Reference manual

version 2.6

YADIFA

Yet Another DNS Implementation For All

Contents

1 Introduction 7

1.1 Domain Name System . . . 8

1.1.1 Zones . . . 8

1.1.2 Authoritative name servers . . . 9

2 Resource Requirements 11

2.1 Hardware . . . 11

2.1.1 CPU . . . 11

2.1.2 Memory . . . 11

2.2 Supported Operating Systems . . . 11

3 Installation 13

3.1 Server . . . 13

3.2 Client . . . 13

3.3 Libraries . . . 13

3.4 From Sources . . . 14

3.4.1 Configure Options . . . 14

3.4.2 Server installation . . . 16

3.5 From Packages . . . 17

3.5.1 RHEL/CentOS/Fedora . . . 17

3.5.2 Debian . . . 18

3.5.3 Ubuntu . . . 18

3.5.4 Arch Linux . . . 19

3.5.5 Gentoo . . . 20

3.5.6 FreeBSD . . . 20

3.5.7 OpenBSD . . . 20

3.5.8 Solaris . . . 20

3.5.9 macOS . . . 20

4 Server Configuration 21

4.1 An authoritative name server . . . 23

4.1.1 Primary name server . . . 23

4.1.2 Secondary name server . . . 23

4.2 Signals . . . 24

5 Server Technical 25

5.1 Zone file reader . . . 25

5.1.1 Known types . . . 26
5.2 YADIFA . . . 26
5.2.1 Control commands . . . 27

6 Key Roll 38
6.1 Introduction . . . 38
6.2 Configuration . . . 39
6.3 Generate time format . . . 41
6.3.1 Command line . . . 42
6.3.2 Primary name server side setup . . . 43
6.3.3 *yakeyrolld* first sequence . . . 43
6.3.4 *yakeyrolld* runtime usage . . . 44
6.3.5 Extend the time covered by the steps . . . 44

7 Domain Name System Security Extensions (DNSSEC) 47
7.1 Introduction . . . 47
7.2 DNSSEC overview . . . 47
7.3 Types of key pairs . . . 49
7.4 Algorithms . . . 49

8 DNSSEC Policies 50
8.1 Introduction . . . 50
8.2 What is needed for DNSSEC? . . . 50
8.2.1 Keys for signing . . . 50
8.2.2 Signed zone . . . 52
8.2.3 Delegated zone . . . 53
8.3 What is needed for yadifa? . . . 53
8.3.1 Zone . . . 54
8.3.2 DNSSEC-Policy . . . 54
8.3.3 Denial . . . 55
8.3.4 Key Suite . . . 57
8.3.5 Key Template . . . 57
8.3.6 Key-roll . . . 58

9 DNS Name Server Identifier (NSID) 61
9.1 Introduction . . . 61
9.2 NSID payload . . . 61

10 DNS Response Rate Limiting (RRL) 63
10.1 Introduction . . . 63
10.2 What is it? . . . 63
10.3 The problem . . . 63
10.4 A solution . . . 64

11 RRSIG Update Allowed 65
11.1 Introduction . . . 65
11.2 The problem . . . 65
11.3 A solution . . . 65

12 Multi Primary Name Server 67
12.1 Introduction . . . 67
12.1.1 Design . . . 67
12.2 What is needed? . . . 74
12.2.1 Zone . . . 74

13 Configuration Reference 75
13.1 Layout . . . 75
13.2 Types . . . 77
13.3 Sections . . . 79
13.3.1 <*main*> section . . . 79
13.3.2 <*zone*> sections . . . 86
13.3.3 <*key*> sections . . . 88
13.3.4 <*acl*> section . . . 89
13.3.5 <*channels*> section . . . 91
13.3.6 <*loggers*> section . . . 94
13.3.7 <*nsid*> section . . . 97
13.3.8 <*rrl*> section . . . 98
13.3.9 <*dnssec-policy*> section . . . 99
13.3.10 <*key-suite*> section . . . 100
13.3.11 <*key-roll*> section . . . 100
13.3.12 <*key-template*> section . . . 101
13.3.13 <*denial*> section . . . 102

