Deploying DNSSEC: what, how and where

(Version 3 – June 2015)

Association Française pour le Nommage Internet en Coopération | **www.afnic.fr** | **contact@afnic.fr**
Twitter: **@Afnic** | Facebook: **afnic.fr**

Contents

Deploying DNSSEC: What, how and where

Caution

The contents of this document are provided "AS IS".

The information herein may contain technical inaccuracies, typographical errors or outdated information. This document may be updated or modified without notice at any time, in its online electronic format on the Afnic website.

Use of the information contained in this document is therefore at your own risk.
Afnic shall in no way be held liable for any loss or damage, including but not limited to consequential losses or damages, or any losses or damages arising from operations related to the use or reproduction of the examples contained in this document.

This document shall not relieve the reader of any technical training required for the understanding and use of resources or software referred to herein.

Foreword

As a key player in the French Internet, Afnic wants to play a leading role in the deployment of DNSSEC (*Domain Name System Security Extensions*) in France. The Registry for the *.fr* TLD has thus developed a multiannual strategic plan to accelerate the deployment of this technology in the *.fr* namespace.

This deployment aims to:

- make the French Internet safer;
- encourage the creation of innovative new services on this chain of trust;
- help boost the image of stakeholders in the *.fr* TLD as leaders in the security of communications infrastructures on the Internet.

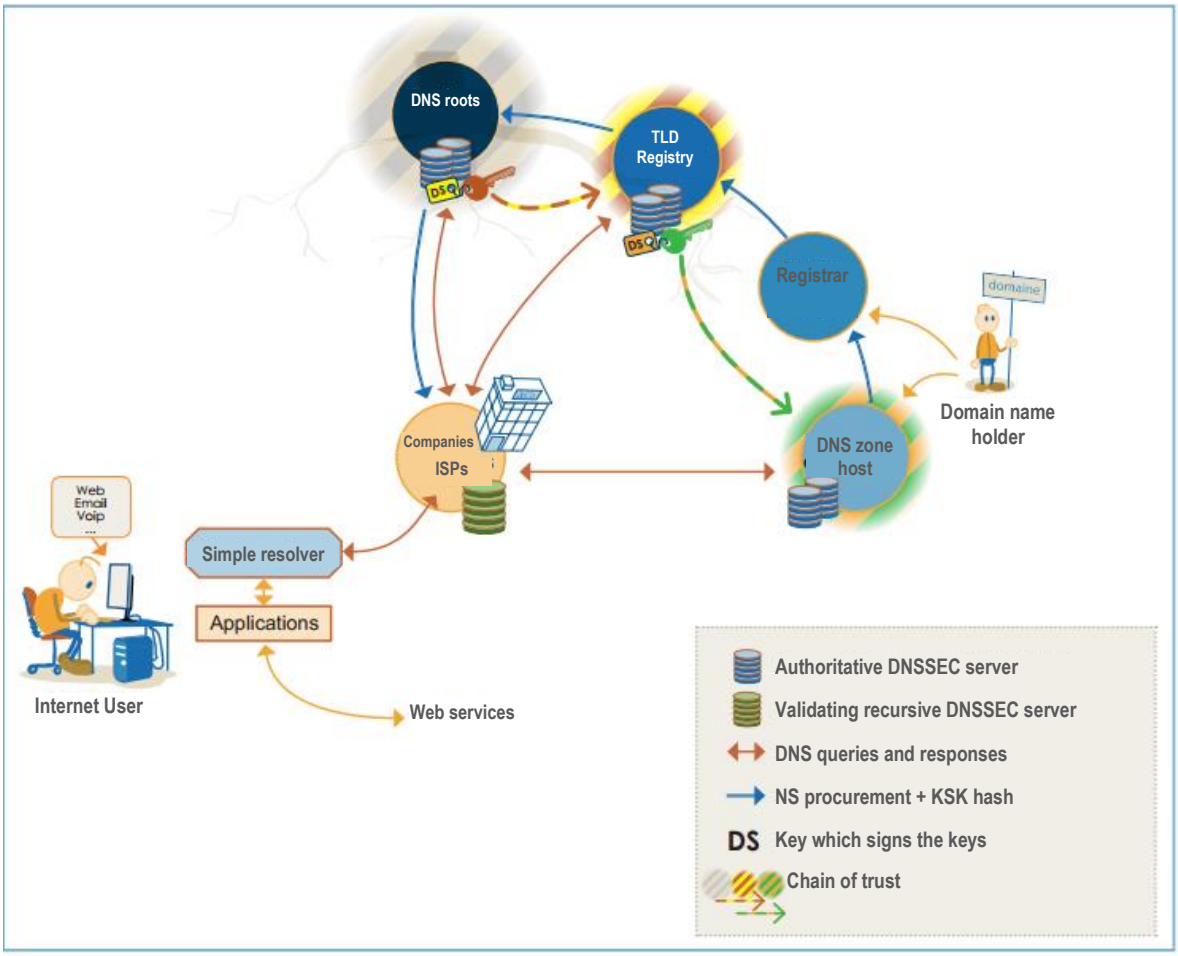
This documentation is intended to be functional, enriched with concrete examples, and aims to help the various DNS hosters with the implementation of DNSSEC.

For further information on the actions of Afnic relating to DNSSEC, or any comments on this document, please contact your custumer service representative or write to **support@afnic.fr**.

1. Introduction

The purpose of DNSSEC is to protect DNS resolution from a number of attacks. This technology today is still the only recognized and effective solution against attacks of the **DNS spoofing** (or *DNS cache poisoning*) type. However, its deployment requires taking a number of precautions because, as with any security technology, poor implementation may cause major technical/operational problems.

This document is intended for system administrators working for a DNS hoster[1], who want to deploy DNSSEC in order to sign the zones they are responsible for while minimizing any problems. With this goal in mind, this guide aims to provide tools that favor solutions to address operational errors rather than those that address massive attacks. Some choices may therefore be simplified in this documentation in order to enable quick implementation.



Components of the DNSSEC chain of trust

This guide begins with the prerequisites that should be validated before considering deploying DNSSEC in production, followed by two concrete examples of DNSSEC management using OpenDNSSEC+NSD software or using BIND.

[1] Which may or may not be a registrar.

It then describes the steps required to implement DNSSEC in production.

The implementation of DNSSEC means each zone must carry out the following operations:

- Create the keys.
- Sign its records.
- Publish the signed zone.
- Manage their validity periods.
- Manage the publications of the key summary in the parent zone with each KSK rollover.
- Check the publication of a new key before signing with it.

Please note: This document does not cover setting up DNSSEC on DNS resolvers (enabling validation), and focuses on authoritative servers.

2. Prerequisites

This section is a requirement insofar as it specifies the conditions to be met before deploying DNSSEC in production.

At this stage, a mimimum level of theoretical knowledge and practical skills is needed.

