Resign>PT2H</Resign>
  <Refresh>P3D</Refresh>
  <Validity>
    <Default>P14D</Default>
    <Denial>P14D</Denial>
  </Validity>
  <Jitter>PT12H</Jitter>

<InceptionOffset>PT300S</InceptionOffset>
</Signatures>

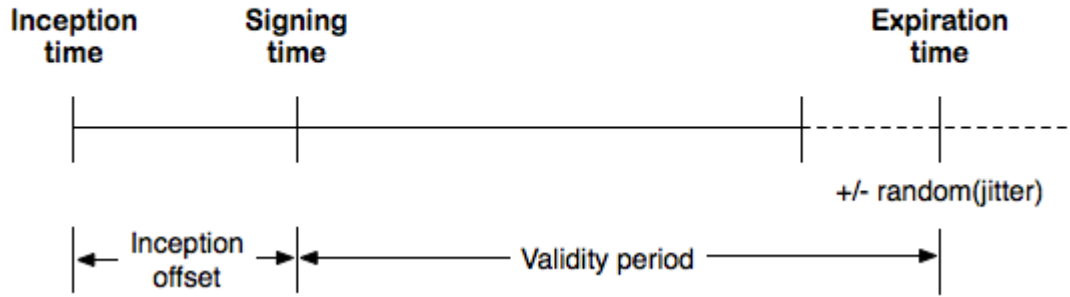
```

Here:

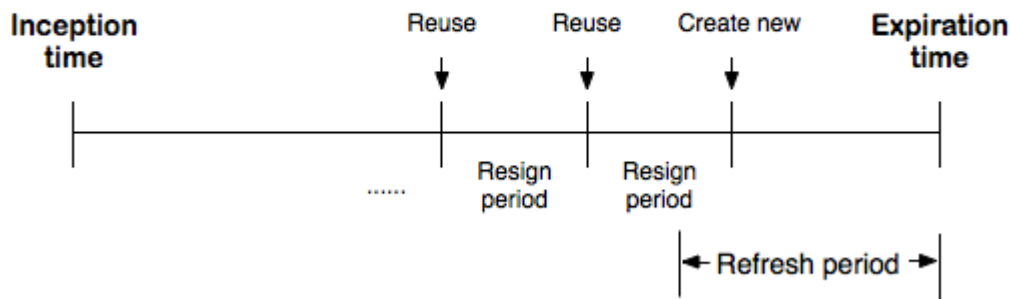
- <Resign> is the re-sign interval, which is the interval between runs of the Signer Engine.
- <Refresh> is the refresh interval, detailing when a signature should be refreshed. As signatures are typically valid for much longer than the interval between runs of the signer, there is no need to re-generate the signatures each time the signer is run if there is no change to the data being signed. The signature will be refreshed when the time until the signature expiration is closer than the refresh interval. Set it to zero if you want to refresh the signatures each re-sign interval.
- <Validity> groups two elements of information related to how long the signatures are valid for - <Default> is the validity interval for all RRSIG records *except* those related to NSEC or NSEC3 records. In this case, the validity period is given by the value in the <Denial> element.
- <Jitter> is the value added to or extracted from the expiration time of signatures to ensure that not all signatures expire at the same time. The actual value of the <Jitter> element is the  $-j + r \% 2j$ , where  $j$  is the jitter value and  $r$  a random duration, uniformly ranging between  $-j$  and  $j$ , is added to signature validity period to get the signature expiration time.
- <InceptionOffset> is a duration subtracted from the time at which a record is signed to give the start time of the record. This is required to allow for clock skew between the signing system and the system on which the signature is checked. Without it, the possibility exists that the checking system could retrieve a signature whose start time is later than the current time.

The relationship between these elements is shown below.

### Signature lifetime



### Reuse of signatures



## Authenticated Denial of Existence

Authenticated denial of existence - proving that domain names do not exist in the zone - is handled by the <Denial> section, as shown below:

...

```

<Denial>
  <NSEC3>
    <OptOut/>
    <Resalt>P100D</Resalt>
    <Hash>
      <Algorithm>1</Algorithm>
      <Iterations>5</Iterations>
      <Salt length="8"/>
    </Hash>
  </NSEC3>
</Denial>

```

<Denial> includes one element, either <NSEC3> (as shown above) or <NSEC>.

### NSEC3

- <NSEC3> tells the signer to implement NSEC3 scheme for authenticated denial of existence (described in [RFC 5155](#)). The elements are:
- <OptOut/> - if present, enable "opt out". This is an optimisation that means that NSEC3 records are only created for authoritative data or for secure delegations; insecure delegations have no NSEC3 records. For zones where a majority of the entries are delegations that are not signed - typically TLDs during the take-up phase of DNSSEC - this reduces the number of DNSSEC records in the zone.
- <Resalt> is the interval between generating new salt values for the hashing algorithm.
- <Algorithm>, <Iterations> and <Salt> are parameters to the hash algorithm, described in [RFC 5155](#).

### NSEC

Should, instead, NSEC be used as the authenticated denial of existence scheme, the <Denial> element will contain the single element <NSEC/> - there are no other parameters.

## Key Information

Parameters relating to keys can be found in the <Keys> section.

```
...  
<Keys>
```

### Common Parameters

The section starts with a number of parameters relating to both zone-signing keys (ZSK) and key-signing keys (KSK):

```
...  
<TTL>PT3600S</TTL>  
<RetireSafety>PT3600S</RetireSafety>  
  
<PublishSafety>PT3600S</PublishSafety>  
<ShareKeys />  
<Purge>P14D</Purge>
```

- <TTL> is the time-to-live value for the DNSKEY resource records.
- <PublishSafety> and <RetireSafety> are the publish and retire safety margins for the keys. These intervals are safety margins added to calculated timing values to give some extra time to cover unforeseen events, e.g. in case external events prevent zone publication.

If multiple zones are associated with a policy, the presence of <ShareKeys/> indicates that a key can be shared between zones. E.g. if you have 10 zones then you will only use one set of keys instead of 10 sets. This will save space in your HSM. If this tag is absent, keys are not shared between zones.

If <Purge> is present, keys marked as dead will be automatically purged from the database after this interval.

### Key-Signing Keys


Parameters for key-signing keys are held in the <KSK> section:

```

...
<KSK>
  <Algorithm length="2048">8</Algorithm>
  <Lifetime>P1Y</Lifetime>
  <Repository>softHSM</Repository>
  <!-- <Standby>1</Standby>
(Experimental) -->
  <ManualRollover/>
</KSK>

```

- <Algorithm> determines the algorithm used for the key (the numbers reserved for each algorithm can be found in the appropriate [IANA registry](#)).

 Once a zone is signed, changes to the algorithm require a rollover which is not currently handled by OpenDNSSEC. Attempts to change the algorithm on a policy will result in a warning message and a request for confirmation.

- <Lifetime> determines how long the key is used for before it is rolled.
- <Repository> determines the location of the keys. Keys are stored in "repositories" (currently, only hardware security modules (HSMs) or devices conforming to the PKCS#11 interface), which are defined in the [conf.xml](#). In the example above, the key is stored in softHSM - the example configuration file distributed with OpenDNSSEC defines this as being the software emulation of an HSM distributed as part of OpenDNSSEC.
- <Standby> **Experimental** (we are currently not supporting offline HSM, which is needed to get the security level needed to fulfill the idea behind standby keys. Will be fixed in a future version) Determines the number of standby keys held in the zone. These keys allow the currently active key to be immediately retired should it be compromised, so enhancing the security of the system. (Without an standby key, additional time is required to allow information about the new key to reach validator caches - see <http://tools.ietf.org/html/draft-ietf-dnsop-dnssec-key-timing-02> for timing details.)
- <ManualRollover/> is an optional tag. This tag indicate that the key rollover will only be initiated on the command by the operator. There is still a second step for the KSK, where the key needs to be published to the parent before the rollover is completed. Read more in the chapter "Running OpenDNSSEC". The ZSK rollover will although be fully automatic if this tag is not present.

### Zone-Signing Keys

Parameters for zone-signing keys are held in the <ZSK> section, and have the same meaning as for the KSK:



```
...
  <ZSK>
    <Algorithm length="1024">8</Algorithm>
    <Lifetime>P90D</Lifetime>
    <Repository>softHSM</Repository>
    <!-- <Standby>1</Standby>
(Experimental) -->
  </ZSK>
```

The ZSK information completes the contents of the <Keys> section.

```
...
</Keys>
```

## Zone Information

General information concerning the zones can be found in the <Zone> section:

```
...
  <Zone>

  <PropagationDelay>PT9999S</PropagationDelay>

  <SOA>
    <TTL>PT3600S</TTL>
    <Minimum>PT3600S</Minimum>
    <Serial>unixtime</Serial>
  </SOA>
</Zone>
```

- `<PropagationDelay>` is the amount of time needed for information changes at the master server for the zone to work its way through to all the secondary nameservers.
- The `<SOA>` element gives values of parameters for the SOA record in the signed zone.

**⊖ These values will override values set for the SOA record in the input zone file.**

The values are:

- `<TTL>` - TTL of the SOA record.
- `<Minimum>` - value for the SOA's "minimum" parameter.
- `<Serial>` - the format of the serial number in the signed zone. This is one of:
  - counter - use an increasing counter (but use the serial from the unsigned zone if possible)
  - datecounter - use increasing counter in YYYYMMDDxx format (xx is incremented within each day)
  - unixtime - the serial number is set to the "Unix time" (seconds since 00:00 on 1 January 1970 (UTC)) at which the signer is run.
  - keep - keep the serial from the unsigned zone (do not resign unless it has been incremented)

## Parent Zone Information

If a DNSSEC zone is in a chain of trust, digest information about the KSKs used in the zone will be stored in DS records in the parent zone. To properly roll keys, timing information about the parent zone must be configured in the `<Parent>` section:

```

...
  <Parent>

  <PropagationDelay>PT9999S</PropagationDelay>

  <DS>
    <TTL>PT3600S</TTL>
  </DS>
  <SOA>
    <TTL>PT3600S</TTL>
    <Minimum>PT3600S</Minimum>
  </SOA>
</Parent>

```

- `<PropagationDelay>` is the interval between the time a new KSK is published in the zone and the time that the DS record appears in the parent zone.

- The <DS> tag holds information about the DS record in the parent. It contains a single element, <TTL>, which should be set to the TTL of the DS record in the parent zone.
- <SOA> gives information about parameters of the parent's SOA record, used by KASP in its calculations. As before, <TTL> is the TTL of the SOA record and <Minimum> is the value of the "minimum" parameter.

This is the last section of the policy specification, so the next element is the policy closing tag:

```
...  
</Policy>
```

If there are any additional policies, they could be entered here, starting with <Policy> and ending with </Policy>. However, in this case there are no additional policies, so the file is ended by closing the <KASP> tag:

```
</KASP>
```

## zonelist.xml

The list of zones that OpenDNSSEC will sign is held in the zone list file `/etc/opendnssec/zonelist.xml`. As well as listing the zones, it also specifies the policy used to sign the zones.

This section explains the parameters of the zone list by referring to the example `zonelist.xml` file supplied with the OpenDNSSEC distribution.

Date/time durations are as specified [here](#).

### On this Page

- [Elements of the zonelist.xml file](#)
  - [Preamble](#)
  - [Zone List](#)
  - [Zones](#)
- [Zone List File Creation](#)
- [File Names](#)

## Elements of the zonelist.xml file

### Preamble

```
<?xml version="1.0" encoding="UTF-8"?>  
<!-- $Id: zonelist.xml.in 1147 2009-06-24  
12:18:17Z jakob $ -->
```

Each XML file starts with a standard element "<?xml...". As with any XML file, comments are included between the delimiters "<!--" and "-->".

## Zone List

```
<ZoneList>
```

The enclosing element of the XML file is the element `<ZoneList>` which, with the closing element `</ZoneList>`, brackets the list of zones.

## Zones

Each zone is defined by a `<Zone>` element:

```
...  
<Zone name="example.com">
```

The mandatory attribute "name" identifies the zone. Each zone within the zone list must have a unique name. Use "." when signing the root.

### Policy

```
...  
<Policy>default</Policy>
```

The first element of the `<Zone>` tag is `<Policy>`, which identifies the policy used to sign the file. Policies are defined in the [kasp.xml](#) file, and the name in this element must be that of one of the defined policies.

### Configuration

Information from the Enforcer to the Signer Engine is passed via a special signer configuration file, the name of which is given by the `<SignerConfiguration>` section of the zone definition:

...