14 Zones 104
14.1 Macros . . . 104
14.1.1 @ . . . 105
14.1.2 $INCLUDE . . . 105
14.1.3 $ORIGIN . . . 106
14.1.4 $TTL . . . 107
14.2 Classes . . . 107
14.3 Resource record types . . . 107

15 Journal 110

16 Statistics 112

17 Configuration Examples 115
17.1 Introduction . . . 115
17.2 YADIFA as a primary name server . . . 116
17.2.1 The One That is Really Easy . . . 116
17.2.2 The One With Activation of Logging . . . 117
17.2.3 The One With NSID . . . 119
17.2.4 The One With RRL . . . 120
17.2.5 The One With ACL . . . 122
17.2.6 The One With DNSSEC Policy ‘diary’ style . . . 123
17.2.7 The One With DNSSEC Policy ‘relative’ style . . . 125
17.2.8 The One With RRSIG Update Allowed . . . 127

17.2.9 The One With the Controller 129

17.3 YADIFA as a secondary name server 131

17.3.1 The One With One Primary 131

17.3.2 The One With Several Primaries 132

17.3.3 The One With Activation of Logging 133

17.3.4 The One With NSID 135

17.3.5 The One With RRL 136

17.3.6 The One With ACL 138

18 Troubleshooting 139

18.1 Submitting a bug report 139

18.2 Stacktrace 142

18.2.1 Using a core dump 143

18.2.2 Running *yadifad* in the debugger 144

18.3 Building yadifad with even more debugging information 145

Bibliography 146

List of Figures

1.1 DNS hierarchy 9

17.1 Primary name server (simple configuration) 116
17.2 Primary name server with logging 117
17.3 Primary name server with NSID 119
17.4 Primary name server with RRL 120
17.5 Primary name server with ACL 122
17.6 Primary name server (DNSSEC policy ‘diary’ style) 123
17.7 Primary name server (DNSSEC policy ‘relative’ style) 125
17.8 Primary name server (RRSIG Update Allowed) 127
17.9 Primary name server with controller 129
17.10 Secondary name server (one primary) 131
17.11 Secondary name server (several primaries) 132
17.12 Secondary name server with logging 133
17.13 Secondary name server with NSID 135
17.14 Secondary name server with RRL 136
17.15 Secondary name server with ACL 138

1 Introduction

YADIFA is a *name server* implementation developed by *eurid*, the registry for the *.eu* top-level domain name. *eurid* developed *yadifa* to increase the robustness of the *.eu* name server infrastructure by adding a stable alternative to the other name server implementations in use.

In a nutshell, *YADIFA*:

- Is an authoritative name server, in both a primary and secondary *Domain Name System*[44] (*DNS*) server configuration
- Is *Request for Comments* (*RFC*) compliant
- Is portable across multiple Operating Systems including *GNU/Linux Operating System* (GNU/Linux), *Berkeley Software Distribution* (BSD) and *Apple's Operating System* (macOS)
- Is written from scratch in C. It is a clean implementation, which uses the *OpenSSL Cryptography and SSL/TLS Toolkit* (*OpenSSL*) or *Libre Secure Socket Layer* (*LibreSSL*) library.
- Supports *EDNS0*[58] (*EDNS0*)
- Supports *Domain Name System Security Extensions*[50] (*DNSSEC*) with *NSEC*[51] (*NSEC*) and *NSEC3*[11] (*NSEC3*)
- Has full and incremental zone transfer handling (*DNS Zone Transfer Protocol*[20] (*AXFR*) and *Incremental Zone Transfer*[45] (*IXFR*)).
- *DNSSEC* signing service

The *YADIFA* software package consists of:

- *yadifad* a daemon which serves *DNS*
- *yakeyrolld* a daemon which serves the key roll system
- *yadifa* an utility which can be used for controlling and configurating the name server *yadifad*.

In future releases new features will be added, including:

- Recursion
- Caching
- Validation
- Split horizon
- Plug-in system to integrate with *eurid*'s proprietary systems
- Dynamic provisioning of new domain names
- Have a backend which is Structured Query Language (SQL)-based[1]

1.1 Domain Name System

The *DNS* is a system and network protocol used on the Internet. *DNS* is a globally distributed database with domain names, which can translate those domain names into *INTERNET Protocol[47]* (*IP*) addresses and vice versa. All Internet-connected systems (routers, switches, desktops, laptops, servers, etc.) use *DNS* to query *DNS* servers for an *IP* addresses.