For theoretical knowledge, review various documents that explain what DNSSEC is and what it brings to the DNS in terms of security extensions. For information and without limitation, the reader may refer to the Afnic Issue paper (*afnic.dnssec-thema*: §5.Bibliography) and to the document published at the JRES (Journées Réseaux) (*bortzmeyer.dnssec-jres*: §5.Bibliography).

For practical skills, in addition to self-training, the reader may attend a dedicated DNSSEC training course.

DNSSEC changes the operating model for the DNS.

Without DNSSEC, DNS is usually called *static*. It is first configured, tested with software such as Zonemaster, and no longer requires any specific changes, with the exception of voluntary ones.
DNSSEC calls for different ways of thinking.

Due to the expiration of DNSSEC signatures[2], the zones have to be periodically re-signed. In addition, it may be useful to change the keys from time to time, in particular to be prepared if a change is suddenly needed. If no changes are ever made, this could make things very difficult when a change is eventually needed (for instance, if a key is lost or compromised). In this case, key rollovers must be handled in addition to the signatures.

[2] But not of the keys: this is a major difference from X.509.

For both signatures and keys, changes must be made in compliance with the caching periods set by DNS caches. The DNS caches can store the information in memory, thus preventing any immediate changes.

DNSSEC requires a temporal perspective.

Signatures must be regenerated at least $TTL[3] before they expire to take into account the above cache-related issues. This is difficult to achieve manually[4], which requires that the process be automated.

The following prerequisites are for people who intend to deploy in production. If you are simply experimenting with the DNSSEC technology, you may skip to the concrete examples below.

2.1. Having a good technical configuration is essential

With "traditional" DNS, even in the event of an error, everything will work thanks to the robustness of this protocol.
DNSSEC uses a different compromise: robustness is decreased slightly in favor of higher integrity guarantees. A good understanding of DNSSEC and adequate training are therefore essential.

One of the fundamental prerequisites is testing the accuracy of your DNS configuration.
This can be done with a large number of tools[5] available online. Here we will focus on **Zonemaster**[6]

Please note: Make sure to check all the issues reported by Zonemaster. If some of them seem unclear, take the time to review and understand them. Keep in mind that DNSSEC requires a good understanding of all the mechanisms involved.

In particular, the following items should be checked:

- that the authoritative servers all respond with EDNS;
- that they are able to send responses greater than 512 bytes[7];
- that they are able to send responses greater than the MTU of the link;
- that they are able to respond using TCP.

DNSSEC is a relatively old protocol (the current standard dates from 2005), but some software have only incorporated DNSSEC fully and bug-free for two or three years.
It is therefore essential to use the latest software only.
If, for system administration reasons, you are temporarily forced to use older versions only, it would be more prudent to redefine/reschedule your plans to implement DNSSEC.

[3] $TTL: the *Time To Live* of DNS records.
[4] Examples of practical problems are available in **bortzmeyer.dnssec-satin**.
[5] **http://www.bortzmeyer.org/tests-dns.html**
[6] **http://www.zonemaster.fr/**
[7] Former limit on the size of the DNS, withdrawn in 1999.

2.2.Monitoring is key

A reliable DNS system requires monitoring.
Even if it is perfectly configured, things can change later, a server may crash, a firewall may be (mis)reconfigured, etc.
All DNS hosters have monitoring in place with tools such as Nagios. This monitoring is even more necessary with DNSSEC.

For example, one problem that is detectable using monitoring tools is access filtering: if a slave server no longer updates itself due to accidental filtering of its access, with traditional DNS, the only consequence will be the distribution of potentially outdated information. But with DNSSEC, as soon as the signatures stored on the server have expired, the entire zone will be invalidated. It is therefore important to be promptly informed of any issues in order to rectify them. Additional tests should also be carried out, such as checking that:

- all servers respond with the signatures;
- the signatures are not nearing expiration;
- etc.

2.3.Synchronise your clocks

DNSSEC depends on clocks that must be set to the right time, as signatures contain a validity start and end date. If you sign using a machine whose clock is behind, your signatures may be considered expired even before you publish them. Naturally, setting machines at the correct time has been a best practice for many years, but before DNSSEC, non-compliance with these recommendations did not necessarily have any visible consequences.

Clocks should therefore be kept at the correct time, e.g. via NTP, and this should be monitored with a monitoring software[8].

2.4. Secure your storage

DNSSEC uses **cryptography** and, as such, encounters the same issues as other cryptographic systems, such as X.509 certificates. This is why it is essential that private keys be kept… private, in order to prevent them from being copied by an attacker, while maintaining backups that will recover them if the storage system fails.

If you already manage cryptographic systems, DNSSEC will be of no surprise to you.

However, if you are only starting with cryptography through DNSSEC, tread with caution. A DNSSEC private key set to "read for all" on a machine that is also a

[8] For example with the check_ntp_peer script for Nagios and compatible tools.

public web server containing a large number of programmed scripts without regard to safety should probably be avoided.

3. Concrete examples

3.1. Making choices

3.1.1. Key management

Two important choices will have to be made with respect to key management.

Firstly, you need to determine whether the zone requires one or two keys. DNSSEC documentation often gives the impression that a separation into two keys, KSK (*Key Signing Key*) and ZSK (*Zone Signing Key*), is required. This is not true[9]. Using a single key is perfectly acceptable[10]. Our advice is: if you manage your zones manually, such as in the BIND example below, use a single key, and if you are using a key management tool such as OpenDNSSEC, feel free to use two keys, the tool will take care of everything. It is worth noting that the software choice (BIND or OpenDNSSEC) is independent of the number of keys, since both tools allow implementing both key management strategies[11].

The second decision to make on key management is on where to store the keys. The least secure method is to simply store the keys on the server that generates the signatures. The safest method is to use a HSM (*Hardware Security Module*), a specialized and rugged computer that stores the keys and generates the signatures. An intermediate method is to store the keys on a USB drive locked in a safe and only removed on signing days (either after the data was changed, or because re-signing before expiration is required).

Using HSMs requires an investment in their acquisition and technical appropriation (training). The decision to use them or not is a priori guided by arbitration on the costs associated with the risks to cover/accept. In practice, for zones considered important/critical by the DNS operator DNS, they invest in a HSM. Otherwise (for “ordinary” zones), other solutions are preferred.

Using a safe is a tempting solution, but this lacks in flexibility, as it prevents the use of automatic re-signing tools such as those described in the next section. Although this solution increases the protection level of the signing key against theft / compromise, it induces an overhead in administration tasks: loading the signing keys online, manually and periodically, and installing a

[9] The largest zone with a single key is co.uk.
[10] This is sometimes referred to as CSK (*Combined Signing Key*).
[11] As a matter of fact, choosing to use both a ZSK and a KSK in the OpenDNSSEC case is based on two reasons: first, it is the choice by default (the one which is likely to provoke the least surprises), then, it is less costly/complex, since OpenDNSSEC “takes care of everything”.

reminder/monitoring system of signautres so as to to avoid forgetting to re-sign[12].