```
<SignerConfiguration>/var/opendnssec/signconf/example.com.xml</SignerConfiguration>
```

(Note that this file is a temporary file that passed between OpenDNSSEC components and is not intended to be edited by users.)

### Adapters

The next part of the zone element specifies from where OpenDNSSEC gets the zone data and to where the signed data is put.

...

```
<Adapters>
  <Input>
    <Adapter
type="File">/var/opendnssec/unsigned/example.com</Adapter>
  </Input>
  <Output>
    <Adapter
type="DNS">/etc/opendnssec/addns.xml</Adapter>
  </Output>
</Adapters>
```

The <Adapters> element comprises an <Input> and <Output> element which (fairly obviously) identify the input source and output sink of the data.

Within each element is a tag defining the type of data source/sink and its parameters. There is type="File", which takes as its only data the name of the input unsigned file, or output signed zone file. And there is type="DNS", which takes a configuration file as its data. The DNS adapter configuration file is described in more detail [here](#).

```
...  
</Zone>
```

The `</Zone>` tag closes the definition of the zone. As indicated above, one or more zones can be defined in this file.

```
</ZoneList>
```

The `</ZoneList>` element closes the file.

### Zone List File Creation

For a small number of zones, the zone list file can be easily edited by hand. Where the number of zones is large - for example, ISPs serving thousands of customers - the intention is that the file be generated by the zone manager's systems using e.g. the `ods-ksmutil zone add` command.

### File Names

As can be seen in the example above, a number of elements that specify file names (`<SignerConfiguration>`, `<Adapter>/<Input>` and `<Adapter>/<Output>`) include the zone name in the name of the file.

✓ Where there are multiple zones, this is strongly recommended as a way of avoiding confusion.

### addns.xml

Per zone you can configure the zone transfer settings. One file can be used for multiple zones, just point to the same file in the zone list file `/etc/opendnssec/zonelist.xml`.

This section explains the parameters of the DNS adapter configuration by referring to the example `addns.xml` file supplied with the OpenDNSSEC distribution.

#### On this Page

- [Elements of the addns.xml file](#)
  - [Preamble](#)
  - [TSIG](#)
  - [Inbound](#)
  - [Outbound](#)
  - [Postamble](#)

### Elements of the addns.xml file

## Preamble

```
<!-- $Id: addnsconf.xml.in 2735 2010-01-28
14:11:27Z matthijs $ -->
```

Each XML file starts with a standard element "<?xml...". As with any XML file, comments are included between the delimiters "<!--" and "-->".

```
<Adapter>
  <DNS>
```

The enclosing element of the XML file is the element `<Adapter>` which, with the closing element `</Adapter>`, brackets the zone transfer configuration. Because we are configuring the DNS Adapter, the element `<DNS>` is used.

## TSIG

```
...
  <TSIG>
    <Name>secret.example.com</Name>
    <!--
http://www.iana.org/assignments/tsig-algor
ithm-names -->
    <Algorithm>hmac-sha256</Algorithm>
    <!-- base64 encoded secret -->

  <Secret>sw0nMPCswVbes1tmQTmlpcMmpNRK+oGMYN
+qKNR/BwQ=</Secret>
  </TSIG>
```

The first element of the `<DNS>` tag is `<TSIG>`, which is the dedicated protection mechanism used. TSIG requires three elements:

- `<Name>` is the name of the TSIG.

- *<Algorithm>* specifies the algorithm used.
- *<Secret>* is the base64 encoded secret.

You can list zero or more TSIGs.

## Inbound

The inbound DNS configuration is bracketed in the enclosing element *<Inbound>*, and the closing element *</Inbound>*.

```
...  
<Inbound>
```

### Requesting Zone Transfers

```
...  
<RequestTransfer>  
  <!-- EXAMPLE: send request to 1.2.3.4  
on the default port 53 -->  
  <Remote>  
    <Address>1.2.3.4</Address>  
  </Remote>  
  <!-- EXAMPLE: send request to  
dead:beef::1 on port 5353, TSIG signed with  
secret.example.com -->  
  <Remote>  
    <Address>dead:beef::1</Address>  
    <Port>5353</Port>  
    <Key>secret.example.com</Key>  
  </Remote>  
</RequestTransfer>
```

*<RequestTransfer>* is used to hold a list of master name servers where this zone can request zone transfers. The name servers are given by the *<Remote>* element:



- `<Address>` is a required element that stores the IPv4 or IPv6 address of the master name server.
- `<Port>` is an optional element that specifies the port to use. If not provided, the port is defaulted to 53.
- `<Key>` refers to a configured TSIG key name. The TSIG must be provided in the same file. If no key is given, TSIG will not be used for this name server.

You can list multiple master name servers. In the example above, the zone transfer can be requested at 1.2.3.4 port 53 with no TSIG, or at dead:beef::1 port 5353 with the secret.example.com TSIG.

### Receiving Notices

```

...
  <!-- Allow NOTIFY messages from host -->
  <AllowNotify>
    <!-- EXAMPLE: allow notifies from
1.2.3.4 -->
    <Peer>
      <Prefix>1.2.3.4</Prefix>
    </Peer>
  </AllowNotify>

```

`<AllowNotify>` lists the name servers that may notify OpenDNSSEC that there is a new version of the zone. Usually, the master name server will also provide the NOTIFY messages. In that case, the address and TSIG key from `<RequestTransfer>` can be copied here. Separating this in the configuration file allows for more flexible environment setups. Here, you can list a number of `<Peer>` elements:

- `<Prefix>` is a required element that stores an address or address prefix.
- `<Key>` refers to a configured TSIG key name. The TSIG must be provided in the same file. If no key is given, TSIG will not be used for this server.

```

...
  </Inbound>

```

- ✓ If no NOTIFY messages are being received, OpenDNSSEC will request a new zone transfer after the SOA REFRESH value has passed in time. If a zone transfer has failed, OpenDNSSEC will retry after the SOA RETRY value has passed in time.

## Outbound


Similar to `<Inbound>`, the outbound DNS configuration goes between `<Outbound>` and `</Outbound>`.

```
...  
<Outbound>
```

#### Providing Zone Transfers

```
...  
<!-- Provide XFR to host -->  
<ProvideTransfer>  
  <!-- EXAMPLE: provide XFR to 1.2.3.5  
with key secret.example.com -->  
  <Peer>  
    <Prefix>1.2.3.5</Prefix>  
    <Key>secret.example.com</Key>  
  </Peer>  
</ProvideTransfer>
```

`<ProvideTransfer>` allows you to configure a list of secondary name servers that can pick up the signed zone through a zone transfer. One or more `<Peer>` elements is used to do that. In this example, one secondary name server is configured: The server with address 1.2.3.5 may request a zone transfer if it is correctly signed with the secret.example.com TSIG key.

 If OpenDNSSEC acts as a secondary for certain zones (e.g., it has Inbound DNS adapters configured), zones may expire if inbound zone transfers are failing. This happens if the SOA EXPIRE value has passed in time after the latest successful zone transfer. If a zone is expired, OpenDNSSEC will stop providing signed zone transfers, but it will still serve normal queries, for troubleshooting purposes.

#### Sending Notices

```
...
  <!-- Send NOTIFY messages to host -->
  <Notify>
    <!-- EXAMPLE: send NOTIFY to 1.2.3.5 on
the default port 53 -->
    <Remote>
      <Address>1.2.3.5</Address>
    </Remote>
  </Notify>
```

For the same reasons as with the inbound DNS configuration, sending notifies and zone transfers has been split up in the configuration to be more flexible. Here, OpenDNSSEC will send a NOTIFY to the server at 1.2.3.5 at port 53, not using TSIG. Again, more than one servers can be configured.

```
...
</Outbound>
```

## Postamble

```
...
</DNS>
</Adapter>
```

The `</DNS>` element closes the DNS adapter configuration. The `</Adapter>` element closes the file.

### signconf.xml

There are xml files for each of the zones that the user wants to sign. Those define the API between the Enforcer and the Signer Engine. The Enforcer creates these files while implementing the policies and key management and the Signer Engine reads them when signing zones.

**i** These files are should not be created or modified by users. However inspection of the content can be useful to troubleshoot problems.

The location of these files can be found in zonelist.xml, but we refer to signconf.xml if we speak about the general signer configuration file. The actual location of these files can be found in zonelist.xml.

Date/time durations are as specified [here](#).

#### On this Page

- [Signer Configuration File](#)
  - [Preamble](#)
  - [Configuration per zone](#)
  - [Signatures](#)
  - [Authenticated Denial of Existence](#)
  - [Key Information](#)
  - [Source of Authority](#)

### Signer Configuration File

## Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: signconf.xml.in 3061 2010-03-16
19:23:51Z jakob $ -->
```

Each XML file starts with a standard element "<?xml...". As with any XML file, comments are included between the delimiters "<!--"

and "-->".

## Configuration per zone

```
<SignerConfiguration>
```

The enclosing element of the XML file is the element <SignerConfiguration?> which, with the closing element </SignerConfiguration>, brackets one signer configuration.

```
...
<Zone name="opendnssec.org">
```

Each zone is included in the <Zone>...</Zone> elements. Each zone has a "name" attribute giving the name of

the zone.

## Signatures

The next section of the file is the Signatures section, which lists the parameters for the signatures created using the policy. This is directly copied from the [KASP policy file](#).

```
...
<Signatures>
  <Resign>PT2H</Resign>
  <Refresh>P3D</Refresh>
  <Validity>
    <Default>P7D</Default>
    <Denial>P7D</Denial>
  </Validity>
  <Jitter>PT12H</Jitter>

<InceptionOffset>PT300S</InceptionOffset>
</Signatures>
```

For more information on the meaning of the elements and their elements, take a look at [KASP policy file](#).

Authenticated Denial of Existence

Authenticated denial of existence - proving that domain names do not exist in the zone - is handled by the <Denial> section, as shown below:

...

```
<Denial>
  <NSEC3>
    <OptOut/>
    <Resalt>P100D</Resalt>
    <Hash>
      <Algorithm>1</Algorithm>
      <Iterations>5</Iterations>
      <Salt length="8"/>
    </Hash>
  </NSEC3>
</Denial>
```

<Denial> includes one element, either <NSEC3> (as shown above) or <NSEC>.

### NSEC3

- <NSEC3> tells the signer to implement NSEC3 scheme for authenticated denial of existence (described in [RFC 5155](#)). The elements are:
- <OptOut/> - if present, enable "opt out". This is an optimisation that means that NSEC3 records are only created for authoritative data or for secure delegations; insecure delegations have no NSEC3 records. For zones where a majority of the entries are delegations that are not signed - typically TLDs during the take-up phase of DNSSEC - this reduces the number of DNSSEC records in the zone.
- <Resalt> is the interval between generating new salt values for the hashing algorithm.
- <Algorithm>, <Iterations> and <Salt> are parameters to the hash algorithm, described in [RFC 5155](#).

### NSEC

Should, instead, NSEC be used as the authenticated denial of existence scheme, the <Denial> element will contain the single element <NSEC/> - there are no other parameters.

## Key Information

Parameters relating to keys can be found in the <Keys> section.

```
...
<Keys>
```

### Common Parameters

The section starts with a common parameter, TTL:

```
...
<TTL>PT3600S</TTL>
```

<TTL> is the time-to-live value for the DNSKEY resource records.

### Keys

Each key has a number of elements so the signer knows how the keyset should be used and published.

Those are held in the <Key> section:

```
...
<Key>
  <Flags>257</Flags>
  <Algorithm>5</Algorithm>

<Locator>DFE7265B783F418685380AA784C2F31D<
/Locator>
  <Publish/>
  <KSK/>
  <ZSK/>
</Key>
```

- <Flags> tells the signer what value it should set on this key when publishing the corresponding DNSKEY resource record.
- <Algorithm> determines the algorithm used for the key (the numbers reserved for each algorithm can be found in the appropriate [IANA registry](#)).
- <Locator> stores the CKA\_ID of the key, e.g. the location of the physical key.

- `<KSK/>` - if present, use key as KSK. If the DNSKEY RRset requires a new signature, use this key to sign the DNSKEY RRset.
- `<ZSK/>` - if present, use key as ZSK. If another RRset (non-DNSKEY) require a new signature, use this key to sign that RRset.
- `<Publish/>` - if present, publish the corresponding DNSKEY resource record in the zone. The Public Key RDATA should be retrieved from the HSM using the CKA\_ID (`<Locator>`).

**i** If both `<KSK/>` and `<ZSK/>` are used on the same `<Key/>`, it will be used to generate new signatures for both DNSKEY and non-DNSKEY RRsets, effectively making it a Combined Signing Key (CSK).