DNS is used by most services on the Internet. Mail, which itself uses the SMTP-protocol, uses *DNS* to get information about where to send emails.

DNS is an hierarchical, distributed system (see figure 1.1). One *DNS* server cannot hold all the information.

If you want to surf to *https://www.eurid.eu* for example, your computer needs the *IP* address of *www.eurid.eu*.

It first asks to the *. ZONE* (*root (.) zone*) name servers which guide you to the *.eu* name servers, which in turn guides you to the *eurid* name servers, where you will get the *IP* address of *www.eurid.eu*.

1.1.1 Zones

The information about a domain name can be found in **zones**. In these **zones** you will not only find a website's *IP* address, eg. *www.eurid.eu*, or a mail server's *IP* address, but also the information that points you to a subsection of the **zone**.

To clarify:
To find the *IP* address of *www.eurid.eu*, you start your search at the *root* server. You are not given the website's *IP* address, but are pointed in the direction where you will be able to find the information. The *root* server points you to a subsection of its zone, it points you to the name server(s) of *.eu*. This we call a *delegation*. The **zone** information has a *Name Server*[44] (*NS*)

[1] *YADIFA* will read zone from files and SQL-based backends

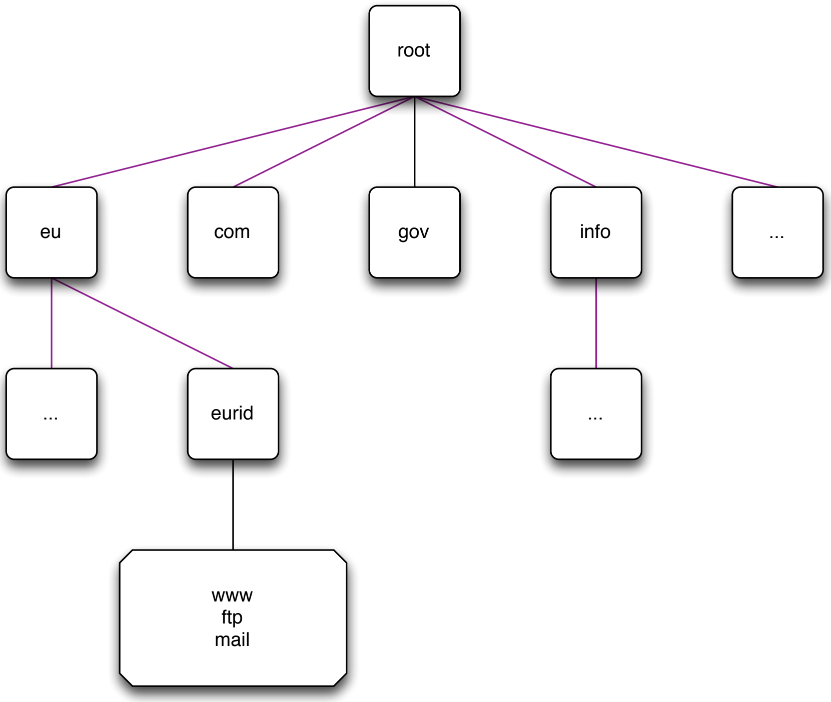

Figure 1.1: DNS hierarchy

resource record (*RR*) which contains the names of the *.eu* name servers. In the *.eu* zone information you will still not find the *IP* address of the *www.eurid.eu* website, but you will find the **delegation** to the next domain name, *eurid.eu*. In the name servers of *eurid.eu* you will find the *IP* address of *www.eurid.eu*.

1.1.2 Authoritative name servers

Name servers with all the information for a particular zone are the *authoritative name servers* for that zone. When querying the information of a domain name with an **authoritative** name server, the name server will give not only the answer, but will also indicate that it is **authoritative** for the information it has provided, by sending an **Authoritative Answer** flag along with the result.

For redundancy purposes a zone does not have only one authoritative name server. Good practice is to have a second and/or third name server in a different sub network.