One solution, the most practical for "ordinary zones", is to store the keys on the signing server. Admittedly, this is the least secure solution among all the abovementioned ones, but for now, this solution has the advantage of simplicity and of avoinding to maintain the status quo (no signature and therefore no security at all). Of course, the key must be protected by the server's usual security mechanisms (do not store the key in a shared directory, ensure its permissions) and the server itself should be administered in compliance with best safety practices (give the root password to authorized persons only, apply security patches, etc.).

3.1.2. Software

There is of course a wide range of systems and software for hosting DNSSEC functions. Here we will focus on the most widely used hosting platform: Unix.

The examples provided here are for machines using the Debian operating system and will require slight adjustments for other systems.

Priority has been given to open source software, with a particular focus on two solutions that we already know and use:

- OpenDNSSEC and NSD,
- BIND with auto-dnssec.

3.1.3. Practical cryptographic aspects

This document is not a cryptography course and thus only addresses this aspect briefly. Bear in mind that for DNSSEC, you only need the operational aspects of cryptography; its mathematical aspects are not needed.

For those unfamiliar with cryptography, the only practical choice is whether or not to use NSEC3. NSEC3, standardized in RFC 5155, reduces the risk of enumerating zone contents via *zone walking*. If your zones are already public (zone transfer authorized) or if the content of your zones is trivial (e.g. only the *apex* and the *www* directory), then NSEC3 is useless here. But the opposite is much more common and, therefore, we recommend using NSEC3.

Finally, we recommend choosing the RSA + SHA256 algorithm (no. 8 in the **IANA algorithm registry**[13]). This is the algorithm that is used in the following examples.

[12] We recall here section 2.2 of this document which recommends setting up a monitoring system. If a safe is to be used, the monitoring system must implemet an autotaic reminder for re-signing the zone periodically, following the best current operational practices (see for example section 11.4 in *[nist.dnssec-deployment]*, §5.Bibliography).

[13] **http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml**

3.2. Example using OpenDNSSEC and NSD

3.2.1. First signing

Changing DNSSEC keys is complex and problems may arise[14].

In order to change keys regularly, automating this task is highly recommended. This is what the OpenDNSSEC software provides. It takes care of key management (from creation to removal, compliant with processing times) and signing (and re-signing). To install it on a Debian machine, use the following command line:
`aptitude install opendnssec softhsm`.

Then, start OPENDNSSEC daemons :
`ods-control start`

OpenDNSSEC can be used with HSM devices or with its own storage system called SoftHSM (which is really a simple local SQLite database that should be protected from unauthorized access attempts).

Note: Verify that the `ods` account has write access on the SoftHSM directory and its contents.

Now, initialise SoftHSM:

```
% softhsm --init-token --slot 0 --label "OpenDNSSEC"
The SO PIN must have a length between 4 and 255 characters.
Enter SO PIN:
The user PIN must have a length between 4 and 255 characters.
Enter user PIN:
The token has been initialized.
```

Note: `SO` stands for `Security Officer`.

Then, configure OpenDNSSEC to use SoftHSM by editing `/etc/opendnssec/conf.xml`:

```
<Repository name="softHSM">
                <Module>/usr/lib/softhsm/libsofthsm.so</Module>
                <TokenLabel>OpenDNSSEC</TokenLabel>
                <PIN>cestressecret</PIN>
                <RequireBackup/>
</Repository>
```

The `PIN` item is the one that was specified when initialising SoftHSM. Check that OpenDNSSEC is able to interact with the database:

```
% ods-hsmutil list
Listing keys in all repositories.
0 key found.

Repository            ID                                Type
```

[14] http://conferences.npl.co.uk/satin/papers/satin2011-Bortzmeyer.pdf

```
----------              --                              ----
```

As you can see, there are no keys, which is normal at this stage (OpenDNSSEC will create the keys itself).

Define the key and signature policy in the `/etc/opendnssec/kasp.xml` file.

Here is the default policy recommended by this document (compliant with the official documentation[15] and the OpenDNSSEC date format[16]).

Comments were added for choices involving more considerations:

```
<Policy name="default">
                <Description>Policy recommended by the Afnic DNSSEC
                HOWTO</Description>
                <Signatures>
                        <Resign>PT2H</Resign>
                        <Refresh>P3D</Refresh>
                        <!-- Signatures are valid for one month. This
                        slightly increases the risk of replay attacks
                        (compared to shorter periods of time) but permits
                        more flexibility in case of problems: if there is
                        a problem with the signing process, we will have
                        30 days to correct it. -->
                        <Validity>
                                <Default>P30D</Default>
                                <Denial>P30D</Denial>
                        </Validity>
                        <Jitter>PT12H</Jitter>
                        <InceptionOffset>PT3600S</InceptionOffset>
                </Signatures>
                 <Denial>
                        <NSEC3>
                                <Resalt>P100D</Resalt>
                                <Hash>
                                        <Algorithm>1</Algorithm>
                                        <Iterations>1</Iterations>
                                        <Salt length="8"/>
                                </Hash>
                        </NSEC3>
                </Denial>
                <Keys>
                        <TTL>PT12H</TTL>
                        <RetireSafety>PT3H</RetireSafety>
                        <PublishSafety>PT3H</PublishSafety>
                        <Purge>P14D</Purge>

                        <KSK>
                                <!-- RSA + SHA-256 -->
                                <Algorithm length="2048">8</Algorithm>
                                <Lifetime>P6Y</Lifetime>
                                <Repository>SoftHSM</Repository>
                                <!-- Do not enable automatic rollovers
                                (since communication with the parent zone
                                is required anyway). -->
                                <ManualRollover/>
                        </KSK>

                        <ZSK>
                                <Algorithm length="1024">8</Algorithm>
```

[15] https://wiki.opendnssec.org/display/DOCS/kasp.xml
[16] https://wiki.opendnssec.org/display/DOCS/Date+Time+durations

```
                                <Lifetime>P60D</Lifetime>
                                <Repository>SoftHSM</Repository>
                        </ZSK>
                </Keys>
                <Zone>
                        <PropagationDelay>PT3600S</PropagationDelay>
                        <SOA>
                                <TTL>PT7200S</TTL>
                                <Minimum>PT3600S</Minimum>
                                <Serial>datecounter</Serial>
                        </SOA>
                </Zone>
                <Parent>
                        <PropagationDelay>PT3600S</PropagationDelay>
                        <DS>
                                <!-- The actual value depends on your
                                parent zone. The value shown here is for
                                the .FR zone. -->
                                <TTL>PT3600S</TTL>
                        </DS>
                        <SOA>
                                <TTL>PT172800S</TTL>
                                <Minimum>PT5400S</Minimum>
                        </SOA>
                </Parent>
</Policy>
```

Edit **/etc/opendnssec/zonelist.xml** to specify the zones that OpenDNSSEC has to handle:

```
<Zone name="dnssec.fr">
       <Policy>default</Policy>
       ...
       <Adapters>
              <Input>
                   <File>/var/opendnssec/unsigned/dnssec.fr</File>
              </Input>
              <Output>
                   <File>/var/opendnssec/signed/dnssec.fr</File>
              </Output>
...
```

As you can see, the policy specified here is `default`, as defined above.