The keyset is closed with the tag:

```
...
</Keys>
```

## Source of Authority

When automating DNSSEC, the SOA record needs to be adjusted from time to time. The necessary parameters can be found in the `<SOA>` section:

```
...
<SOA>
  <TTL>PT3600S</TTL>
  <Minimum>PT3600S</Minimum>
  <Serial>unixtime</Serial>
</SOA>
```

**⊖** These values will override values set for the SOA record in the input zone file.

The values are:

- `<TTL>` - TTL of the SOA record.
- `<Minimum>` - value for the SOA's MINIMUM RDATA element.
- `<Serial>` - the format of the serial number in the signed zone. This is one of:
  - counter - use an increasing counter (but use the serial from the unsigned zone if possible)
  - datecounter - use increasing counter in YYYYMMDDxx format (xx is incremented within each day)
  - unixtime - the serial number is set to the "Unix time" (seconds since 00:00 on 1 January 1970 (UTC)) at which the signer is run.
  - keep - keep the serial from the unsigned zone (do not resign unless it has been incremented)



This is the last section of the signconf specification, so the next element is the zone closing tag:

```
...  
</Zone>
```

and the file is ended by closing the <SignerConfiguration> tag:

```
</SignerConfiguration>
```

## Migrating zone fetcher to DNS adapters

OpenDNSSEC version 1.4 introduces DNS adapters. With the introduction of DNS adapters, OpenDNSSEC is able to read unsigned zones directly from AXFR or IXFR, and to output signed AXFR and IXFRs. Version 1.3 also understood incoming, unsigned AXFRs, thanks to the zone fetcher process. The zone fetcher had its own configuration file, [zonefetch.xml](#). With the introduction of the DNS adapters, a zone fetcher is not needed anymore and is removed in version 1.4. Version 1.4 also provides IXFR and outgoing zone transfers, and requires more configuration options. A new configuration file, [addns.xml](#), is required to configure the DNS adapters.

If you upgrade OpenDNSSEC from 1.3 or lower to 1.4 or higher, and you used the zone fetcher, you will have to migrate the configuration. This page explains how you do that.

### Changes in conf.xml

In 1.3 and earlier, the zone fetcher configuration filename had to be specified in [conf.xml](#) in the <ZoneFetchFile> element. In version 1.4, this element must be removed.

In 1.3 and earlier, the zonefetch.xml file contains an element <NotifyListen>: this defines which interface and port the system must bind to listen NOTIFY messages on. For example, to let the zone fetcher listen on the IPv4 address 192.0.2.100 on port 53, you would have configured the zone fetcher like this:

```
...  
<NotifyListen>  
  
<IPv4>192.0.2.100</IPv4><Port>53</Port>  
</NotifyListen>
```

With OpenDNSSEC 1.4, the signer can listen to more than NOTIFY messages. It also accepts zone transfer requests and 'regular queries' like SOA, DNSKEY, and so on. The listener becomes more general and therefore the configuration now goes in conf.xml, under the <Signer> element:

```
...  
  <Listener>  
  
    <Interface><Address>192.0.2.100</Address><  
    Port>53</Port></Interface>  
  </Listener>
```

Instead of `<NotifyListen>`, the element name is made more general and is now called **<Listener>**. There is an additional element `<Interface>` to indicate this describes an interface and to support multiple listening interfaces. The configuration does not need different elements for IPv4 and IPv6 anymore, so also for an IPv6 address you would use the **<Address>** element. The **<Port>** element remains unchanged.

#### From zonefetch.xml to addns.xml

In 1.3 and earlier, all the zone fetcher configuration (except the `<NotifyListen>`) is encapsulated within the elements `<Zonefetch><Default>`. In 1.4, the DNS adapter configuration is encapsulated in **<Adapter><DNS>**. The **<TSIG>** part is unchanged and can be copied straightforward.

Because the zone fetcher only was able to do inbound zone transfers, the `<RequestTransfer>` element was immediately listed under the `<Default>` element. The DNS Adapter configuration is for both inbound and outbound zone transfers, so it should now be encapsulated within the **<Inbound>** element. Again, suppose you had the following configuration in `zonefetch.xml`:

```
<Zonefetch>

...

<Default>
  <RequestTransfer>
    <IPv4>192.0.2.100</IPv4>
    <Port>53</Port>
    <Key>secret.example.com</Key>
  </RequestTransfer>

...

</Default>
</Zonefetch>
```

You would have to translate that into the following configuration in addns.xml:

```
<Adapter>
  <DNS>
    <TSIG>
      <Name>secret.example.com</Name>

<Algorithm>hmac-sha256</Algorithm>

<Secret>sw0nMPCswVbes1tmQTmlpcMmpNRK+oGMYN
+qKNR/BwQ=</Secret>
  </TSIG>

  <Inbound>
    <RequestTransfer>
      <Remote>

<Address>192.0.2.100</Address>
      <Port>53</Port>

<Key>secret.example.com</Key>
      </Remote>
    </RequestTransfer>

    ...

  </Inbound>

  ...

  </DNS>
</Adapter>
```

The <IPv4> element is renamed to **<Address>** again, which is IP version independent. **<Port>** and **<Key>** remain unchanged. The three elements are encapsulated into the element **<Remote>**, so that multiple remote zone transfer sources can be configured.

That's it! In the DNS adapter configuration you can now also configure from which sources you allow NOTIFY messages and you can configure outbound zone transfers. For more information on that, see the documentation on [addns.xml](#).

## Zone content

OpenDNSSEC can handle various formatting of the zone file, including different directives and Resource Records (RRs).

### On this Page

- [Formatting](#)
- [Supported Directives](#)
- [RR types](#)
  - [Handling of unknown RR types](#)

## Formatting

The zone file can be formatted in many ways including multi-lined RR, comments, etc.

## Supported Directives

As defined in RFC 1035 the following directives are supported by OpenDNSSEC:

<b>\$ORIGIN</b> <code>example.com.</code>	What origin to use.
<b>\$TTL</b> <code>1h3m</code>	The default TTL to use. Treated as seconds, if no suffix is used: s, m, h, d, w, S, M, H, D, W
<b>\$INCLUDE</b> <code>&lt;path&gt;</code>	Include a file from a given path

## RR types

OpenDNSSEC support all of the RR specified by [IANA](#), with some exceptions:

<b>Not supported</b>	ATMA, APL, EID, NIMLOC, HIP, SINK, NINFO, RKEY, TA
<b>Obsoleted</b>	MD, MF, WKS, GPOS, SIG, KEY, NXT, A6, and NSAP-PTR
<b>Not allowed in master</b>	NULL, OPT, TKEY, TSIG, IXFR, AXFR, MAILB, MAILA, *

## Handling of unknown RR types

But OpenDNSSEC does handle unknown RR types in accordance with [RFC3597](#) e.g:

```
example.com.      IN          TYPE1
# 4 0A000001
```

## Migrating to OpenDNSSEC

It is possible to migrate a DNSSEC signed zone over to OpenDNSSEC. How to migrate your DNSSEC signed zone over to OpenDNSSEC really depends on how your current solution looks like.

The zone data is no problem. Just place a copy of the unsigned zone in the directory for unsigned zones. But the trick is to maintain the private and public keys.

When moving from one system to another, you need to exchange public keys between them in order to always have a valid DNSSEC state. There are three possible solutions:

- Export the keys
- Prepublish DNSKEY record
- Start fresh

### On this Page

- [Export the keys](#)
  - [On disc](#)
  - [On a smartcard with no PKCS#11 interface](#)
  - [On an HSM](#)
  - [Add the keys to OpenDNSSEC](#)
- [Prepublish DNSKEY record](#)
- [Start fresh](#)

## Export the keys

One solution is to move the key pairs and make them accessible by OpenDNSSEC. The goal is to have the key pairs available to the system using PKCS#11.

The key pairs can e.g. be stored:

- on disc (e.g. BIND .private-key format)
- on a smartcard with no PKCS#11 interface
- in an HSM

### *On disc*

When the key pairs are stored on disc, it means that you have access to files containing the key pairs. The key pairs can be imported into your new HSM using the PKCS#11 API or any tool available from your HSM vendor.

The BIND .private-key file can be converted into the PKCS#8 file format using the tool available with SoftHSM. If you have another file format, then OpenSSL probably can help you to convert it into the PKCS#8 file format.

```
softhsm-keyconv --topkcs8 --in  
Kexample.com.+005+42952.private --out  
key.pem
```

- **--topkcs8**, To indicate that you want to convert from BIND .private-key format to PKCS#8.
- **--in <path>**, The path to the BIND .private-key file.
- **--out <path>**, A path to the temporary PKCS#8 file.

The PKCS#8 file can then be imported into the SoftHSM token (if you are using SoftHSM as your HSM).

```
softhsm --import key.pem --slot 1 --pin  
123456 --label A2 --id A2
```

- **--import <path>**, The path to the PKCS#8 file that you want to import. This should point to the temporary file that you created in the previous step.
- **--slot <number>**, The key should be imported to a token. Indicate which slot it is connected to.
- **--pin <PIN>**, Provide the PIN so that we can login to the token.
- **--label <text>**, Choose an arbitrary text string. Not used by OpenDNSSEC.
- **--id <hex>**, Choose an ID of the new key pair. The ID is in hexadecimal with a variable length. It must not collide with an existing key pair.

#### ***On a smartcard with no PKCS#11 interface***

Just connect a smartcard reader to your system and insert your smartcard. Then use **opensc** and **pcscd** to give it a PKCS#11 interface. Remember to protect the location where you have your smartcard reader, since the smartcard needs to be online.

#### ***On an HSM***

You can either move the HSM to the new server and install it there. Or some vendors may have some functionality to export/transfer the key pairs.

#### ***Add the keys to OpenDNSSEC***

Once you have the key pairs available on the system via PKCS#11, then you must add them to OpenDNSSEC. Give this command before you start OpenDNSSEC. Also make sure that the zone is properly configured with OpenDNSSEC.

```
ods-ksmutil key import --cka_id <CKA_ID>
--repository <repository> --zone <zone>
--bits <size> --algorithm <algorithm>
--keystate <state> --keytype <type> --time
<time>
```

- **--cka\_id <CKA\_ID>**, Each key object in the HSM has an ID, the CKA\_ID attribute. The private and public key object must have the same ID in order for OpenDNSSEC to find them. The CKA\_ID of the key pair to import is indicated, in hexadecimal, by using this option. E.g. *A2*
- **--repository <repository>**, The name of the repository, from conf.xml. E.g. *softHSM1*
- **--zone <zone>**, The name of the zone. E.g. *example.com*
- **--bits <size>**, The key length, E.g. *1024*
- **--algorithm <algorithm>**, The algorithm. E.g. *5* or *7*
- **--keystate <state>**, The key state. E.g. *active* or *ready*
- **--keytype <type>**, The key type. *KSK* or *ZSK*
- **--time <time>**, The time stamp when the key entered the given state. So that OpenDNSSEC know when to change the state. E.g. *200910301000*
- **--retire <retire>**, Optional. If you set the state to active, then you may set the time when the key should be retired. E.g. *201001010000*. Otherwise will OpenDNSSEC use the key lifetime from the KASP.

The difference between active and ready is:

- **active:** Will make the key active, thus used for signing. If there already is an active key, then you will have two of them. If this is not desired, then make sure to give this command right after setup and before you start the system.
- **ready:** The key will only be published in the zone. It will become active in a future rollover, if the key parameters match the policy.

## Prepublish DNSKEY record

A second alternative, when migrating a signed zone to OpenDNSSEC, is to do a manual key rollover. When moving from one system to another, you need to exchange public keys between them in order to always have a valid DNSSEC state.

The steps below will perform a manual Double-DS KSK rollover and a manual Pre-Publication ZSK rollover. There must be a period of time between each step and the system rollover; otherwise there will not be sufficient time for the information to propagate out on the Internet. The exact time depends on your setup, but it is typically between two and four weeks. Read more in the DNSSEC Key Timing draft from IETF.

1. Before the system rollover you need to:
  - Extract the DS corresponding to the KSK in the new system and publish it in the parent zone.
  - Publish the new ZSK in the old system.
  - Publish the old ZSK in the new system.
2. System rollover:
  - Re-delegate the zone in the parent zone.
3. After the system rollover you need to:



- Remove the old DS from the parent zone.
- Remove the new ZSK from the old system.
- Remove the old ZSK from the new system.

## Start fresh

A third solution is to start fresh. Remove any DS records from the parent zone. Stop signing your zone when the DS records are removed from the DNS caches. It is safe to remove the public keys from your zone when the signatures are not present in any DNS caches. Then transfer the zone over to OpenDNSSEC. And let OpenDNSSEC start signing it again.