Primary name server

Only one name server has the original zone information. Most name servers have this kind of information in a text file, also known as a **zone file**. Which authoritative name server is the *primary name server* of a domain name can be found in the *Start Of Authority* (*SOA*) *RR*. This information can be obtained from any of the domain name's authoritative name server(s).

Sometimes a *primary name server* is called **master name server**.

Secondary name server

The **secondary name server** has the same information as the *primary name server*, but differs in that it does not have the original *zone file*. A **secondary name server** receives its initial information from a transfer of the *primary name server*. There are several techniques for getting this information.

Sometimes a *secondary name server* is called **slave name server**.

2 Resource Requirements

2.1 Hardware

2.1.1 CPU

The *Central Processing Unit* (CPU) must be able to handle 64-bit integers (natively or through the compiler). It has to run a memory model where the data pointer size must be equal to the code pointer size. Threading is also required.

2.1.2 Memory

One record takes about 135 bytes of memory. Enabling *DNSSEC* is more expensive and triples that value. At runtime, zone management and processing may require additional storage space, up to 150% of the zone file size.

2.2 Supported Operating Systems

Please find below a list of operating systems and architectures we support and which are known to work.

[1] qemu-aarch64 emulated
[2] Pi Zero W
[3] Pi 2 Model B
[4] Pi 4
[5] missing required features

OS	x86-32	x86-64	arm64
Debian 8		YES	
Debian 9		YES	
Debian 10	YES	YES	YES[1]
Debian 11	YES	YES	YES[1]
Ubuntu 16.04 LTS		YES	
Ubuntu 18.04 LTS		YES	
Ubuntu 20.04 LTS		YES	YES[1]
Raspbian 9	N/A	N/A	YES[2]
Raspbian 10	N/A	N/A	YES[3][4]
CentOS 7	YES	YES	
CentOS 8.3	YES	YES	
AlmaLinux 8.5	YES	YES	
RHEL 7	YES	YES	
RHEL 8	YES	YES	
Fedora 32	N/A	YES	YES[2]
Fedora 33	N/A	YES	
Fedora 34	N/A	YES	
Arch	N/A	YES	N/A
FreeBSD 11.4-RELEASE		YES	
FreeBSD 12.2-RELEASE		YES	
FreeBSD 13.0-RELEASE		YES	
Windows 10	WSL2	WSL2	
MacOS 10.15	N/A	TODO	N/A
OpenBSD[5]	NO	NO	NO

SUPPORTED OSes

yadifa supports a number of different cryptographic backends. The *Secure Sockets Layer*[39] (*SSL*) backend needs to be chosen at compile time and can not be dynamically changed. Note that the client and server will only be able to use the algorithms supported by the *SSL* backend.

LIBRARY	VERSION
OpenSSL	1.1.1
LibreSSL	3.1.4

SUPPORTED SSL Libraries

The architecture of *yadifa* is very portable and will run on most flavours of GNU/Linux, but these configurations are untested.

3 Installation

The current version of *YADIFA* is: 2.6.2

YADIFA is a collection of two daemons, *yadifad*, *yakeyrolld*; one client, *yadifa*; three libraries; seven man pages, *yadifad.8*, *yadifa.8*, *yadifa.rc.5*, *yadifa.conf.5*, *yadifad.conf.5*, *yakeyrolld.conf.5* and *yakeyrolld.8*; and example configuration files.

3.1 Server

- Two daemon *yadifad*, *yakeyrolld*
- A man page *yadifad.8*
- A man page *yadifad.conf.5*
- A *yadifad.conf.example* file
- A man page *yakeyrolld.8*
- A man page *yakeyrolld.conf.5*
- A *yakeyrolld.conf.example* file.

3.2 Client

3.3 Libraries

- *dnscore*
- *dnsdb*
- *dnslg*.

3.4 From Sources

Everything can be installed in a *GNU Operating System* (GNU) fashion with *configure*, *make* and *make install*.