Generate the keys using the following command line:
ods-ksmutil key generate --policy test --interval 1Y

Then, initialise the OpenDNSSEC database using **ods-ksmutil setup**. OpenDNSSEC then creates keys for the policy ("default" here):

```
% ods-hsmutil  list
Listing keys in all repositories.
4 keys found.

Repository            ID                                Type
----------            --                                ----
softHSM               cc59d2b16e421c57eec35ed4ceae0099  RSA/1024
softHSM               e17b1d0465e6976536119c872a353911  RSA/1024
softHSM               fa8cdfc8da319311283b66fe55839212  RSA/2048
softHSM               60b57dff6604cc35ec6fdd8aef7710a2  RSA/2048
```

Four keys are listed (the exact result depends on the policy defined): two ZSK keys (1024 bits by default) and two KSK keys (2048 bits by default), both types have one active key while the other is ready to take over when needed. To view key details, use **ods-ksmutil**:

```
% sudo ods-ksmutil key list --zone dnssec.fr  --verbose
SQLite database set to: /var/lib/opendnssec/db/kasp.db
Keys:
Zone:                            Keytype:      State:    Date of next
transition:  Repository:                        Keytag:
dnssec.fr                        KSK           active    2013-07-31
15:11:45       SoftHSM                            8680
dnssec.fr                        ZSK           active    2013-09-17
15:15:18       SoftHSM                            23283
```

Note: If you do not get the keys in the list, you may need to restart the daemon **ods-control**.

The key state (**active** in this case) can be one of the following:

- **Generated**: the key has been created (OpenDNSSEC can create keys in advance, e.g. for backup purposes, using **ods-ksmutil key generate --policy test --interval 1Y**, where **1Y** is used to generate keys for the coming year);
- **Publish**: the key has been published in the DNS as a DNSKEY record but has not necessarily propagated to all the resolvers yet (as their cache may contain an old version of all the DNSKEYs);
- **Ready**: the key has been published and has propagated to all the resolvers;
- **Active**: the key is in use for signing;
- **Retired**: the key is no longer in use for signing, but is still published, as some caches may still contain old signatures generated with this key;
- **Dead**: the key is still published but should no longer be of any use, as all signatures have expired from the caches;
- **Removed**: the key has been deleted entirely;
- **Revoked**: the key has been "manually" cancelled.

As for signing, it is automatic. OpenDNSSEC will retrieve the file from the location specified (here, **/var/opendnssec/unsigned/dnssec.fr**) and place it where specified (here, **/var/opendnssec/signed/dnssec.fr**). OpenDNSSEC will also handle re-signing.

3.2.2. Maintenance

Once signing or re-signing has been completed, the name server must be reloaded. The recommended method is the OpenDNSSEC **NotifyCommand** option in **conf.xml**:

```
<NotifyCommand>/usr/local/sbin/opendnssec-nsd-reload</NotifyCommand>
```

Now, what should the `/usr/local/sbin/opendnssec-nsd-reload` script contain?

Up to and including version 3[17], NSD has no mechanism for communicating with the server. For this reason, it is necessary to create a script that calls a program that will reload the name server. That can be done by a setuid program or by using sudo. Here, we use sudo. The script can contain:

```
#!/bin/sh
logger -i -t OpenDNSSEC-signer -p daemon.info "Reloading NSD,
modification in zone $1 (file $2)"
sudo -u nsd /usr/sbin/nsdc rebuild
sudo -u nsd /usr/sbin/nsdc reload
sudo -u nsd /usr/sbin/nsdc notify
```

The sudo configuration (`/etc/sudoers`) must contain:

```
opendnssec ALL = (nsd) NOPASSWD: /usr/sbin/nsdc reload, /usr/sbin/nsdc
rebuild, /usr/sbin/nsdc notify
```

Open the log to view the "reloading" notifications:

```
Jul 26 02:54:59 aetius OpenDNSSEC-signer[15803]: Reloading NSD,
modification in zone dnssec.fr (file /var/opendnssec/signed/dnssec.fr)
```

After updating the zone, you must instruct OpenDNSSEC to re-sign the zone file:

```
# ods-signer sign dnssec.fr
Zone dnssec.fr scheduled for immediate re-sign.
```

3.2.3. *Sending DS records and changes*

Signing a zone ends with sending the DS records to the parent zone manager (typically the TLD Registry). To find out what DS records to send, use the following command:

```
# ods-ksmutil key export --ds dnssec.fr
;ready KSK DS record (SHA1):
dnssec.fr.      3600    IN      DS      8680 8 1
050a641297fbd70912eec6d6e1e056354635b370

;ready KSK DS record (SHA256):
dnssec.fr.      3600    IN      DS      8680 8 2
317348d9d1b567df63d038cdef48b0a26d1c5594aa7ba253ad622d84872baa72
```

Then, send these DS records to the Registry:

- if the DNS hoster is a registrar, the DS records may be sent via the interface provided by the Registry (EPP or Web client).

[17] This function will be itnroduced in version 4.

- if the DNS hoster is not a registrar, the DS records should be sent through the user interface provided by the registrar (Web interface, API, or other).

If the command above returns no DS records, this often means that you did not wait long enough: OpenDNSSEC has to manage processing times and will not accept to export the DS records as long as the DNSKEY record has not propagated to all the resolvers (**ods-ksmutil key list --verbose** will display this information under *Date of next transition*.)
We will now add a zone to this setup. Edit **zonelist.xml** to add the zone, in this case **dnssec.pm**. Notify OpenDNSSEC of the change:

```
# ods-ksmutil update all
...
Zone dnssec.pm found
Policy set to default.
Added zone dnssec.pm to database
```

OpenDNSSEC has now added the zone to its database:

```
# ods-ksmutil key list --zone dnssec.pm
Keys:
Zone:               Keytype:      State:    Date of next transition:
dnssec.pm           KSK           publish   2013-07-27 01:40:03
dnssec.pm           ZSK           active    2013-08-25 11:40:03
```

To roll the KSK manually, e.g. because you suspect it may have been compromised, instruct OpenDNSSEC to perform a rollover:

```
% ods-ksmutil key rollover --zone dnssec.fr --keytype KSK
```

The new key immediately appears in the **publish** state. It is published in the DNS but cannot be used right away, as it may not yet have propagated to all the caches:

```
% ods-ksmutil key list --zone dnssec.fr  --verbose
...
dnssec.fr                       KSK           publish   2013-08-01
05:11:46       c12c000741d3d79c2ce22cfaa3567e9c  SoftHSM
                           34292
```

After some time, depending on the chosen TTL, the key switches to the **ready** state:

```
% ods-ksmutil key list --zone dnssec.fr  --verbose
...
dnssec.fr                   KSK           ready     waiting for ds-seen
...
```

The DS records can now be extracted and sent to the parent zone as described above. Once a DS record has been added to the parent zone (this should be checked using **dig**), notify OpenDNSSEC that the DS record has been seen:

```
% ods-ksmutil key ds-seen --zone dnssec.fr --keytag 34292
```

The key switches to the `active` state. You do not have to do anything about the old key, it will be automatically removed as soon as is safe to do so.