 Your zone will not be secured by DNSSEC during this transfer.

## External Auditing of Zones

In the 1.4 release of OpenDNSSEC the auditor component has been removed. This component was only lightly used in deployed systems and its removal means that OpenDNSSEC no longer depends on Ruby.

There are a number of third party products available that specialise in generalised zone auditing. It is anticipated that users wishing to audit their zones will use such a product and some recommendations are made below.

### On this Page

- [Workflow](#)
- [Validation Tools](#)
  - [validns](#)
  - [credns](#)

## Workflow

With no 'in-built' auditing function in OpenDNSSEC one option is to post process the zone. It is possible to do this using the NotifyCommand in conf.xml. An example workflow would be:

- Configure the signer put the signed zone files into an intermediate directory e.g. /signed\_but\_unchecked
- Configure the NotifyCommand in conf.xml to call a script which does the following:
  - Call the validation tool of choice and parse the reply
  - On success: copy the signed zone file into a directory for distribution to a nameserver e.g. /signed\_and\_checked and notify the nameserver
  - On failure: e.g. send an e-mail to prompt further investigation

An alternative solution is that the signed zones from OpenDNSSEC may be transferred to a hidden master and a validation tool may be run before the zones are distributed to the slave servers.

## Validation Tools

### validns

<http://www.validns.net/>


### credns

<http://www.nlnetlabs.nl/projects/credns/>

## Plugins

One plugin is currently provided with OpenDNSSEC to aid users in automating their DNSSEC deployment:

- [simple-dnskey-mailer](#)
  - [Description](#)
  - [Configuration](#)
  - [Running](#)
- [eppclient](#)

 The plugins provided are examples only and should be reviewed by users before use.

### simple-dnskey-mailer

#### Description

This makes it easier to notify the OpenDNSSEC operator that a new set of keys needs to be sent to the parent zone for publication, as needed when you perform key rollovers on the KSK. This plugin is just a simple example on how to use this API to send an e-mail with the keys.

#### Configuration

In the plugins directory in the source code of OpenDNSSEC you have the directory `simple-dnskey-mailer` containing the shell script `simple-dnskey-mailer.sh`. Edit this script and change the e-mail address you want to be notified about the new keyset.

In [conf.xml](#) you have the `<DelegationSignerSubmitCommand>` tag where you add the path to where you place this script in your system. Don't forget to uncomment the tag, and set the executable flag on the script.

#### Running

When the Enforcer decides that it has an updated set of keys for publication at the parent level, it will run the script with the complete set of keys that you need to publish. When you have published the whols set of keys, any publication of new keys will have to be reported back by the operator to the Enforcer so that it knows that the parent has published the new keys. This will complete the key rollover. You do this by issuing the `ds-seen` comm and like this:

```
ods-ksmutil key ds-seen --zone example.com
--keytag 12345
```

or

```
ods-ksmutil key ds-seen --zone example.com
--cka_id a6f66e07781469df81e6c908bf22b168
```

You find the `cka_id` by listing the keys for your zone in verbose mode.

## eppclient

**i** This plugin is no longer maintained and has been removed from the OpenDNSSEC source directory. It is still available for reference at trunk/contrib but there are know issues with the use of libcurl in this implementation.

## Running OpenDNSSEC

This section describes how to start OpenDNSSEC and the operations used to manage and monitor the system.

The details of the command utilities shown below can be found [here](#).

### On this Page

- [Before starting OpenDNSSEC for the first time](#)
- [HSM login](#)
- [Starting / Stopping the system](#)
- [Uploading a Trust Anchor \(Publishing DS record to the parent\)](#)
- [Key Management](#)
- [Zone Management](#)
- [Updating the KASP policy](#)
- [Logging](#)

**✓** All directories are prepared by the build script and are set to be owned by root, so all commands in the default configuration must also be run by root. To change this, alter the configuration or privileges on the files and directories.

### Before starting OpenDNSSEC for the first time

Before you run the system for the first time you must import your policy and zone list into the database using the following command:

```
ods-ksmutil setup
```

After running this the first time, you will be ready to run OpenDNSSEC with an empty set of zones. On the other

hand, if this command is run on an existing database, then will all meta-information about the zones be lost. The keys would then still exist in HSMs, so you should not forget to clean them up.

## HSM login

If the PIN to the HSM is not in conf.xml, then it must be entered by the user with the following command:

```
ods-hsmutil login
```

If the PIN has not been entered before OpenDNSSEC is started, then the daemons will not start.

There are strict privilege control on the shared memory segment containing the PIN, so remember to run commands and processes with the correct privileges. If the daemons are dropping privileges, then you also need to run ods-hsmutil with the correct user and group.

```
$ sudo -u <user> -g <group> ods-hsmutil  
login
```

If you experience trouble please see [Troubleshooting](#).

## Starting / Stopping the system

OpenDNSSEC consist of two daemons, *ods-signerd* and *ods-enforcerd*. To start and stop them use the following commands:

```
ods-control start
```

A proper-looking response to this commands is:

```
Starting enforcer...  
OpenDNSSEC ods-enforcerd started (version  
1.4.0), pid 31937  
Starting signer...  
OpenDNSSEC signer engine version 1.4.0  
Engine running.
```

At any time, you can stop OpenDNSSEC's daemons orderly with:

```
ods-control stop
```

After this, your logs should contain messages like:

```
Stopping enforcer...
Stopping signer engine...
Engine shut down.
```

### Uploading a Trust Anchor (Publishing DS record to the parent)

Your zone will be signed, once you have setup the system and started it. When you have verified that the zone is operational and working, it is time to upload the trust anchor to the parent zone. The Enforcer is waiting for zone to be connected to the trust chain before considering the KSK to be active.

```
ods-ksmutil key list --verbose
```

```
Keys:
```

```
Zone:                                     Keytype:
State:      Date of next transition:  CKA_ID:
Repository:                                     Keytag:
example.com                                     ZSK
active      2010-10-15 06:59:28
92abca348b96aaef42b5bb62c8daffb0  softHSM2
28743
example.com                                     KSK
ready      waiting for ds-seen
9621ca39306ce050e8dd94c5ab837001  softHSM1
22499
```

1. Export the public key either as DNSKEY or DS, depending on what format your parent zone wants it in. See the section, [Export the public keys](#), on how to get the key information.

✓ This step can be automated or semi-automated by placing a command in the <DelegationSignerSubmitCommand> tag. This should point to a binary which will accept the required key(s) as DNSKEY RRs on STDIN.

2. Notify the Enforcer when you can see the DS RR in your parent zone. You usually give the keytag to the Enforcer, but if there are KSKs with the same keytag then use the CKA\_ID.

```
ods-ksmutil key ds-seen -z example.com -x 22499
```

or

```
ods-ksmutil key ds-seen -z example.com -k
9621ca39306ce050e8dd94c5ab837001
```

```
Result:
Found key with CKA_ID 9621ca39306ce050e8dd94c5ab837001
Key 9621ca39306ce050e8dd94c5ab837001 made active
```

And you will see that your KSK is now active:

```
ods-ksmutil key list

Keys:
Zone:                Keytype:      State:      Date of next
transition:
example.com          ZSK           active      2010-10-15
07:20:53
example.com          KSK           active      2010-10-15
07:31:03
```

## Key Management

The details of common key management activities are described on the [Key Management](#) page - these include:

- Configuring the system to only make keys active once they have been backed up.
- Exporting public keys.
- Performing manual key rollovers.

## Zone Management

The details of common zone management activities are described on the [Zone Management](#) page - these include:

- Adding / Removing zones
- Updating an unsigned zone

## Updating the KASP policy

When you make changes to a policy or add a new policy in kasp.xml you must update the changes to the database.

```
ods-ksmutil update kasp
```

## Logging

Details of logs produced by the system can be found on the [Logging](#) page.

## Key Management

This sections describes common key management activities of OpenDNSSEC.

The details of the command utilities shown below can be found [here](#).

### On this Page

- [Pre-generating keys](#)
- [Marking keys as backed up](#)
- [Export the public keys](#)
- [Key rollovers](#)
  - [First step](#)
  - [Second step for KSKs - Publish the DS to the parent](#)
  - [Key rollovers on exact dates](#)

## Pre-generating keys

OpenDNSSEC will generate keys on the fly as required to perform the key rollovers specified by the policy; the timings of this generation will not always be easy to predict. However there may be situations where users would prefer to pre-generate a pool of keys (for example, some high availability setups; or just to have a more deterministic system). This is possible with the following command:

```
> ods-ksmutil key generate --policy  
<policy> --interval <interval>
```

The interval specifies how long the pool of created keys should last for. The interval should be given careful

thought as depending on the policy it may result in the generation of a large number of keys (the command will report the number to be generated and then prompt for confirmation). It is advisable to backup the keys after generating keys in this way.

## Marking keys as backed up

You can configure the system to only make keys active once they have been backed up. This is done by editing the [conf.xml](#) file. The user must do backups and then notify OpenDNSSEC about this, so that the key rollover process can continue. The keys must be backed up regularly, because OpenDNSSEC is generating new keys prior to a rollover.

1. First prepare the backup by telling the Enforcer that you want to do backup of the keys. This is so that keys generated after you have done your backup won't accidentally be marked as backed up.

For all of the repositories:

```
ods-ksmutil backup prepare
```

or a single repository:

```
ods-ksmutil backup prepare --repository <repository>
```

2. Then you can safely do your backups. **Please read the documentation of your HSM for instructions on how to do backups.**


When you are done, then notify the Enforcer about this:


For all of the repositories:

```
ods-ksmutil backup commit
```

or a single repository:

```
ods-ksmutil backup commit --repository <repository>
```

 The command `ods-ksmutil backup done` will mark your keys as backed up in one step. This means that keys may have been generated between you doing the backup and giving the command. Thus accidentally marking them as backed up. This command is deprecated and should not be used, unless you make sure to stop the Enforcer when doing your backup.

 Backups are specified by way of a repository option in `conf.xml`:



```
<RequireBackup/>
```

If you decide you want to change this facility, you should edit conf.xml accordingly, and run:

```
ods-ksmutil update conf
```

It will report something along the lines of:

```
RequireBackup set.
```

### Export the public keys

You need to publish your key to the parent or to interested parties. If you are doing a key rollover, then only the ready KSK should be exported. The following command will extract the trust anchors:


```
ods-ksmutil key export --zone example.com  
[--keystate READY]  
ods-ksmutil key export --zone example.com  
--ds [--keystate READY]
```

What you get in return is the DNSKEY or DS in BIND-format.

### Key rollovers

#### First step

 This step is not needed for a scheduled rollover.

 The rollovers are done automatically according to the policy of the zone. But a **manual** keyroll over may be desired in cases of emergency, such as having lost a private key.

A manual rollover can be done using the *ods-ksmutil* command like this:

```
ods-ksmutil key rollover --zone example.com
--keytype KSK
```

This will roll the KSK key in a timely manner following the policy used for the zone *example.com*. If you want to roll the Zone Signing Key use *--keytype ZSK* instead.

You can also roll all the keys for zones which have a certain policy. This can be useful if you want to move all keys from one key store to another.

```
ods-ksmutil key rollover --policy default
--keytype KSK
```

### Second step for KSKs - Publish the DS to the parent

Unlike ZSKs, a KSK rollover requires a second step involving manual intervention. This intervention is a multi-stage process. First, the DNSKEY record for the new key is added to the zone. Then, after a suitable interval, the new DS record is submitted to the parent; at this point the old DS record can be removed from the parent.

The stages are:

1. Extract the DNSKEY record for the new key and publish it in the parent zone. (The new record replace any existing records for the zone being signed.) When it is time for this to happen a message with log-level "info" will be sent to syslog looking something like:

```
Mar 16 11:39:05 sion ods-enforcerd: DS Record set has changed, the
current set looks like:
Mar 16 11:39:05 sion ods-enforcerd: example.com. 3600 IN DNSKEY 257 3 7
AwEAAbcTSmphJUMKvegvDgqGspRM8IHlKZqoU5pkPaTtRLkioxGyZ5iIh4bNnvqmxlzWitt
uJ6erGUMOatMm3SXxiTr9OLaRPr86KVpo6mzejTqFicGxSp3KsrbUvyIs/V84Ry7XZBKVKV
jgppjmqeS8mRtXM4UynwTEJk0hKQfCcmkH0Q/fhZibwBVG+OcBfvTdsQbp8LZN4oVqn/vzh
nuxFkE8biTr19jmKTdtgkhp524ML59v7prg7F/+Lb20JLc8Gg6pastUeqXc/Iv2CdVyOvMW
RW39VCzyLbKpmyqB8Hc4Kn1pT5Idqc3/N3qBvXVe3HyYiZbjHGxOT6RZNNT8= ;{id =
51994 (ksk), size = 2048b}
Mar 16 11:39:05 sion ods-enforcerd: Once the new DS records are seen in
DNS please issue the ds-seen command for zone example.com with the
following cka_ids, 04260cd6eac67280cd2dea94c6e38cb7
```

The DNSKEY or DS RR can also be retrieved by using the commands in the section *Export the public keys*.