YADIFA will compile with many compilers. Please find below a list of versions that are confirmed in our test environment:

COMPILER	VERSION
GCC	4.8.5 / 4.9.2 / 5.4.0 / 6.3.0 / 7.5.0 / 8.3.0 / 8.3.1 / 9.3.0 / 10.2.0 / 10.3.1 / 11.0.0 / 11.1.0 / 11.1.1
CLANG	3.4.2 / 3.5.0 / 3.8.0 / 3.8.1 / 6.0.0 / 7.0.1 / 10.0.0 / 10.0.1 / 11.0.0 / 11.1.0 / 12.0.0

YADIFAD builds

YADIFA will work with other compilers. *GNU Compiler Collection* (*GCC*) 4.8.x has a bug which omits the file stdatomic.h, which is required by *YADIFA* to build. *YADIFA* sources contain the missing header file.

If you want to compile *YADIFA* for a certain compiler you need to add the "CC" environmental variable:

shell

```
$> ./configure CC=gcc-10
```

or

shell

```
$> ./configure CC=clang
```

3.4.1 Configure Options

You can configure *YADIFA* with several options, the most notable options available:

Functionality

OPTION	DESCRIPTION
–enable-shared	build shared libraries [default=no]
–enable-static	build static libraries [default=yes]
–disable-build-timestamp	Disable timestamps in the build
–disable-yadifa	Disable building the controller of yadifad
–disable-rrl	Disable DNS Response Rate Limiter
–disable-master	Disable DNS master
–disable-ctrl	Disable remote control support
–disable-nsid	Disable NSID support
–disable-dynupdate	Disable dynamic update support
–disable-rrsig-management	Disable RRSIG verification and generation for zones
–disable-zalloc	Disable zalloc memory system
–enable-log-thread-id	Enable write the thread id in each line of log
–disable-log-thread-tag	Disable a column with a 8 letters human-readable tag identifying a thread in each log line (overrides the thread id)
–enable-log-pid	Enable write the pid in each line of log
–enable-full-ascii7	Enable YADIFA will now accept ASCII7 characters in DNS names (not recommended)
–disable-ecdsa	Disable Elliptic Curve (ECDSA) support (i.e.: when the available SSL library does not supports it)
–enable-systemd-resolved-avoidance	Avoid conflict with systemd-resolved. Effectively changes the default value of ”do-no-listen” to ”127.0.0.53 port 53”.
–enable-non-aa-axfr-support	Enable Allows AXFR answer from master without AA bit set (Microsoft DNS)
–enable-lto	Enable LTO support, requires gold linker
–without-tools	build ”build without the DNS tools”
–without-tests	build ”build without the test programs”

CONFIGURE OPTIONS

Location

OPTION	DESCRIPTION
–prefix=PREFIX	install architecture-independent files in PREFIX [/usr/local]
–exec-prefix=EPREFIX	install architecture-dependent files in EPREFIX [PREFIX]
–bindir=DIR	user executables [EPREFIX/bin]
–sbindir=DIR	system admin executables [EPREFIX/sbin]
–sysconfdir=DIR	read-only single-machine data [PREFIX/etc]
–localstatedir=DIR	modifiable single-machine data [PREFIX/var]
–libdir=DIR	object code libraries [EPREFIX/lib]
–includedir=DIR	C header files [PREFIX/include]
–datarootdir=DIR	read-only arch.-independent data root [PREFIX/share]
–mandir=DIR	man documentation [DATAROOTDIR/man]
–docdir=DIR	documentation root [DATAROOTDIR/doc/yadifa]

CONFIGURE OPTIONS

3.4.2 Server installation

When installing *YADIFA* in /opt/, the install_prefix needs to be set to /opt/

shell

```
$> tar zxvf yadifa-2.6.2-10816.tar.gz
$> cd yadifa-2.6.2-10816
$>
$> ./configure --prefix=/opt/
$> make
$> sudo make install
```

After the installation a tree structure with files will have been created:

```
${install_prefix}/bin/
${install_prefix}/etc/
${install_prefix}/include/dnscore/
${install_prefix}/include/dnsdb/
${install_prefix}/include/dnslg/
${install_prefix}/lib/
${install_prefix}/sbin/
${install_prefix}/share/doc/yadifa
${install_prefix}/share/man/man5/
${install_prefix}/share/man/man8/
${install_prefix}/var/log/
${install_prefix}/var/run/
${install_prefix}/var/zones/keys/
```

```
${install_prefix}/var/zones/masters/
${install_prefix}/var/zones/slaves/
${install_prefix}/var/zones/xfr/
```