3.3. Example using BIND

Since version 9.9, BIND is able to handle signing and re-signing. All that is left to do is key management, but as we saw earlier, systematic and automatic key rollovers are in no way mandatory.

Thus, the principle with BIND is as follows:

1. create the key(s);
2. instruct BIND to manage signing and re-signing for the zone on its own.

Install the only software required in this example, i.e. BIND, for instance using (on Debian):
`aptitude install bind9.`
Then, start by calling `dnssec-keygen`. Use the following for a single key:

```
% dnssec-keygen -a RSASHA256 -3 -f KSK -b 2048 dnssec.fr
Generating key pair.........+++
.............................................+++
Kdnssec.fr.+008+49734
```

Note: You can add the `-n ZONE` option, but this is the default value anyway.

If you notice that this is taking a long time to complete and that, after a command like **top** reports that the machine is doing very little, this is normal: generating a key requires entropy to produce completely random numbers, and your machine might not have enough entropy. Paradoxically, your machine needs to work in order to make the key generation process go faster (e.g. compile a very large program[18]), as hard drive input/output is the main source of entropy.
When completed, `dnssec-keygen` creates two files:

```
% ls -lt Kdnssec.fr*
-rw-r--r-- 1 bortzmeyer bortzmeyer  598 Jul 19 10:53
Kdnssec.fr.+008+49734.key
-rw------- 1 bortzmeyer bortzmeyer 1776 Jul 19 10:53
Kdnssec.fr.+008+49734.private
```

The private key is very important: it must be securely stored on a reliable machine, and backups should be made in case of machine failure. Key management is a new and important aspect of DNSSEC.
The public key must be stored in a directory where BIND can find it (`key-directory`, see below)[19].

Then, configure BIND:

```
options {
       directory "/etc/bind";
       key-directory "/var/bind/keys";
       recursion no;
       dnssec-enable yes;
```

[18] In Unix, using `find / -type f` is also a good solution.
[19] If you read old online documentation, you may find indications stating that the public key should be stored in the zone file. This is no longer necessary with the method described here.

```
};

zone "dnssec.fr" in {
       inline-signing yes;
       auto-dnssec maintain;
       update-policy local; # Necessary, says the ARM (otherwise, you cannot
freeze/thaw)
       type master;
       file "dnssec.fr";
       ...
}
```

And that's it. BIND will find the keys in **key-directory** and will perform the signing on its own[20]. BIND will also re-sign whenever necessary[21].

Please note that the zone was signed with NSEC. In order to sign with NSEC3, you must specify it in BIND and provide a salt, i.e. a random number that will be used to make dictionary attacks more difficult:

```
% dd if=/dev/random bs=1 count=10 | md5sum | cut -c1-8
10+0 records in
10+0 records out
68f499ee
10 bytes (10 B) copied, 0.00046732 s, 21.4 kB/s

[Use the salt value below]
% rndc signing -nsec3param 1 0 10  68f499ee auto.rd.nic.fr
```

As you can see, the zone has now been signed with NSEC3.

If you want to use two keys, i.e. a KSK and a ZSK, create them first (please note that one key is generated with **-f KSK**, and that the ZSK has a smaller size):

```
% dnssec-keygen -a RSASHA256 -3 -f KSK -b 2048 dnssec.fr
Generating key pair....+++ ...................+++
Kdnssec.fr.+008+09634

% dnssec-keygen -a RSASHA256 -3 -b 1024 dnssec.fr
Generating key pair..............++++++ ..............++++++
Kdnssec.fr.+008+29433
```

Copy both keys into the directory. BIND will sign the DNSKEY record with both keys, and everything else with the ZSK key.

Once you have thoroughly tested your zone, and that the new information has propagated to all caches[22], you can complete the DNSSEC chain of trust by submitting a DS record to the parent zone for publication. Calculate the DS record of the zone using **dnssec-dsfromkey**:

```
% dnssec-dsfromkey Kdnssec.fr.+008+49734.key
```

[20] The following message – or similar – **Key dnssec.fr/RSASHA256/46747 missing or inactive and has no replacement: retaining signatures** will appear in the log from time to time; please ignore it, as it is incorrect.
[21] The re-signing interval can be controlled with the **sig-validity-interval** parameter.
[22] OpenDNSSEC automatically takes care of these processing times. Without it, manual calculation will be necessary.

```
dnssec.fr. IN DS 49734 8 1 3C5BA1C95F17214536AA8E82E9406321A96FBD22
dnssec.fr. IN DS 49734 8 2
A3B4267E78CF26B4442007E14B55B8F7C83A4EB81015122410B9E47E FEA2DBFD
```

If you are a registrar, send these DS records yourself, *a priori* using the EPP protocol. Otherwise, use the mechanism provided by your registrar, which will probably be a Web interface.

If any changes are made to the zone, instruct BIND to re-sign and reload:

```
% rndc  freeze dnssec.fr

[Edit the zone file]

% rndc  thaw dnssec.fr
A zone reload and thaw was started.
Check the logs to see the result.
```

As we saw earlier, systematic key rollovers are not required. However, you may be forced to perform systematic rollovers, e.g. because you found out that the key was compromised, or simply to test the procedures used in the event of such an issue.

The principle with BIND is as follows for a single key: create the new key, store it in the key directory, edit the old key to specify that it is no longer in use, and BIND will perform the rollover:

```
[Create the new key]
% dnssec-keygen   -a RSASHA256 -f KSK -b 2048 dnssec.fr
Generating key pair.....+++ ....................................+++
Kdnssec.fr.+008+00847

[Copy it into the directory, if necessary]
% cp Kdnssec.fr.+008+00847* /where/are/the/keys

[Edit the old key, i.e. 49734]
[Revoke the key immediately]
% dnssec-settime -R +0 Kdnssec.fr.+008+49734.key
[Remove the key in two days (allowing enough time for the new key to
be propagated to the caches]
% dnssec-settime -I +2d Kdnssec.fr.+008+49734.key
[Delete the key in six days (allowing enough time for signatures made with
the old key to have expired from the caches)
% dnssec-settime -D +6d Kdnssec.fr.+008+49734.key
```

Please keep in mind that processing times should be calculated taking into account the TTL of the zone records. Performing a key rollover is not easy, so be very careful when doing so (or use OpenDNSSEC, which handles the whole process automatically).

Furthermore, do not forget to change the DS record that is in the parent zone, and to synchronize the change with the changes already made in your zone. Again, this task may be delicate and should be carefully prepared.

To add a new zone to the zones served, follow the steps below:

1. generate keys for the new zone,
2. add the zone to the configuration file (`named.conf`), and add the zone file to the BIND directory,

3. reload the server (e.g. using `rndc reload`).