✓ This step can be automated or semi-automated by placing a command in the `<DelegationSignerSubmitCommand>` tag. This should point to a binary which will accept the required key(s) as DNSKEY RRs on STDIN.

2. When the records indicated have been seen in DNS then this can be communicated to OpenDNSSEC with the `ds-seen` command as indicated:

```
ods-ksmutil key ds-seen --zone example.com --cka_id
04260cd6eac67280cd2dea94c6e38cb7
```

3. If the DS records were not swapped, i.e. the old DS was left in the parent when the new one was added, then the `--no-retire` flag can be added to the `ds-seen` command. Then, at some later time, the old key can be retired with the command:

```
ods-ksmutil key ksk-retire --zone example.com ---cka_id
87f1385b114f9b299e6b551d728bfb
```

or

```
ods-ksmutil key ksk-retire --zone example.com
```

The former command will retire the specific key (provided the key is active, and the action will not leave the zone without any active keys). The latter command will retire the oldest active key on the zone, again provided it will not leave the zone without any active keys.

⚠ If you wish to run like this and use the `DelegationSignerSubmitCommand` hook then you will need to add the current key back into the set yourself.

### Key rollovers on exact dates

Some users want to have more control over their key rollovers and roll keys on exact dates, for example the first day of each month. To do this you need to specify that you want manual key rollovers in the `kasp.xml` configuration. Add the `<ManualRollover/>` tag to the type and key you want to roll manually.

When this is done you can add the rollover commands to a cron job, with a command like this:

```
ods-ksmutil key rollover --zone example.com
--keytype ZSK
```

## Zone Management

This section describes common Zone Management activities in OpenDNSSEC.

The details of the command utilities shown below can be found [here](#).

#### On this Page

- [Adding / Removing zones](#)
- [Updating an unsigned zone](#)

### Adding / Removing zones

Zones can be added and removed at will. If the optional parameters are not given, then it will default to the policy *default* and the */var/opendnssec/* subdirectories.

```
ods-ksmutil zone add --zone example.com
[--policy <policy> --signerconf
<signerconf.xml> --input <input> --output
<output>]
ods-ksmutil zone delete --zone example.com
```

This command will report positively with a message like:

```
zonelist filename set to
/etc/opendnssec/zonelist.xml.
SQLite database set to:
/var/opendnssec/kasp.db
Imported zone: example.com
```



Using this command thousands of times might be slow since it also writes to *zonelist.xml*. Use *--no-xml* to stop this behavior. Then export the zonelist when you are finished:

```
ods-ksmutil zonelist export > zonelist.xml
```

Alternatively, you could manually edit the *zonelist.xml* and then give the command:

```
ods-ksmutil update zonelist
```

After zones are added, they will show up in your logs as follows:

```
ods-enforcerd: Zone example.com found.  
ods-enforcerd: Policy for example.com set  
to default.  
ods-enforcerd: Config will be output to  
/var/opendnssec/signconf/example.com.xml.
```

If you opened the latter file, you would find the settings that were applied to the zone at the time this file was added.

### Updating an unsigned zone

- ✓ When you update the content of an unsigned zone you must tell the signer engine to re-read the unsigned zone file using the *ods-signer* command like this:

```
ods-signer sign example.com
```

This will also have the effect that the zone is scheduled for immediate resigning.

### Logging

Currently all logging is handled by syslog. Other logging methods may come later.

There is a lot of output from the daemons of which a lot of things right now are for debugging. We might reduce the amount of logging for later versions of OpenDNSSEC.

The signer outputs some statistics into the logs:

```
[STATS].opendnssec.org RR[count=32
time=1(sec)] NSEC[count=32 time=1(sec)]
RRSIG[new=1 reused=31 time=1(sec)
avg=1(sig/sec)] TOTAL[time=5(sec)]
```

The RR[...] block says something about reading the unsigned zone file. The line above tells us that there have been 32 RRs read and it took about one second. The count will be 0 on lines where only re-signing occurred.

The NSEC[...] block says something about the number of NSEC or NSEC3 records created on the latest run. This count will also be 0 on lines where only re-signing occurred.

The RRSIG[...] block says something about the number of created signatures. In the above example, the last time there was only created one new signature, and 31 other signatures were reused. The re-signing was finished in about a second and thus the average number of signatures created per second is in this example 1.

The TOTAL[...] block tells us how long the re-signing took, including reading the unsigned zone and adding NSEC(3) records, if applicable.

## High availability

 This page is under construction.....

Some operators choose to deploy OpenDNSSEC in a high availability configuration. This page describes some general concepts to take into consideration when designing such a setup.

- [Replicating a running instance of OpenDNSSEC](#)
  - [Before copying data](#)
  - [What to copy](#)
  - [After copying data](#)
- [Master/Slave](#)
- [Signatures lifetimes](#)
- [Sanity checks](#)
- [SOA considerations](#)
- [User setups](#)

### Replicating a running instance of OpenDNSSEC

The entire state of a running instance of OpenDNSSEC does not need to be replicated. It is perfectly safe to re-sign the zone as long as you use the same key set.

#### *Before copying data*

For greater consistency consider manually running the enforcer immediately before copying data.

Before taking copies of the required data you should:

- Review the backup status of your keys (using the Require Backup option in conf.xml is recommended for high availability deployments)
- Backup your database with [this command](#)
- Stop the enforcer demon
- If you are only using file based adapters (i.e. not AXFR or IXFR) then stop signer demon

#### **What to copy**

- The KASP database
  - for MySQL also see the documentation at: [myslq.com](http://myslq.com)
  - for SQLite the database file is found by default in /var/opendnssec
- The configuration files (default location is the /etc/opendnssec/ directory)
- The state data
  - the minimum data required are the signconf files (default location is the /var/opendnssec/signconf directory)
  - NOTE: when using only file adapters it may be convenient to copy the entire contents of /var/opendnssec directory (assuming default location)
- And, of course, the unsigned zones and keys must be available to the replicated instance

And that should be all you need to do.

#### **After copying data**

- Restart any demons that were stopped
- If restoring into an existing install of OpenDNSSEC ensure that the /var/opendnssec/tmp and /var/opendnssec/signed directories are empty

## **Master/Slave**


Careful consideration should be given to which, if any, process are run on a slave (or on each master in a Master-Master) configuration. Some operators don't run either the enforcer or the signer on a slave instance but merely duplicate the data between the two instances in a timely fashion. Others run two master servers, both enforcing and signing but only publishing from an 'active' master.

## **Signatures lifetimes**

When choosing signature lifetimes some consideration should be given for how long it may take to detect a failure and then fail over to a backup instance of OpenDNSSEC. Signatures should be valid for **at least** as long as this process is likely to take.

## **Sanity checks**

Many operators configure a set of sanity checks to ensure the output from 2 instances of OpenDNSSEC produce consistent results. These are often done via custom scripts.

 Note that the signed zones from two different instances will never be identical for several reasons e.g. the inception/expiration times of the signatures will be different.

## **SOA considerations**

Consideration also needs to be given to managing the zone SOA over a failure.

## User setups

Examples of user deployments can be found in the [User Reference Material](#)

## Command Utilities

This section details the various command utilities that are available with OpenDNSSEC.

### On this Page

- [ods-control](#)
- [ods-ksmutil](#)
- [ods-signer](#)
- [ods-hsmutil](#)
- [ods-hsmspeed](#)
- [ods-kaspcheck](#)
- [hsmbully](#)
- [Daemons](#)
  - [ods-signerd](#)
  - [ods-enforcerd](#)

### ods-control

Is a wrapper around the commands below.

```
usage: ods-control
ksm | hsm | signer | start | stop
```

- The first three options pipe commands to *ods-ksmutil*, *ods-hsmutil*, and *ods-signer*.
- The last two options start and stop the two daemons of OpenDNSSEC, *ods-enforcerd* and *ods-signerd*.

### ods-ksmutil

You need a way to interact to the KASP Enforcer, for example to add and remove zones that are handled by OpenDNSSEC. The *ods-ksmutil* utility provides a number of commands to make this easier, all commands are invoked on the unix command line.

- **You must run the *setup* option before you ever run any sub-system in OpenDNSSEC.** This reads the configuration *kasp.xml* and imports these settings into the KASP Enforcer database.

⊖ The *setup* command deletes the current content of the database! (Including information on keys; such that existing keys will become unusable and new keys will need to be generated.)

- If you make any changes to *kasp.xml* these changes must be imported into the database. Use the *update* command to do this without losing any other data.
- To add a zone to be handled by OpenDNSSEC, use the *zone add* command. This command needs a



parameter to specify the zone, and optional parameters for which policy to use and which paths to use for input and output. An example of use:

```
ods-ksmutil zone add -z example.com -p default -i /var/example.com -o
/var/example.com.signed
```

- The command *zone delete* is simpler and needs no further parameters but the name of the zone.

A complete list of commands can be found by running:

```
ods-ksmutil -h
```

or they are shown in detail here: [ods-ksmutil commands](#)

## **ods-signer**


The *ods-signer* provides a Command Line Interface to the *ods-signerd*. There are a number of commands you give to *ods-signer*. If you start the CLI without any command line parameters you enter a shell where you can issue commands:

```
ods-signer
cmd> help
Commands:
zones          Show the currently known
zones
sign <zone>    Read zone and schedule
zone for immediate (re-)signing
sign --all     Read all zones and
schedule all for immediate (re-)signing.
clear <zone>   Delete the internal
storage of this zone.
               All signatures will be
regenerated on the next re-sign.
queue         Show the current task
queue.
flush         Execute all scheduled
tasks immediately.
update <zone>  Update this zone signer
configurations.
update [--all] Update zone list and all
signer configurations.
start         Start the engine.
running      Check if the engine is
running.
reload       Reload the engine.
stop         Stop the engine.
verbosity <nr> Set verbosity.
```

The same commands can be passed as command line arguments in your unix shell.

## ods-hsmutil

The *ods-hsmutil* utility is designed to interact directly with your HSM and can be used to manually list, create or delete keys. It can also be used to perform a set of basics HSM tests.


 Be careful before create or deleting keys using *ods-hsmutil*, as the changes are **not** synced with the KASP Enforcer.

## ods-hsmspeed

The tool *ods-hsmspeed* does performance testing on your HSM. This is also useful to find out at what speed you can get from SoftHSM on your CPU.

## ods-kaspcheck

This tool is provided to check that the configuration files (conf.xml and kasp.xml) are semantically sane and contain no inconsistencies.

 It is advisable to use this tool to check your configuration before starting to use OpenDNSSEC.

```
ods-kaspcheck -h
```

```
Usage: ods-kaspcheck options
```

```
Specific options:
```

```
-c, --conf PATH_TO_CONF_FILE Path to  
OpenDNSSEC configuration file  
      (defaults to the default conf.xml  
file)
```

```
-k, --kasp PATH_TO_KASP_FILE Path to KASP  
policy file  
      (defaults to the path given in the  
configuration file)
```

```
Common options:
```

```
-h, -?, --help                Show  
this message
```

## hsmbully

The `hsmbully` tool may be used to test your HSM for compliance with PKCS#11. This tool is not part of OpenDNSSEC, but can be found in the SVN repository:

```
svn co
http://trac.opendnssec.org/browser/trunk/h
smbully hsmbully
```

## Daemons

You can also run the two OpenDNSSEC daemons `ods-signerd` and `ods-enforcerd` from the command line, they are installed into the `sbin` directory.

### `ods-signerd`

This is the component that performs all of the signing. It first reads `zonelist.xml` and then goes through all zones to sign them if needed. Start the daemon by running:

```
ods-signer start
```

or if you want to use specific command line options:

**ods-signerd -h****Usage: ods-signerd [OPTIONS]****Start the OpenDNSSEC signer engine daemon.****Supported options:**

**-c | --config <cfgfile>** Read configuration from file.

**-d | --no-daemon** Do not daemonize the signer engine.

**-1 | --single-run** Run once, then exit.

**-h | --help** Show this help and exit.

**-i | --info** Print configuration and exit.

**-v | --verbose** Increase verbosity.

**-V | --version** Show version and exit.

**BSD licensed, see LICENSE in source package for details.**

**Version 1.4.0. Report bugs to <<http://bugs.opendnssec.org/>>.**

**ods-enforcerd**

The Enforcer daemon creates keys if needed (and configured to); it also maintains the states of the keys according to the appropriate policy. As the states of keys change, it communicates these changes to the signer via the configuration files that the signer uses when signing the zones. To run, call:

# ods-enforcerd

## ods-ksmutil

This is a utility that allows several different actions to be performed (relatively) easily.