The most important files are found in:

```
${install_prefix}/etc/yadifad.conf
${install_prefix}/bin/yadifa
${install_prefix}/sbin/yadifad
${install_prefix}/sbin/yakeyrolld
${install_prefix}/share/man/man5/yadifa.conf.5
${install_prefix}/share/man/man5/yadifa.rc.5
${install_prefix}/share/man/man5/yadifad.conf.5
${install_prefix}/share/man/man5/yakeyrolld.conf.5
${install_prefix}/share/man/man8/yadifa.8
${install_prefix}/share/man/man8/yadifad.8
${install_prefix}/share/man/man8/yakeyrolld.8
```

An elaborate configuration can be found at:

```
${install_prefix}/share/doc/yadifa/yadifad.conf
```

Depending on the manner of compilation you will find the libraries in:

```
${install_prefix}/lib/
```

and the include files in:

```
${install_prefix}/include/dnscore/
${install_prefix}/include/dnsdb/
${install_prefix}/include/dnslg/
```

3.5 *From Packages*

3.5.1 RHEL/CentOS/Fedora

YADIFA source and binary packages are available from *Extra Packages for Enterprise Linux* (*EPEL*), provided by Denis Fateyev.

Preparation

For RHEL/CentOS, the *EPEL* repository is required. We would like to refer you to the proper installation guide at https://fedoraproject.org/wiki/EPEL.

Installation

Once the repositories are setup, installation can be completed using the following command:

shell

```
$> sudo yum install yadifa
```

3.5.2 Debian

Preparation

When using *Debian Linux Distro* (Debian) STABLE, the package is in the official stable repository since Debian 9 ”Stretch” and can be easily installed using the default package manager (See Installation).

Currently the version in Debian is version 2.4.2, if a more recent version is desired, it can be built manually from source. We are working to provide a newer version of the Debian package.

Installation

From the official repository:

shell

```
$> sudo apt-get install yadifa
```

3.5.3 Ubuntu

Preparation

The package is available through the official [universe] repository since Xenial Xerus (16.04 LTS)

shell

```
$> sudo apt-get install yadifa
```

For older versions of *Ubuntu Linux Distro* (Ubuntu), the package is not in the official repository and needs to be built manually.

Please follow the Debian build procedure.

3.5.4 Arch Linux

YADIFA is available from *Arch User Repository* (AUR), provided by BlackIkeEagle.

Preparation

You are encouraged to read *https://aur.archlinux.org* for a full description on how to use AUR.

The package is available at *YADIFA AUR*.

shell

```
$> curl https://aur.archlinux.org/cgit/aur.git/snapshot/yadifa.tar.gz -o
    ↪ yadifa.tar.gz
$> tar zxvf yadifa.tar.gz
$> cd yadifa
$> makepkg
```

Installation

Once the repositories are setup, installation can be completed using the following command:

shell

```
$> sudo pacman -U yadifa-2.6.2-1-x86_64.pkg.tar.xz
```

Or when you have installed pacaur, the preparation step can be skipped.

shell

```
$> sudo pacaur -S yadifa
```

3.5.5 Gentoo

Currently there is no emerge package available for *Gentoo Linux Distro* (Gentoo).

Please follow the source install option.

3.5.6 FreeBSD

YADIFA is available from *FreeBSD Operating System Ports* (FreeBSD Ports).

Installation

shell

```
$> cd /usr/ports/dns/yadifa && make install clean
```

YADIFA is now installed in /usr/local.

3.5.7 OpenBSD

OpenBSD Operating System (OpenBSD) is not supported as the Operating System is lacking required functionality.

3.5.8 Solaris

There are no packages available for *Solaris Operating System* (Solaris).

Please follow the source install option.

3.5.9 macOS

Currently there is no macOS package available.

Please use the source install.

4 Server Configuration

YADIFA is an authoritative name server only. Currently, it does not have the functionalities to be a *caching name server*, a *validating name server* or a *forwarder*.

YADIFA can start up without prior configuration, and it just requires an empty configuration file. Of course with an empty configuration file it does not do much, but you can test certain functionalities. It will answer queries, but with no zones configured it will return a flag which indicates that the query has been refused (*REFUSED*). This flag will be explained later in the manual.

