How do I...?

Although under the hood a lot has changed in OpenDNSSEC 2.0, the architecture and workflow has more in common with OpenDNSSEC 1.4 than it differs. This 'HOWTO' will initially focus on procedures that have changed or are formerly not possible. In the future this should be more complete and include updated sections now only found in the 1.4 Documentation.

Upgrade OpenDNSSEC 1.4.10 to OpenDNSSEC 2.0

With the rewrite of the `ods-enforcerd` daemon the database layout has changed as well. Upgrade scripts for Sqlite3 and MySQL are provided in the source tarball in `enforcer/utils/1.4-2.0_db_convert`. See "Migration from 1.4 to 2.1" for more information. The text file README.md in that directory explains the process. Note: while 1.4 supplied scripts to convert from one database backend to the other, 2.0 does not have these yet. If you are planning to change database backend do that first before upgrading OpenDNSSEC.

Perform an non-scheduled key rollover

OpenDNSSEC can perform a key rollover at any time. It does not matter if another rollover is already happening.

```
ods-enforcer key rollover -z example.com -t ZSK
```

The next scheduled rollover (unless `<ManualRollover/>` set in the policy) for this key type will be offset from now. I.e the new key will be used for the full lifetime configured in the policy. Keys that are no longer desired are being phased out as soon as the policy tolerates and within the bounds of what are valid DNSSEC states. This means it is very well possible an already running rollover is never completed. Consequently if the lifetime of a key is very short, in the order of the TTL of the DNSKEY, it might be possible the Enforcer is never able to complete a rollover and the old key will be used indefinitely. This is considered a bad configuration.

Change a policy configuration

A zone is tied to a policy, and policies are described in `kasp.xml` in `/etc/opendnssec`. The 2.0 enforcer will store policy timing information per key in its database. As a result all TTL values can safely be changed without running the risk of a zone become bogus during a current or future rollover. In contrast, if one would shorten a TTL in 1.4 the next rollover has a change to be performed to quickly as the records would still be in cache with the old, long TTL. After editing the policy in `kasp.xml` one should issue a policy import and urge the Enforcer to see if there is any work to do on the zones.

```
ods-enforcer policy import
ods-enforcer enforce
```

There is no need to restart the enforcer. In fact, the KASP is not being read on startup. Any new keys will use the new timings.

Change the signing algorithm

Changing the algorithm is no different than changing other policy parameters. Change the <Algorithm> field in the KASP (make sure to do this for both KSK and ZSK) and run policy import.

```
ods-enforcer policy import
ods-enforcer enforce
```

Soon the enforcer wil notice there are no keys with the newly chosen algorithm and will introduce new ones. Algorithm rollovers are a bit different than normal rollovers. During the process your zone will be signed with two keys.

Clear all state and start over

Suppose you are testing OpenDNSSEC in your environent and at some point want to reset is to be like a fresh installed instance. The main thing to do is run `ods-enforcer-db-setup` (after stopping OpenDNSSEC entirely). The Enforcer database is then clean and empty. We still need to files from `/var` to make sure the Signer isn't still using old data.

```
rm /var/opendnssec/enforcer/zones.xml
rm /var/opendnssec/signconf/*
rm /var/opendnssec/signer/*
ods-enforcer-db-setup
```

Stop using DNSSEC for a zone

If it is no longer desired to sign a zone, OpenDNSSEC can help to stop signing in a safe way. The zone will become insecure without running the risk of some validators to see a bogus zone. The way to do it is to configure no keys in its policy. The Enforcer will then retract the current keys as soon as possible.