### On this Page

- [Global Options](#)
- [Command: --version](#)
- [Command: setup](#)
- [Command: start|stop|notify](#)
- [Command: update](#)
- [Command: zone add](#)
- [Command: zone delete](#)
- [Command: zone list](#)
- [Command: repository list](#)
- [Command: policy export](#)
- [Command: policy import](#)
- [Command: policy list](#)
- [Command: key list](#)
- [Command: key export](#)
- [Command: key import](#)
- [Command: key rollover](#)
- [Command: key purge](#)
- [Command: key generate](#)
- [Command: key ds-seen](#)
- [Command: key ksk-retire](#)
- [Command: key delete](#)
- [Command: backup done](#)
- [Command: backup prepare](#)
- [Command: backup commit](#)
- [Command: backup rollback](#)
- [Command: backup list](#)
- [Command: database backup](#)
- [Command: rollover list](#)
- [Command: zonelist export](#)
- [Command: zonelist import](#)
- [Allowed values](#)

### Global Options

```
--config <config>          aka -c
```

Change the conf.xml file that is used, from the default.

**Command: --version**

```
ods-ksmutil --version          aka -V
```

Report version information

**Command: setup**

```
ods-ksmutil setup
```

Import conf.xml, kasp.xml and zonelist.xml into a database (deletes current contents, including any keys).

**Command: start|stop|notify**

```
ods-ksmutil start|stop|notify
```

Start, stop or SIGHUP the ods-enforcerd

**Command: update**

```
ods-ksmutil update kasp
ods-ksmutil update zonelist
ods-ksmutil update conf
ods-ksmutil update all
```

Update database from config\_dir (like above, but existing contents are kept)

**Command: zone add**

```
ods-ksmutil zone add
```

Add a zone to both zonelist.xml and the database (both locations read from conf.xml).

## Options

```
--zone <zone>                aka -z
[--policy <policy>]          aka -p
[--signerconf <signerconf.xml>] aka -s
[--input <input>]           aka -i
[--output <output>]         aka -o
[--no-xml]                   aka -m
```

Defaults are provided for all options but zone name.

The "no-xml" flag is useful when adding a number of zones; it prevents zonelist.xml from being written to thus speeding up the process. If the "no-xml" flag is used then after all the zones have been added then the zonelist file will need to be updated via the command:

```
ods-ksmutil zonelist export
```

## Command: zone delete

```
ods-ksmutil zone delete
```

Delete a zone to both zonelist.xml and the database (both locations read from conf.xml).

## Options

```
--zone <zone> | --all        aka -z /
-a
```

## Command: zone list

```
ods-ksmutil zone list
```



List zones from the zonelist.xml

**Command: repository list**

```
ods-ksmutil repository list
```

List repositories from the database

**Command: policy export**

```
ods-ksmutil policy export
```

Export a policy from the database in kasp.xml format.

**Options**

```
--policy <policy> | --all          aka -p /  
-a
```

**Command: policy import**

```
ods-ksmutil policy import
```

Update the database with the contents of kasp.xml; identical to "update kasp".

**Command: policy list**

```
ods-ksmutil policy list
```

List policies available.

**Command: key list**

## ods-ksmutil key list

List information about keys in zone.

### Options

```
[--verbose]
--zone <zone> | --all          aka -z /
-a
(will appear soon:
[--keystate <state>]          aka -e
[--keytype <type>]           aka -t
[--ds]                         aka -d
)
```

### Command: key export

## ods-ksmutil key export

Export key information in a suitable format for putting into a zonefile

### Options

```
--zone <zone> | --all          aka -z
[--keystate <state>]          aka -e
[--keytype <type>]           aka -t
[--ds]                         aka -d
```

### Command: key import

## `ods-ksmutil key import`

Add a key which was created outside of the OpenDNSSEC code into the database

### Options

```
--cka_id <CKA_ID>                aka -k
--repository <repository>         aka -r
--zone <zone>                      aka -z
--bits <size>                      aka -b
--algorithm <algorithm>           aka -g
--keystate <state>                aka -e
--keytype <type>                  aka -t
--time <time>                    aka -w
[--retire <retire>]              aka -y
```

### Command: key rollover

## `ods-ksmutil key rollover`

Rollover active keys on a zone or policy

### Options

```
--zone <zone> / --policy <policy>
[--keytype <type>]
```

"keytype" specifies the type of key to roll (both are rolled if nothing is specified) After running, the enforcer will be woken up so that the signer can be sent the new information

If the policy that the zone is on specifies that keys are shared then all zones on that policy will be rolled. A backup

of the sqlite DB file is made (if appropriate).

**Command: key purge**

```
ods-ksmutil key purge
```

Remove keys that are in the "Dead" state from the repository and from the enforcer DB

**Options**

```
--zone <zone> / --policy <policy>  
aka -z / -p
```

**Command: key generate**

```
ods-ksmutil key generate
```

Create enough keys for the given policy to last for the period of time given by interval.

**Options**

```
--policy <policy>          aka  
-p                          aka  
--interval <interval>    aka  
-n
```

Intervals are specified in the format used in the configuration files, see [Configuration](#).

**Command: key ds-seen**

```
ods-ksmutil key ds-seen
```

Indicate that a submitted DS record has appeared in the parent zone (this triggers the completion of a KSK rollover, or the provisioning of a standby KSK).

### Options

```
[--zone <zone>                                aka
-z ]
--keytag <keytag>                              aka
-x
--cka_id <CKA_ID>                              aka
-k
[--no-retire]
```

Specifying a zone will speed up the search of keys by narrowing the field but is not mandatory; cka\_id can be used to resolve a keytag clash. By default the command will simultaneously move the current key into the retired state. If you wish to delay this step then add the --no-retire flag and use the ksk-retire command when needed.

### Command: key ksk-retire

```
ods-ksmutil key ksk-retire
```

Move a key from active to retired (if a replacement key is already active).

### Options

```
--zone <zone>                                aka
-z
--keytag <keytag>                              aka
-x
--cka_id <CKA_ID>                              aka
-k
```

Specifying a zone alone will retire the oldest key in the zone; if the cka\_id or keytag are specified then that key will be retired. The specified key must be in the active state, and there must be 2 or more active keys on the zone for this command to work.

**Command: key delete**

```
ods-ksmutil key delete
```

Remove a key from the system.

**Options**

```
--cka_id <CKA_ID>      aka  
-k  
--no-hsm
```

Keys in the GENERATE or DEAD state can be removed from the system at any time as they are not actually being used. The no-hsm flag indicates that the key should be left on the HSM.

**Command: backup done**

```
ods-ksmutil backup done
```

**i DEPRECATED:** This command is deprecated in OpenDNSSEC 1.4.0 and onwards; it will be removed in OpenDNSSEC 2.0.0 and beyond.

It can be replaced with the following command sequence:

```
ods-ksmutil backup prepare
ods-ksmutil backup commit
#OR# ods-ksmutil backup rollback
```

The improvement of switching to this sequence is transactional security of the backup. Prior to this change, there were two ways of making uncertain backups:

- Running `ods-ksmutil backup done` after the actual backup: If the Enforcer was running and creating keys during the backup, the newest keys might not have been backed up, but they would still be marked as if they were by the backup done command.
- Running `ods-ksmutil backup done` before the actual backup: If the backup failed, then there would be no way to fix this. In the new situation, one can retry or execute `ods-ksmutil backup rollback`

This sequence creates time between a preparation phase and the actual commit of the backup. This time is meant for actually doing the backup.

Indicate that a backup of the given repository has been done, all non-backed up keys will now be marked as backed up.

This is especially important if the repository used has the **RequireBackup** flag set.

**NOTE:** Keys generated between a backup being made and the backup done command being run will be erroneously marked as having been backed up. To avoid this, either choose a backup schedule that doesn't run while the enforcer might be generating keys, or shutdown the enforcer while a backup is performed.

## Options

```
--repository <repository>      aka -r
--force
```

- (If no options are given then all repositories are marked as backed up.) Include this call in a HSM backup process to avoid warnings or errors about using non-backed up keys.
- If the `--force` option is not used then a manual confirmation step is required to complete the backup.

## Command: backup prepare

## `ods-ksmutil backup prepare`

Mark all keys in preparation of an HSM key backup. Later invocation of `ods-ksmutil backup commit` or `ods-ksmutil backup rollback`, will only influence these marked keys.

This command is intended as part of the following key backup procedure:

- `ods-ksmutil backup prepare`
- Make the actual HSM key backup
- `ods-ksmutil backup commit` (or, in case of problems, `ods-ksmutil backup rollback`)

This is especially important if the repository used has the **RequireBackup** flag set.

### Options

```
--repository <repository>      aka -r
```

(If no options are given then all keys in all repositories are marked for backed up.) Include this call in a HSM backup process to avoid warnings or errors about using non-backed up keys.

### Command: backup commit

## `ods-ksmutil backup commit`

Mark all keys that were previously marked with `ods-ksmutil backup prepare` as actually having been backed up. This means that the Enforcer can henceforth depend on these keys. A successful backup is a necessary precondition for the dependency on keys if the **RequireBackup** flag is set.

This command is intended as part of the following key backup procedure:

- `ods-ksmutil backup prepare`
- Make the actual HSM key backup
- `ods-ksmutil backup commit` (or, in case of problems, `ods-ksmutil backup rollback`)

### Options

```
--repository <repository>      aka -r
```

(If no options are given then all keys in all repositories that were marked before are made available to the Enforcer.) Include this call in a HSM backup process to avoid warnings or errors about using non-backed up



keys.

### Command: backup rollback

```
ods-ksmutil backup rollback
```

Undo the previous marking of keys for backup with `ods-ksmutil backup prepare`. This means that the Enforcer cannot depend on these keys if the **RequireBackup** flag is set – their availability is not sufficiently guaranteed. A future backup can try again, starting once more from `ods-ksmutil backup prepare`.

This command is intended as part of the following key backup procedure:

- `ods-ksmutil backup prepare`
- Make the actual HSM key backup
- `ods-ksmutil backup commit` (or, in case of problems, `ods-ksmutil backup rollback`)

Note that OpenDNSSEC does not place restrictions on the repair of a failing HSM backup by attempting several times; the core idea is simply that `ods-ksmutil backup commit` is *only* executed when a backup has succeeded. The command `ods-ksmutil backup rollback` exists to handle persisting problems with backups, and to enable the backup procedure to start again with marking all the keys that are available for backup at *that* moment. It is probably good practice to round off a backup procedure with either `commit` or `rollback` on the same day (or other timeslot) as `prepare` was invoked.

### Options

```
--repository <repository>      aka -r
```

(If no options are given then all keys in all repositories will have their backup-preparation marking undone.)

### Command: backup list

```
ods-ksmutil backup list
```

List the backups that have been made on the given repository.

### Options

```
--repository <repository>      aka -r
```

### Command: database backup

## `ods-ksmutil database backup`

Make a copy of the enforcer database (if using sqlite). It makes sure that the database is in a consistent state by taking a lock out first.

### Options

```
[--output <output>]      aka -o
```

If `--output` is omitted then the usual `enforcer.db.backup` is used.

### Command: rollover list

```
ods-ksmutil rollover list
```

List the expected dates and times of upcoming rollovers.

### Options

```
[--zone <zone>]         aka -z
```

### Command: zonelist export

```
ods-ksmutil zonelist export
```

Export the zone information held in the kasp database to `zonelist.xml` formatted text.

### Command: zonelist import

```
ods-ksmutil zonelist import
```

Synchronise the database with the contents of `zonelist.xml`; identical to "update zonelist".