All logs will be sent to the standard output.

The *YADIFA* configuration file has thirteen sections:

Eight standard sections:

- *"main" section* (see on page 79) (<*main*>)
- *"zone" section* (see on page 86) (<*zone*>)
- *"key" section* (see on page 88) (<*key*>)
- *"acl" section* (see on page 89) (<*acl*>)
- *"channels" section* (see on page 91) (<*channels*>)
- *"loggers" section* (see on page 94) (<*loggers*>)
- *"nsid" section* (see on page 97) (<*nsid*>)
- *"rrl" section* (see on page 98) (<*rrl*>)

And five sections for DNSSEC-Policy (see on page 50) (DNSSEC-Policy) only:

- *"dnssec-policy" section* (see on page 99) (<*dnssec-policy*>)
- *"key-suite" section* (see on page 100) (<*key-suite*>)

- *“key-roll” section* (see on page 100) (*<key-roll>*)
- *“key-template” section* (see on page 101) (*<key-template>*)
- *“denial” section* (see on page 102) (*<denial>*)

Each section has its own set of configuration elements.

- *<main>* contains all the configuration parameters needed to start up *YADIFA*
- *<zone>* contains all the configuration parameters needed for the zones
- *<channels>* and *<loggers>* are needed to configure your log information
- *<key>* contains *TSIG*[30] (*TSIG*) information
- *<nsid>* contains the “DNS Name Server Identifier Option”
- *<rrl>* contains the “Response Rate Limiting in the Domain Name System”.
- *<dnssec-policy>* (see chapter 8).

The configuration file also supports the use of **include**s. Included configuration files can itself contain **include** directives, with a maximum depth of 255. Relative path names will be treated as relative from the path of the configuration file where the **include** directive was defined.

configuration

```
<some_section>
 ...
</some_section>

include "../relative/to_this_file/include.conf" # with or without quotes
include include.conf                            # same directory as the
                                                # current file

<other_section>
 ...
</other_section>

include /absolute/path/to/file.conf             # absolute path
```

note

Included files are included in-line. This means the order is respected and later sections and configuration options overwrite previously defined options.

4.1 An authoritative name server

To allow *YADIFA* answering queries for its domain names, you have to declare them to the *zone* section.

4.1.1 Primary name server

An example of a zone with domain name .

configuration example

```
<zone>
    domain          somedomain.eu
    file            primaries/somedomain.eu.
    type            primary
</zone>
```

Where:

- **domain** is the *Fully Qualified Domain Name* (*FQDN*)
- **file** is the absolute or relative path of the zone file in text format
- **type** is the kind of name server *YADIFA* is for this zone. **Type** can be:
  - Primary
  - Secondary.

In this example, *YADIFA* is configured as a *primary* name server. This means that the original zone file is on this server, and you need to edit the zone file on this server.

note

For a working example you can find the zone file on page 104.

4.1.2 Secondary name server

YADIFA is authoritative for the zone , but does not have the original information. *YADIFA* needs to get the information from a *primary* name server for this zone file.

configuration example

```
<zone>
    domain          somedomain.eu
    file            secondaries/somedomain.eu.
    type            secondary
    primary         192.0.2.1
</zone>
```

In this example the **type** changes to *secondary*. *YADIFA* needs to know where it can find the primary zone file. This will be done with the additional configuration parameter **primary**, where you can specify the *IP* address of the primary name server for this domain name.

4.2 Signals

On a unix-like operating systems you can send a *signal* to a process, this is done with the *kill* command.

A few signals are implemented:

- **SIGTERM** will shut down *YADIFA* properly
- **SIGINT** will shut down *YADIFA* properly
- **SIGHUP** will reopen the log files and reload all updated zone files from disk. [1]
- **SIGUSR1** will save all zone files to disk. Zone files matching the zone in memory will not be overwritten.

For example:

shell

```
$> ps -ax | grep yadifad
$> 67071   2  S+     0:03.47 ./yadifad
$> kill -HUP 67071
```

[1] only the zone files with a higher serial number on disk than in the database will be affected

5 Server Technical

For now there are three entry points to the database:

1. Zone File
2. *AXFR* and *IXFR*
3. *Dynamic Updates in the Domain Name System*[12] (*DNS UPDATE*).

All three use the same principles to accept a *RR*:

- First-come, first-served
- Semantic errors will drop the relevant *RR*
- Syntax errors will drop the relevant entity.