3.4. Debugging

As previously mentioned, DNSSEC requires a quality approach: thoroughly testing your configuration is essential. This was already recommended with traditional DNS. But in practice, it often worked even if nothing had been tested beforehand. One cannot hope for that kind of leniency with DNSSEC. Here are the main debugging and testing tools available (a longer and up-to-date list is **available online**[23].)

For DNSSEC-specific issues, the most commonly used tool is **DNSviz**[24]. It is a web tool that performs a number of tests on a signed zone and displays the results graphically, in a way that is very easy to understand.
Here is an example with a correct zone:

[23] http://www.bortzmeyer.org/tests-dns.html
[24] http://dnsviz.net/

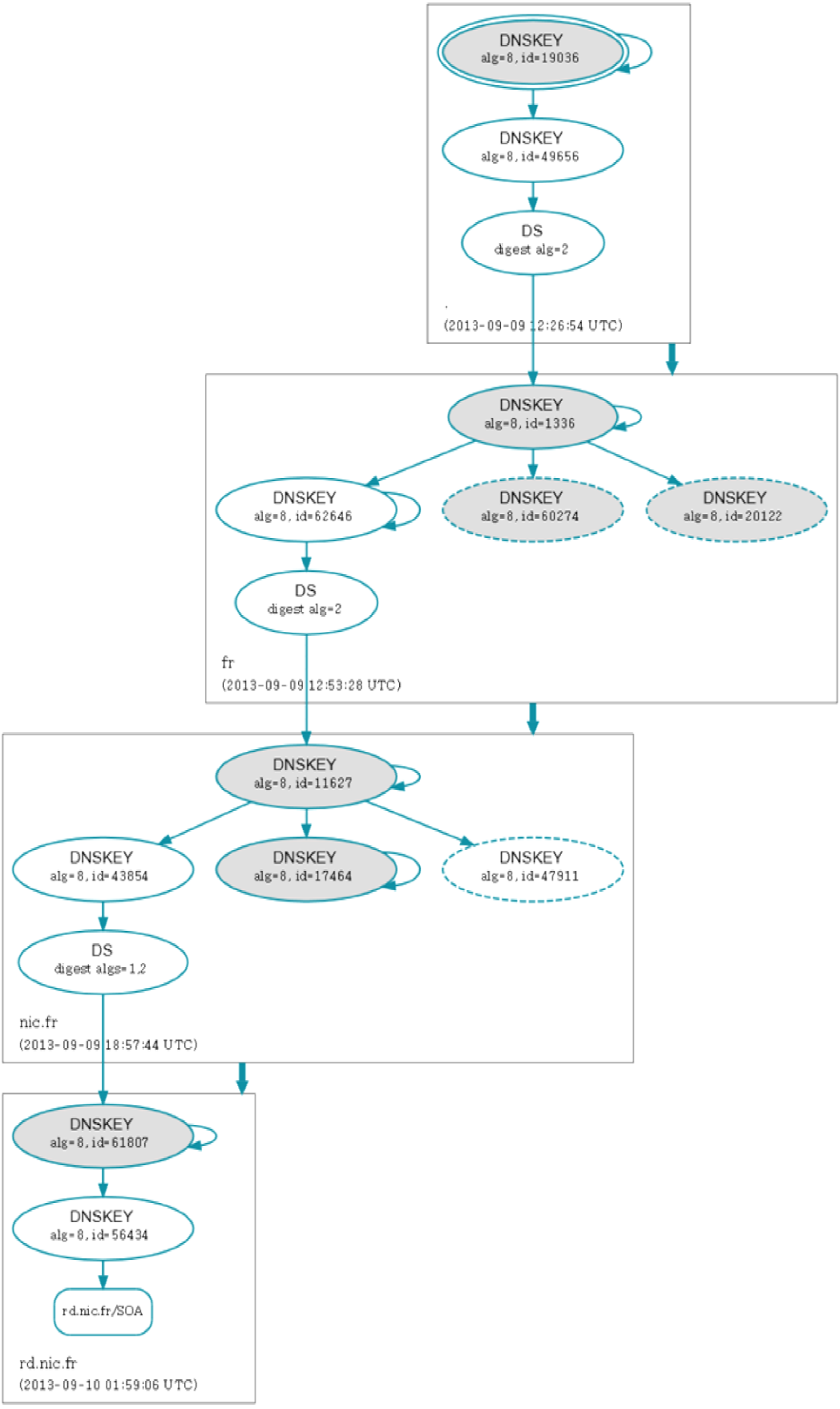


Figure 1. `rd.nic.fr` analyzed by DNSviz

And here is an example with an incorrect zone (the signatures have expired, details appear in the left column):