## Allowed values

When specifying a keystate the following keywords are recognised:

```
GENERATED | PUBLISHED | READY | ACTIVE | RETIRED | R  
EVOKED | DEAD
```

When specifying a keytype the following keywords are recognised:

```
KSK | ZSK
```

When specifying a time (for **key import**) the following formats can be used:

```
YYYYMMDD[HH[MM[SS]]]  
(all numeric)
```

```
or D-MMM-YYYY[:| ]HH[:MM[:SS]]  
(alphabetic month)
```

```
or DD-MMM-YYYY[:| ]HH[:MM[:SS]]  
(alphabetic month)
```

```
or YYYY-MMM-DD[:| ]HH[:MM[:SS]]  
(alphabetic month)
```

```
D-MM-YYYY[:| ]HH[:MM[:SS]]  
(numeric month)
```

```
DD-MM-YYYY[:| ]HH[:MM[:SS]]  
(numeric month)
```

```
or YYYY-MM-DD[:| ]HH[:MM[:SS]]  
(numeric month)
```

... and the distinction between them is given by the location of the hyphens.

## Getting Help

### Running into problems?

- First try checking our [Troubleshooting Guide](#) or looking through the [Frequently Asked Questions](#).
- There is also useful information in the [User Reference Material](#) including hints and tips
- See if the issue is already reported in our [Issue Tracker](#).
  
- Need support? [Find out how](#) using our [Issue Tracker](#) or [Mailing list](#).
- Found a bug? [Report it](#).

### Troubleshooting

There are a number of common issues that are straightforward to diagnose and fix... If OpenDNSSEC is not behaving as expected then the first place to look is in the logs. Where these will be depends on your system and your configuration.

The following are log messages which you may see, and what to do about them (if anything).

#### On this Page

- [Enforcer](#)
- [Signer](#)
- [HSM login / PIN daemon](#)
  - [hsm\\_prompt\\_pin\(\): Could not access the named semaphore / shared memory: ...](#)
  - [hsm\\_prompt\\_pin\(\): Bad memory size, ...](#)
  - [hsm\\_prompt\\_pin\(\): Bad permissions on the shared memory, ...](#)
  - [Clearing the PIN daemon shared memory](#)

### Enforcer

#### **ods-enforcerd: ERROR: Trying to make non-backed up ZSK active when RequireBackup flag is set**

This is not an error as such. It means that in conf.xml you have indicated that keys should not be used unless they are backed up. However, the enforcer has determined that if it continues then a non backed up key will be made active.

*The solution* Take a backup of your keys (how this is done will depend on your key storage).

Once this has been done then run **ods-ksmutil backup done** to mark all keys as having been backed up.

#### **ods-enforcerd: WARNING: Making non-backed up KSK active, PLEASE make sure that you know the potential problems of using keys which are not recoverable**

This is the same as above, but without **RequireBackup** being set in conf.xml

#### **ods-enforcerd: WARNING: key rollover not completed as there are no keys in the ready state: ods-enforcerd will try again when it runs next**

This is seen when a rollover is happening but there is no replacement key ready (because one has not been published for long enough). It indicates that the rollover will be delayed until the replacement key is ready, the time that this will happen depends on the policy.

#### **ods-enforcerd: Could not call signer engine**

If the enforcer makes a change to a zones signer configuration (say it adds a new key) it calls the signer to get it to resign that zone. This message indicates that the signer is not running, although it has been seen on a system where everything is working fine. (See KNOWN\_ISSUES.)

#### **ods-enforcerd: Not enough keys to satisfy zsk policy for zone**

**or ods-enforcerd: Not enough keys to satisfy ksk policy for zone**

One of these messages will be seen if the enforcer does not have enough unallocated keys to provide for the zone specified. If the **ManualKeyGeneration** tag is set in conf.xml then you will need to create new keys using **ods-ksmutil key generate**, otherwise new keys will be created when the enforcer runs next. (Don't forget to backup any new keys.)

**ods-enforcerd: Rollover of KSK expected at <DATE TIME> for <ZONE>**

This is not an error, but a notification of an upcoming (scheduled) rollover. This will appear in your logs at a time prior to the rollover as configured in conf.xml (the **Enforcer/RolloverNotification** tag).

**ods-enforcerd: WARNING: KSK Retirement reached; please submit the new DS for <ZONE> and use ods-ksmutil key ksk-roll to roll the key.**

Rolling a KSK requires the DS record of the replacement key to be published in the parent of the zone. This message indicates that your KSK has reached the end of its life (as specified by your policy), and that it is time to submit the DS record to the parent.

**ods-enforcerd: Error: database in config file <path\_to\_conf.xml> does not match libksm**

This indicates that either you have libksm built for sqlite, but have specified a MySQL database in conf.xml, or *vice versa*.

*The solution* is to either rebuild libksm or to change conf.xml

**ods-enforcerd: Error reading config**

This usually means that conf.xml is either absent, not readable by the user, or badly formed. There should be a line above this one which gives a more specific error message.

**ods-enforcerd: Error getting db lock**

When using sqlite any process using the database tries to get an exclusive write lock on a file in the same directory as the kasp.db. If this directory is not writeable by the user then this message will be seen, again a more specific error message should have been issued.

**ods-enforcerd: Repository <NAME> is full, cannot create more <KSKs|ZSKs> for policy <POLICY>**

In conf.xml a capacity can be specified for a repository. When this is reached then no more keys will be generated in that repository.

*The solution* is to either run **ods-ksmutil key purge** to remove dead keys, or to raise this capacity and run **ods-ksmutil update conf** to push this change into the database. If the repository is really at capacity, and purge does not free up any space, then a new repository will be needed.

**ods-enforcerd: Repository <NAME> is nearly full, will create X <KSKs|ZSKs> for policy <POLICY> (reduced from Y)**

Y keys were needed to satisfy the policy, but the repository only has room for X more. This warning might precede the error detailed above, and the solution is the same.

**ods-enforcerd: NOTE: keys generated in repository <NAME> will not become active until they have been backed up**

This is not an error, but a reminder that a backup needs to be done (as new keys have just been generated).

**ods-enforcerd: Signconf not written for <ZONE>**

Some error has happened and the enforcer will not overwrite the existing signconf file, so the old one will be left in place. There should be a more specific message indicating the root cause just prior to this line. (*E.g.* attempting to use a non backed up key.)

**ods-enforcerd: There are no <KSKs|ZSKs> in the generate state; please use "ods-ksmutil key generate"**

**to make some**

**ManualKeyGeneration** has been set (in conf.xml) and the system has run out of keys.

*The solution* is to run the **ods-ksmutil key generate** command, back up the keys, and the system will recover when it runs next.

**Signer**

**These messages might show up in the logs if there is a parse or semantic error in one of the configuration files.**

```
ods-signerd: error: unable to read cfgfile <file>
ods-signerd: error: unable to parse cfgfile <file>
ods-signerd: error: unable to read conf rng file <file>
ods-signerd: error: unable to create XML RelaxNGs parser context
ods-signerd: error: unable to parse a schema definition resource
ods-signerd: error: unable to create RelaxNGs validation context
ods-signerd: error: configuration file validation failed
ods-signerd: error: unable to create new XPath context for cfgfile <file>
ods-signerd: error: unable to evaluate required element <element> in cfgfile <file>
ods-signerd: error: cfgfile <file> has errors
ods-signerd: error: unable to evaluate xpath expression <expr>
ods-signerd: error: unable to open zone list file <file>
ods-signerd: error: unable to extract zone name from zonelist
ods-signerd: error: unable to read zone <dname>; skipping
ods-signerd: error: unable to add zone <zone> to zone list
ods-signerd: error: error parsing zone list file <file>
ods-signerd: error: invalid salt <salt>
ods-signerd: error: unable to parse signconf file <file>
ods-signerd: error: unable to read signconf file <file>
ods-signerd: error: signconf-check: no signature resign interval found
ods-signerd: error: signconf-check: no signature resign interval found
ods-signerd: error: signconf-check: no signature default validity found
ods-signerd: error: signconf-check: no signature denial validity found
ods-signerd: error: signconf-check: no signature jitter found
ods-signerd: error: signconf-check: no signature inception offset found
ods-signerd: error: signconf-check: no nsec3 algorithm found
ods-signerd: error: signconf-check: wrong nsec type <rrtype>
ods-signerd: error: signconf-check: no keys found
ods-signerd: error: signconf-check: no dnskey ttl found
ods-signerd: error: signconf-check: no soa ttl found
ods-signerd: error: signconf-check: no soa minimum found
ods-signerd: error: signconf-check: wrong soa serial type <string>
```

**These messages might show up in the logs if the signer engine daemon was unable to start up. All of them are provided with a specific message indicating the cause.**

```
ods-signerd: error: unable to create command handler, <reason>
ods-signerd: error: cannot connect to command handler: <reason>
ods-signerd: error: setup failed: chdir to <directory> failed: <reason>
ods-signerd: error: setup failed: unable to drop privileges
ods-signerd: error: setup failed: unable to fork daemon: <reason>
ods-signerd: error: setup failed: unable to setsid daemon: <reason>
ods-signerd: error: setup failed: unable to write pid file
ods-signerd: error: setup failed: unable to start command handler
```

ods-signerd: error: setup failed: unable to start command handler  
ods-signerd: error: setup failed: error initializing libhsm (errno <no>)  
ods-signerd: error: signer setup failed  
ods-signerd: error: failed to fork zone fetcher: <reason>  
ods-signerd: error: failed to setsid zone fetcher: <reason>  
ods-signerd: error: cannot stop zone fetcher: <reason>  
ods-signerd: error: cannot start zone fetcher

### **These messages might show up in the logs if the signer was unable to sign the zone**

ods-signerd: error: task [read zone <dname>] failed  
*File not found or readable, parse error, ...*  
ods-signerd: error: task [add dnskeys to zone <dname>] failed  
*HSM re-initialized, no privileges for accessing HSM, ...*  
ods-signerd: error: task [update zone <dname>] failed  
*DNS related errors in zone, for example other RRs next to a CNAME*  
ods-signerd: error: task [nsecify zone <dname>] failed  
ods-signerd: error: task [sign zone <dname>] failed  
*No privileges for accessing HSM, ...*  
ods-signerd: error: task [write zone <dname>] failed  
*Output directory not writable*

### **These messages might show up in the logs if a zone update failed**

ods-signerd: error: cannot keep SOA SERIAL from input zone (<serial>): output SOA SERIAL is <serial>  
*<SOA><Serial> is set to keep in kasp policy file, but SOA SERIAL in unsigned zone file was not increased*  
ods-signerd: error: occluded (non-glue non-DS) data at <dname> NS  
*Found unallowed RRs at the delegation*  
ods-signerd: error: occluded data at <dname> (below <dname> DNAME)  
*Found RRs below DNAME*  
ods-signerd: error: occluded (non-glue) data at <dname> (below <dname> NS)  
*Found non-glue RRs below delegation*  
ods-signerd: error: other data next to <dname> CNAME  
*Found unallowed RRs next to CNAME*  
ods-signerd: error: multiple records for singleton type at <dname> <rrtype>  
*Found multiple RRs of a singleton RRtype (CNAME or DNAME) at the same owner name*  
ods-signerd: error: update zone <dname> failed: zone data contains errors

### **These messages might show up in the logs if one of the backup files was corrupted**

ods-signerd: error: error creating DNSKEY for key <locator>  
ods-signerd: error: error adding DNSKEY[<keytag>] for key <locator>  
ods-signerd: error: error creating NSEC3 parameters for zone <dname>  
ods-signerd: error: error adding NSEC3PARAMS record to zone <dname>  
ods-signerd: error: error adding DNSKEYs to zone <dname>  
ods-signerd: error: error adding NSEC3PARAMS RR to zone <dname>  
ods-signerd: error: cannot backup zone: cannot open file <file> for writing  
ods-signerd: error: error adding key from backup file <file> to key list  
ods-signerd: error: error recovering DNSKEY[<keytag>] rr  
ods-signerd: error: error recovering nsec3 parameters from file <file>  
ods-signerd: error: error recovering NSEC3PARAMS rr  
ods-signerd: error: error reading key credentials from backup  
ods-signerd: error: error reading RRSIG from backup  
ods-signerd: error: expecting RRtype RRSIG from backup  
ods-signerd: error: error reading domain from backup file

```
ods-signer: error: error adding domain from backup file
ods-signer: error: error reading NSEC(3) RR from backup file
ods-signer: error: error adding NSEC(3) RR from backup file
ods-signer: error: unable to recover zone state from file <file>: <reason>
ods-signer: error: unable to recover denial of existence from file <file>: <reason>
ods-signer: error: unable to recover unsorted zone from file <file>: <reason>
ods-signer: error: unable to recover dnskeys from file <file>: <reason>
ods-signer: error: unable to recover rrsigs from file <file>: <reason>
ods-signer: error: domain part in backup file is corrupted
ods-signer: error: unable to recover RR to domain: failed to add RRset
ods-signer: error: ods-signer: error: unable to recover RRSIG to domain: no NSEC RRset
ods-signer: error: unable to recover RRSIG to domain: no NSEC3 RRset
ods-signer: error: unable to recover RRSIG to domain: no such RRset
ods-signer: error: nsec3params part in backup file is corrupted
ods-signer: error: key part in backup file is corrupted
ods-signer: error: unable to recover signconf backup file <file>: corrupt
```

## HSM login / PIN daemon

### **hsm\_prompt\_pin(): Could not access the named semaphore / shared memory: ...**

These errors might indicate that shared memory is not configured or configured incorrectly for the system or the OpenDNSSEC user, please see your systems manual for shared memory and ipc/ipcs commands.