For example

kasp.xml

```
<Keys>
        <TTL>PT3600S</TTL>
        <RetireSafety>PT3600S</RetireSafety>
        <PublishSafety>PT3600S</PublishSafety>
        <Purge>P14D</Purge>
        <KSK>
                <Algorithm length="2048">8</Algorithm>
                <Lifetime>P1Y</Lifetime>
                <Repository>SoftHSM</Repository>
        </KSK>
        <ZSK>
                <Algorithm length="1024">8</Algorithm>
                <Lifetime>P90D</Lifetime>
                <Repository>SoftHSM</Repository>
        </ZSK>
</Keys>
```

Will become

kasp.xml

```
<Keys>
        <TTL>PT3600S</TTL>
        <RetireSafety>PT3600S</RetireSafety>
        <PublishSafety>PT3600S</PublishSafety>
        <Purge>P14D</Purge>
</Keys>
```

After this you must instruct the Enforcer to reread the policy and reevaluate the zones.

```
ods-enforcer policy import
ods-enforcer enforce
```

Use a single key as ZSK and KSK

New in OpenDNSSEC 2.0 is the support for CSK's (Combined Signing Key) Which take the role of KSK and ZSK. Its configuration is exactly like that of the other types but uses the <CSK> tag in the KASP.

kasp.xml

```
<Keys>
        <TTL>PT3600S</TTL>
        <RetireSafety>PT3600S</RetireSafety>
        <PublishSafety>PT3600S</PublishSafety>
        <Purge>P14D</Purge>
        <CSK>
                <Algorithm length="2048">8</Algorithm>
                <Lifetime>P1Y</Lifetime>
                <Repository>SoftHSM</Repository>
        </CSK>
</Keys>
```

It is entirely possible to roll from a split key (KSK+ZSK) policy to a single key policy (CSK) and vice versa by changing the policy.

Get a parsable version of key list

The `-p` (or `--parsable`) is added to `ods-enforcer key list` to aid processing and monitoring of the key states by an external process. `key list -v` and `key list -d` accept this flag. The information outputted and order remains the same.

```
$ ods-enforcer key list -z example.com -d 2>/dev/null
Keys:
Zone:          Key role:   DS:       DNSKEY:      RRSIGDNSKEY:  RRSIG:       Pub:  Act:  Id:
example.com    KSK         rumoured  omnipresent  omnipresent   NA           1     1
7a188b4177bed1a744c1bcb9aa4362cf
example.com    ZSK         NA        omnipresent  NA            omnipresent  1     1
1c86793b7b779d935756906ec7fd28f7

$ ods-enforcer key list -z example.com -d -p 2>/dev/null
example.com;KSK;rumoured;omnipresent;omnipresent;NA;1;1;7a188b4177bed1a744c1bcb9aa4362cf
example.com;ZSK;NA;omnipresent;NA;omnipresent;1;1;1c86793b7b779d935756906ec7fd28f7
```

While usable for automated processing, the output should not be considered stable yet, it might change in future versions.

Passing zones through OpenDNSSEC unsigned

It is possible to pass zones through OpenDNSSEC without signing them. It's useful if you want to have the same work flow for OpenDNSSEC signed and unsigned (or externally signed) zones. To achieve this add the `<Passthrough/>` element to the policy like so:

kasp.xml

```
<KASP>
    <Policy name="default">
                <Passthrough/>
        <Description>A default policy that will amaze you and your friends</Description>
        <Signatures>
                        ...
                ...
        </Policy>
        ...
</KASP>
```

Most of the following parameters such as TTLs and algorithms will not be used for this zone. They are still required to be present however, like in any other policy. Please also note that this is different than defining a policy without keys. A policy without keys will get its DNSSEC related records stripped.

Change Database Backend

If you are running OpenDNSSEC and want to change its used database backend there are three steps involved:

1. Change database settings in `conf.xml`
2. Create database
3. Convert the database

To accommodate step 3 we provided two scripts: `convert_mysql_to_sqlite` and `convert_sqlite_to_mysql` both located in `enforcer/utils`.

Usage

```
usage: ./convert_mysql_to_sqlite -i DATABASE_MYSQL -o DATABASE_SQLITE [-h HOST] [-u USER] [-p PASSWORD]
usage: ./convert_sqlite_to_mysql -i DATABASE_SQLITE -o DATABASE_MYSQL [-h HOST] [-u USER] [-p PASSWORD]
```

- DATBASE_MYSQL, Name of the MySQL database. Make sure you created the database beforehand.
- DATABASE_SQLITE, Path of SQLite database file. Will overwrite existing file
- HOST, USER, PASSWORD apply to the MySQL database. HOST will default to localhost.

When creating a SQLite database make sure the resulting file is readable and writable for the user OpenDNSSEC runs as.