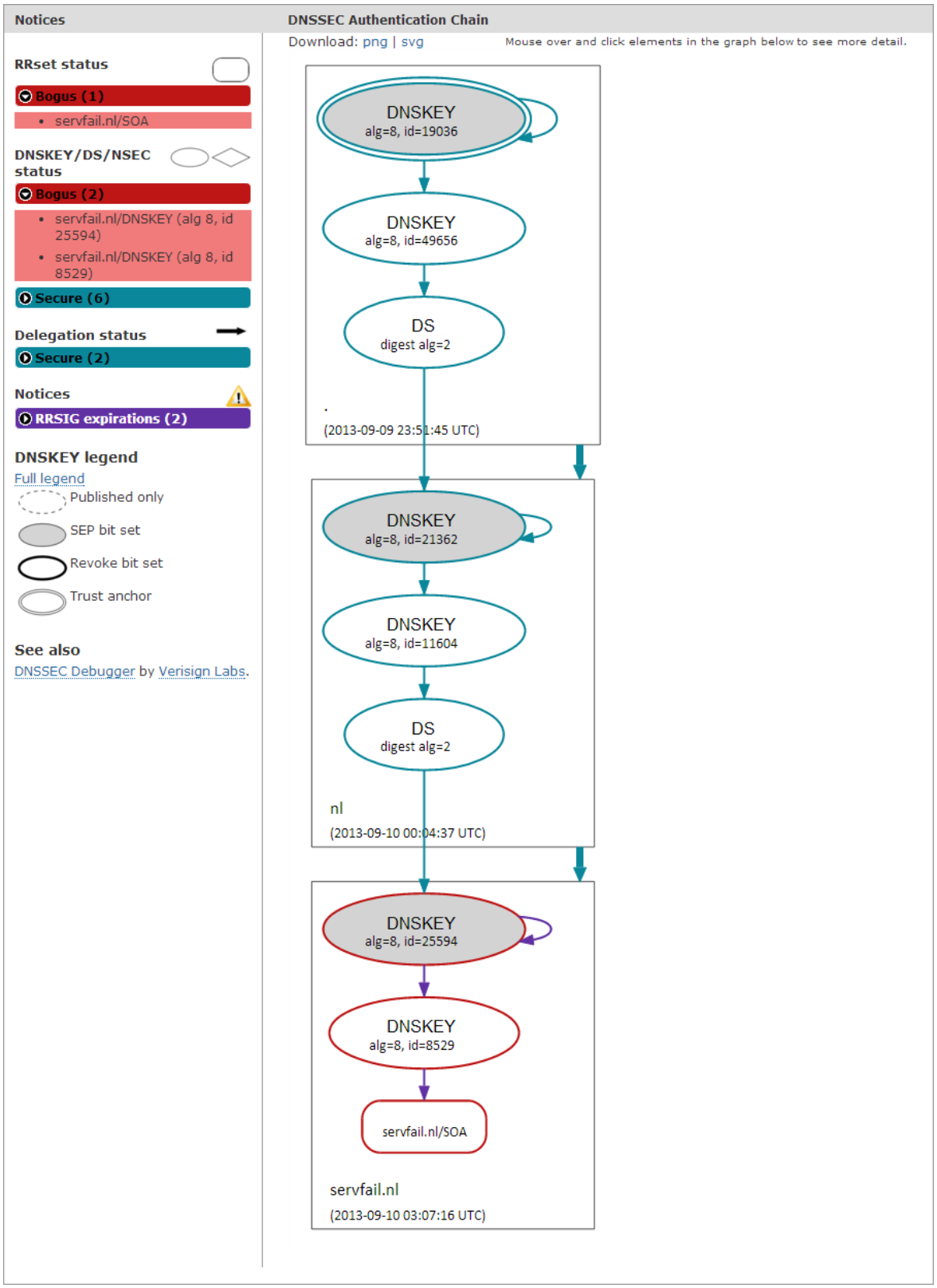


Figure 2. `servfail.nl` analyzed by DNSviz

To check all aspects of the configuration, not only DNSSEC, you can use Zonemaster. Here is a command line example (the -s option instructs it to test DNSSEC as well):

```
% zonemaster-cli rd.nic.fr
Seconds Level Message
======= ========= =======
2.63 NOTICE Nameserver ns2.rd.nic.fr has an IP address (192.134.4.81)
with mismatched PTR result (lea.rd.nic.fr.).
2.67 NOTICE Nameserver ns2.rd.nic.fr has an IP address (2001:67c:
2218:3::1:7) with mismatched PTR result (dalila.rd.nic.fr.).
16.77 NOTICE No target (MX, A or AAAA record) to deliver e-mail for
the domain name.
```

And here it is used on the domain that has a problem:

```
% zonemaster-cli servfail.nl
…
9.93 ERROR RRSIG with keytag 25594 and covering type(s) DNSKEY has
already expired (expiration is: 1326490909).
9.93 ERROR RRSIG with keytag 8529 and covering type(s) SOA has
already expired (expiration is: 1326490909).
10.09 ERROR Signature for DNSKEY with tag 25594 failed to verify with
error 'DNSSEC signature has expired'.
10.09 ERROR The apex DNSKEY RRset was not correctly signed.
...
```

Please note that if you are working with zone files[25] signed using a process which is not yet reliable, you can check the integrity of a signed file by using **validns**[26], an excellent and very fast tool (even with large zones):

```
% validns -p all -z dnssec.fr zones/dnssec.fr
%
```

The traditional `dig` tool can also be used, although it is mainly intended for people who know the DNS well. If your local resolver validates DNSSEC, `dig` will specify which zones are correctly signed with the `ad` (*Authentic Data*) flag:

```
% dig A www.afnic.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53599
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 6, AUTHORITY: 7, ADDITIONAL: 23
```

[25] When using BIND's auto-signing function, the zone file on the drive is in binary format and has to be converted to text using **named-compilezone -j -i none -f raw -F text -o dnssec.fr.signed-text dnssec.fr dnssec.fr.signed**.

[26] http://www.validns.net/

If the zone is incorrectly signed, **dig** will return a **SERVFAIL** (*SERver FAILure*):

```
% dig A www.servfail.nl
...
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 52126
```

Is it really a DNSSEC-related issue? Other reasons may explain this output. To find out, use **dig** with the **+cd** (*Checking Disabled*) option:

```
% dig +cd A www.servfail.nl
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53197
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1
...
;; ANSWER SECTION:
www.servfail.nl.        60      IN      CNAME   www.forfun.net.
```

Remember this simple rule: if you obtain a **SERVFAIL** by default, and a normal response with the **+cd** option, then the problem is DNSSEC-related.

Performing this test with a validating resolver is, in a way, the best option. After all, DNSSEC is used in order to allow validating resolvers to validate. But if you do not have a validating resolver yet, or if you are not sure of how to configure it correctly, then you can use a public resolver. OARC provides one, the ODVR[27]. It comprises two softwares, a BIND and an Unbound. Here is a test using the BIND server:

```
% dig +dnssec @2001:4f8:3:2bc:1::64:20 SOA dnssec.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18196
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 3
...
```

The **ad** (*Authentic Data*) flag is there. We shall now test a domain name that was not signed correctly using the other server of the ODVR, i.e. the Unbound server:

```
% dig +dnssec @2001:4f8:3:2bc:1::64:21 SOA servfail.nl
...
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 53610
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
```

A validating resolver can be used to test the signed zones from a monitoring software like Nagios. For example, the official Nagios plugin **check_dig** can be configured as follows:

```
define service{
       use                             generic-service
       host_name
                       résolveur_validant_1,résolveur_validant_2
       service_description             DNS
       check_command                   check_dig!-H $HOSTADDRESS$ -l
dnssec.fr -T SOA
       }
```

A Nagios alert will be triggered if the domain returns a **SERVFAIL**.

[27] https://www.dns-oarc.net/oarc/services/odvr

3.5. Monitoring

The solution shown here was tested with Icinga monitoring software but it is only based on tools compatible with the Nagios API, so it should work with many monitoring tools.

The basic tool is the Duane Wessels test script. Its specifications: it connects to all the DNS servers in a zone, requests the signatures, looks at the expiration dates and can report a warning or an error depending on thresholds chosen by the user. An example in hand:

```
% perl check_zone_rrsig_expiration -Z nic.fr
ZONE OK: No RRSIGs expiring in the next 3 days; (1.04s)
|time=1.042905s;;;0.000000
```

You can choose the thresholds, so imagine you want to issue a warning if there is less than one week left:

```
% perl check_zone_rrsig_expiration -Z nic.fr -W 7
ZONE WARNING: MX RRSIG expires in 3.7 days at ns6.ext.nic.fr; (0.28s)
|time=0.281515s;;;0.000000
```

To install and run this script, you need Perl and some of the modules indicated in the documentation. Be careful: if you do not install all the indicated packages, you'll get a message that is not clear at all:

```
***  WARNING!!!  The program has attempted to call the method
***  "sigexpiration" for the following RR object:
```