### **hsm\_prompt\_pin(): Bad memory size, ...**

This error should most likely be a result of an upgrade where the data structures stored in memory have changed size. There should be information about this in the migration documentation between versions. If you did not make an upgrade something else might have changed the memory outside of OpenDNSSEC.

You can try and clear the shared memory, see section below.

### **hsm\_prompt\_pin(): Bad permissions on the shared memory, ...**

This error indicated that the shared memory exist but has the wrong permissions, either it was created by another user or something else changed the permissions.

If you are using user/group privileges see to it that you use the same user/group when you login the PIN:

```
$ sudo -u <user> -g <group> ods-hsmutil  
login
```

You can view the shared memory to see if the permissions is correct using ipcs, the user/group should be what you configured in conf.xml and permissions should be read+write for user and read+write for group.

Note that output will vary between systems.



```
$ ipcs
----- Shared Memory Segments -----
key          shmid      owner      perms
bytes        nattch     status
0x0d50d5ec  983040    opendnssec 660
25600        0

----- Semaphore Arrays -----
key          semid      owner      perms
nsems
0x0d50d5ec  425985    opendnssec 660
1
```

#### Clearing the PIN daemon shared memory

You can clear the PIN daemon memory by destroying the shared memory, first shutdown OpenDNSSEC.

```
$ ods-control stop
```

Then find the shared memory and the semaphore by using ipcs.

```

$ ipcs
----- Shared Memory Segments -----
key          shmid      owner      perms
bytes        nattch     status
0x0d50d5ec  983040    opendnssec 660
25600        0

----- Semaphore Arrays -----
key          semid      owner      perms
nsems
0x0d50d5ec  425985    opendnssec 660
1

```

Now we remove the shared memory and the semaphore by logging out.

```

$ ods-hsmutil logout

```

And then run ipcs again so we see that the shared memory and the semaphore are gone.

```

$ ipcs
----- Shared Memory Segments -----
key          shmid      owner      perms
bytes        nattch     status

----- Semaphore Arrays -----
key          semid      owner      perms
nsems

```

## Frequently Asked Questions

**On this Page**

- [The Signer Engine creates files in the tmp-directory, but nothing is written to the signed-directory. What went wrong?](#)
- [I get this message when doing a manual rollover: "WARNING: key rollover not completed as there are no keys in the 'ready' state; ods-enforcerd will try again when it runs next"](#)
- [I am using a Sun Crypto Accelerator \(SCA\) 6000 under Linux and all HSM operations fail with CKR\\_HOST\\_MEMORY](#)
- [How does OpenDNSSEC provide the DS record to the parent zone?](#)
- [How can I load the signed zone into my name server?](#)
- [SoftHSM doesn't work with OpenSC on MacOSX 10.6](#)
- [How can I validation the zones produced by OpenDNSSEC?](#)

**The Signer Engine creates files in the tmp-directory, but nothing is written to the signed-directory. What went wrong?**

Is auditing enabled for this zone and is there a finalized file in the tmp-directory?

If yes, then the Auditor does not allow to distribute this zone. Could be that unsupported RR were used, Signer Engine and the Auditor disagree on how to parse the information, or that OpenDNSSEC is not following the policy. Check the log to see what it complained about, or run:

```
ods-auditor -s <path to the
"zone".finalized file> -z "zone"
```

If no, then something went wrong in the signing process. Please check the logs and report to the OpenDNSSEC team.

**I get this message when doing a manual rollover: "WARNING: key rollover not completed as there are no keys in the 'ready' state; ods-enforcerd will try again when it runs next"**

OpenDNSSEC makes sure that the zone is secure during the rollover process. This message comes when there is no key that has been published long enough. You probably have no standby keys in your policy. When you initiate the rollover, then OpenDNSSEC first needs to publish the key and after a moment make it active. So do not worry, the rollover process will be finished in a moment.

**I am using a Sun Crypto Accelerator (SCA) 6000 under Linux and all HSM operations fail with CKR\_HOST\_MEMORY**

You need to make sure the user running OpenDNSSEC is a member of the opencryptoki group (usually "pkcs11?"), or it cannot access the shared memory region used by openCryptoki.

**How does OpenDNSSEC provide the DS record to the parent zone?**

Currently, manual intervention is required for a KSK rollover. This intervention is a three-stage process that is described in the [Uploading a Trust Anchor](#) section of the Running OpenDNSSEC page.

For future versions, we are automating this process as much as possible, including integration points for interfacing with a parent registry.

**How can I load the signed zone into my name server?**

The configuration file `conf.xml` holds a specific Signer configuration section. In there, you can configure `NotifyCommand` to be called by the signer after the zone has been successfully signed. You can put `%zone` and `%zonefile` in here, which will expand to the name of the zone that was signed and the filename of the signed zone.

For example:

```
<NotifyCommand>nscd reload<\NotifyCommand>
```

or

```
<NotifyCommand>rndc reload  
%zone<\NotifyCommand>
```

#### SoftHSM doesn't work with OpenSC on MacOSX 10.6

If you're building SoftHSM in 64-bit mode (which is the default on 10.6), you need a 64-bit version of OpenSC as well – e.g. the latest [OpenSC SCA](#) snapshot.

Note: `pkcs11-tool` from OpenSC is typically used for low-level PKCS#11 debugging and is not required by OpenDNSSEC.

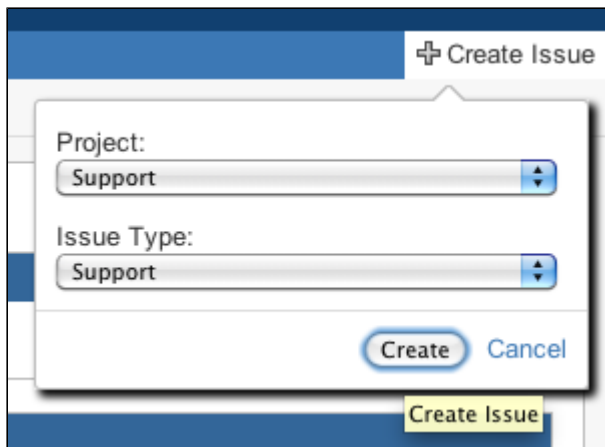
#### How can I validate the zones produced by OpenDNSSEC?

In [version 1.3](#) and earlier the auditor function in OpenDNSSEC can be used. In 1.4 the auditor has been removed and suggestions for how to use an external tool to validate the zones are given on the [Zone Auditing](#) page.

## Getting Support

**You can receive support by creating a support issue in our [Issue Tracker](#) or sending an email to our [OpenDNSSEC user mailing list](#).**

You will have to create an account in order to create issues.

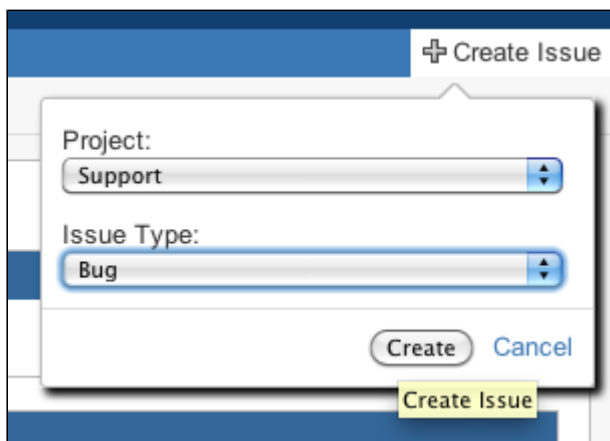


## Reporting bugs

Reporting bugs is done via our [Issue Tracker \(JIRA\)](#) by creating a new issue (upper right corner) in the Support project and selecting the issue type Bug.

✓ Or bookmark this link to create new bugs in Jira: <http://bugs.opendnssec.org/>

You will have to create an account in order to create issues.



Be sure to supply as much information as possible to help us find the problem faster, we recommend you supply the following information:

- The operating system and distribution that you are using and what version it has.
- OpenDNSSEC version and if its a package or self compiled.
- SoftHSM version if its used and if its a package or self compiled.
- Output from `/var/log/syslog` at the time of the problem, please attach it as a file if its large.
- Configurations used at the time if your not prohibited by company policy or something else, please attach it as a file.
- Zone file used at the time, both unsigned and signed if you wish, if your not prohibited. Please attach it as a file.
- Stack traces from core dumps or valgrind output (see [How to get information from a segmentation fault](#)).
- strace/dtrace output logs if you made any, please attach as a file.

- **And the most important thing; Actions/steps you did to trigger and/or replicate the problem!**

This is so we can replicate the problem on our end and find a solution faster.

If you are unsure about creating an issue, feel free to post a message to our OpenDNSSEC user mailing list. You can subscribe to it at <http://lists.opendnssec.org/mailman/listinfo/opendnssec-user>.

### How to get information from a segmentation fault

**These steps might now work if you are running a distribution package of OpenDNSSEC since the debug symbols are most likely removed.**

If its possible please compile OpenDNSSEC yourself and reproduce the problem.

### Stack trace from a core dump

First you need to verify that your system lets you get a core dump file. You should be able to see in the syslog or dmesg if a core dump has already been created otherwise you can run the following command and rerun the program that segfaulted from the same shell and it should create a core dump.

```
$ ulimit -c unlimited
```

Now that you have a core dump file you run it against gdb to be able to produce a stack trace:

```

$ gdb <full path to program that
segfaulted> <core dump file>
(gdb) bt full
#0  0x0000003e236330c5 in raise (sig=6) at
    ../nptl/sysdeps/unix/sysv/linux/raise.c:64
    resultvar = 0
    pid = <value optimized out>
    selftid = <value optimized out>
#1  0x0000003e23634a76 in abort () at
    abort.c:92
    save_stage = 2
    act = {__sigaction_handler =
{sa_handler = 0, sa_sigaction = 0}, sa_mask
= {__val = {266882107328, 5, 0,
            139663065951328, 352,
            4294967295, 0, 0, 266885219392, 0, 0, 2051,
            1050323, 1, 33188, 0}}}, sa_flags = 0,
    sa_restorer = 0xd95}
    sigs = {__val = {32, 0 <repeats 15
times>}}

```

## Running valgrind

[Valgrind](#) is a tool to detect memory management bugs and can catch some issues that gdb can't. You should be able to find it as a package for most distributions, otherwise you will need to install it.

Please note that running a program with valgrind makes it very slow so you should only do this if you can replicate the problem/crash.

Easiest way to run the signer or enforcer in valgrind is to edit the ods-control and prefix the start of the programs with valgrind. You can also specify an log file for valgrind for easy attachment to an issue later on.

ods-control:

```
echo "Starting signer..."
      valgrind
--log-file=/tmp/valgrind.log
"$sbindir/ods-signer" start
```

```
echo "Starting enforcer..."
      valgrind
--log-file=/tmp/valgrind.log
"$sbindir/ods-enforcerd"
```

## Reference Material

### Downloads

Versions of the online documentation for download are [available here](#).

### References

Useful [reference material](#) can be found below:

- Training:
  - [Training Videos and Study Material](#)

✓ The training material is highly recommended for new users!

- Technical references
  - [OpenSSL](#)
  - [PKCS#11](#)
- HSMs
  - [HSM Buyer's Guide](#)
  - [HSM Vendors](#)
- Conference presentations
  - [RIPE 65](#)

### User provided reference material

A special area of the wiki is available for users to provide and share reference material related to OpenDNSSEC that they have prepared themselves. This includes presentations, how-to's and trouble shooting tips:

[User reference material](#)



## Downloads

### Latest Versions of Downloadable Documentation

The latest version of OpenDNSSEC is 1.3.

We supply documentation for the latest version of of the documentation in PDF and HTML format:

- [DOCS:OpenDNSSEC Documentation v1.3.pdf](#)
- [DOCS:OpenDNSSEC Documenation v1.3.html.zip](#)

### Downloading selected pages

Export selected pages using the tool [here](#) (it is recommended to exclude the \_Inclusions\_Library).  
Export individual pages using the 'Tools->Export as PDF' option.