Once the program is correctly installed, I recommend option -d if you want to debug in detail what it does.
You then configure Icinga, for example:

```
define command{
        command_name    check-zone-rrsig
        command_line    /usr/local/sbin/check_zone_rrsig_expiration -Z
$HOSTADDRESS$ -W $ARG1$ -C $ARG2$
        }

...

define service {
       use dns-rrsig-service
       hostgroup_name My-zones
       service_description SIGEXPIRATION
       # Five days left: warning. Two days left: panic.
       check_command   check-zone-rrsig!5!2
}

define host{
        name                            my-zone
        use                     generic-host
 check_command              check-always-up
        check_period                    24x7
        check_interval                  5
        retry_interval                  1
        max_check_attempts              3
        contact_groups                  admins
        notification_period             24x7
        notification_options u,d,r
        register 0
}

define hostgroup{
       hostgroup_name My-zones
```

```
        members dnssec.fr,etc-etc
}

define host{
        use moi-zone
        host_name dnssec.fr
}
```

Once that is done, you restart Icinga. Here is a test with a deliberately broken zone (it was manually signed indicating the expiration date, without re-signing afterwards). Icinga sends this kind of warning:

```
Notification Type: PROBLEM

Service: DNSRRSIG
Host: broken.rd.nic.fr
Address: broken.rd.nic.fr
State: WARNING

Date/Time: Fri Mar 28 06:50:38 CET 2014

Additional Info:

ZONE WARNING: DNSKEY RRSIG expires in 1.2 days at ns2.dnssec.pm: (1.12s)
```

Then a CRITICAL, and then, once the zone really has expired:

```
Notification Type: PROBLEM

Service: DNSRRSIG
Host: broken.rd.nic.fr
Address: broken.rd.nic.fr
State: CRITICAL

Date/Time: Mon Mar 31 09:10:38 CEST 2014

Additional Info:

ZONE CRITICAL: ns2.dnssec.pm has expired RRSIGs: (1.10s)
```

If you then re-sign, the problem disappears:

```
Notification Type: RECOVERY

Service: DNSRRSIG
Host: broken.rd.nic.fr
Address: broken.rd.nic.fr
State: OK

Date/Time: Mon Mar 31 09:40:38 CEST 2014

Additional Info:

ZONE OK: No RRSIGs expiring in the next 3 days: (1.04s)
```

And in the Icinga log, the following will appear:

```
[Fri Mar 21 21:40:32 2014] SERVICE ALERT:
broken.rd.nic.fr;DNSRRSIG;CRITICAL;SOFT;2;ZONE CRITICAL: DNSKEY RRSIG
expires in 0.6 days at ns2.dnssec.pm: (1.09s)
...
```

```
[Fri Mar 21 21:42:32 2014] SERVICE ALERT:
broken.rd.nic.fr;DNSRRSIG;OK;SOFT;3;ZONE OK: No RRSIGs expiring in the next
7 days: (0.68s)
```

For users of OpenDNSSEC, the important parameter to be set at the same time as the monitoring thresholds is the <Refresh> interval. As indicated in documentation: "The signature will be refreshed when the time until the signature expiration is closer than the refresh interval." So in practice, what you need to indicate it is the duration, using the -C option (critical threshold). Be careful: OpenDNSSEC adds slight variations (jitter), so you need to specify a duration slightly longer than the <Refresh> interval.

3.6.Other solutions

Remember that there are other possibilities: there are of course many ways of using DNSSEC. We chose to focus on two methods (OpenDNSSEC + NSD or BIND with auto-dnssec), but there are others:

- The **PowerDNS**[28] software is very popular, especially in North European countries, and is used for many zones. It is used for **signing automatically**[29].
- OpenDNSSEC can also be used with BIND (and not with NSD like in the example above). In this case, `<NotifyCommand>rndc reload</NotifyCommand>` is typically used.
- There are several "appliances" that handle DNS and DNSSEC, such as **Efficient IP**[30], **Secure64**[31] or **InfoBlox**[32].

[28] **https://www.powerdns.com/**
[29] **http://doc.powerdns.com/html/dnssec-operational-doctrine.html**
[30] **http://www.efficientip.com/dnssec**
[31] **http://www.secure64.com/**
[32] **http://www.infoblox.com/solutions/best-practices/dns-security-center**

4. Conclusion

New techniques that help "improve" DNS attacks by poisoning are regularly released[33].

Working on deploying DNSSEC is therefore a necessity and contributes to the robustness of the shared resource that is DNS.

But DNSSEC remains a technique that requires skills, methodology and precision.

It is therefore important to ensure its proper deployment.

In addition to the documentation already mentioned, we would like to draw your attention to the *nist.dnssec-deployment* guide (§5.Bibliography) (this guide is very complete and is in English).

Furthermore, if you wish to follow practical training, with computer exercises, on the techniques presented in this document, please note that Afnic has a partnership with HSC and holds regular DNSSEC **training courses**.

[33] As of writing, the latest technique was presented at the end of July 2013 **by Haya Shulman** at the IETF in Berlin.

5.Bibliography

- *[afnic.dnssec-thema]* *Issue paper: DNSSEC*. 2010. Afnic. *Document in French.* **https://www.afnic.fr/medias/documents/afnic-dossier-dnssec-2010-09.pdf**

- *[bortzmeyer.dnssec-jres]* *Sécurité du DNS et DNSSEC*. 2009. Stéphane Bortzmeyer. *Document in French.* **https://2009.jres.org/planning_files/article/pdf/5.pdf**

- *[bortzmeyer.dnssec-satin]* *Monitoring DNSSEC zones: what, how and when?* 2011. Stéphane Bortzmeyer. http://conferences.npl.co.uk/satin/papers/satin2011-Bortzmeyer.pdf

- *[nist.dnssec-deployment]* *Secure Domain Name System (DNS) Deployment Guide*. 2010. Ramaswamy Chandramouli. Scott Rose. **http://csrc.nist.gov/publications/nistpubs/800-81r1/sp-800-81r1.pdf**

6. Glossary

BIND: *Berkeley Internet Name Domain* or *Berkeley Internet Name Daemon*. Name of the free software most commonly used for the implementation of DNS protocols. BIND consists of three parts: a domain name server, a client capable of querying other DNS servers, and technical testing tools.

DNS: *Domain Name System*. Distributed service which links Internet resources (IP addresses, e-mail relays, etc.) to domain names.

DNSKEY: DNS record used to store a public key.

DNSSEC: *Domain Name System Security Extensions*. Set of security extensions for the DNS protocol.

DS: *Delegation Signer*. DNS record corresponding to the public key hash.

EDNS: *Extension mechanisms for DNS* is an extension of the DNS protocol that increases the size of certain parameters, such as DNS responses, whose size used to be limited to 512 bytes.

KSK: *Key Signing Key*. Key that signs the ZSK keys. Its hash is published in the parent zone to create a chain of trust that guarantees its authenticity as well as that of the keys that sign the zone.

NS: *Name Server*. Also called a DNS server. Server used to host a domain name.

NSD: *Name Server Daemon*. Authoritative DNS server developed by NLnetLabs and used by many TLDs.

NTP: *Network Time Protocol*. Protocol which synchronizes local clocks within a computer network by redistributing a reference time.

OpenDNSSEC: Automatic DNSSEC key and signing management software.

Unbound: Validating DNS resolver maintained by NLnetLabs.

ZSK: *Zone Signing Key*. Key that signs the zone records. The public part of the ZSK is used to check the signatures.