Dropping the relevant entity can mean several things. If a syntax error occurs in a *DNS UPDATE* just this packet will be dropped and not the relevant zone file. A syntactical error can be a typo, but for security reasons the entity will be dropped completely.

If a syntax error is not a typo, but something against the *RFC*s, only that *RR* will be dropped.

5.1 *Zone file reader*

The zone file reader will check each *RR* as a single entity. Inconsistencies are only checked once the whole zone has been loaded.

What are inconsistencies?

- Semantics of a *RR*
- Non-existing MACROS/DIRECTIVES (e.g. typos in MACROS/DIRECTIVES)

- Multiple *SOA* records at the apex or an *SOA* record outside of the apex
- Forbidden *CNAME*[44] (*CNAME*) record(s) at the apex
- *CNAME*'s alongside records of types other than *Resource Record Signature*[51] (*RRSIG*) or *NSEC*
- No *NS* record found at the apex
- *Delegation Signer*[51] (*DS*) records without an *NS* record present
- *DS* records at the apex
- Unexpected records at a delegation
- Unexpected records under a delegation
- *RRSIG* records with an original *Time to Live* (*TTL*) not matching the covered type's
- *RRSIG* records signed by an *Domain Name System KEY*[51] (*DNSKEY*) not present in the zone
- *RRSIG* records covering a type not present in the domain
- *RRSIG* records covering *RRSIG* records
- *RRSIG* made with a *Key Signing Key*[51] (*KSK*) over a type other than *DNSKEY*.

5.1.1 Known types

For more information see section 14.3.

5.2 YADIFA

yadifa is the tool used to access the *yadifad* servers. *yadifa* can be used to configure a name server and control a name server.

yadifa communicates with the name server over a *Transmission Control Protocol[48]* (*TCP*) connection. This communication can be authenticated with *TSIG*'s. This *TSIG* can be given via the command line or a configuration file.

Default there's control support in *YADIFA*.

shell

```
$> ./configure
```

If you don't want to have control support in *YADIFA* you need to disable this function before compiling the sources.

shell

```
$> ./configure --disable-ctrl
```

After the *configure*, you can do the normal *make* and *make install*.

shell

```
$> make
$> make install
```

note

For control support you need to add **allow-control** in the <*main*> of *yadifad.conf* (13.3.1).

5.2.1 Control commands

For controlling *yadifad* the client, *yadifa*, is needed with its *control* module. The commands available in the *control* module can be seen with:

shell

```
$> yadifa ctrl help
```

TYPES	DOMAIN NAME	ARGUMENTS
cfgreload		
freeze	[<domain name>]	
logreopen		
notify	[<domain name>]	
querylog		[disable | enable]
reload	<domain name>	
shutdown		
sync	<domain name>	[clean]
unfreeze	[<domain name>]	
zonecfgreload	<domain name>	

COMMANDS

shell example

```
$> yadifa ctrl -s "192.0.2.1 port 53" -t <commands> -q [<domain name>] [<
    ↪ arguments>]
```

A more friendlier version of the commands can be used.

-s Can be reduced to @

-t Can be omitted in most cases

-q Can be omitted in most cases.

shell example friendlier version

```
$> yadifa ctrl @192.0.2.1 <commands> [<domain name>] [<arguments>]
```

In all commands the verbose -*v* option can be used:

shell example friendlier version

```
$> yadifa ctrl -v @192.0.2.1 <commands> [<domain name>] [<arguments>]
```

note

Where the non-verbose mode only gives back the exit code, you can have a more elaborated view in verbose mode.

cfgreload

This command will reload all keys, and the zones configurations and the zones. The port can be optionally supplied.

shell example

```
$> yadifa ctrl @192.0.2.1 cfgreload
```

Gives as result in verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 48854
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    CFGRELOAD

;; ANSWER SECTION:

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 3 msec
;; WHEN: Tue Jun 23 10:11:22 2020
;; MSG SIZE rcvd: 17
```

freeze

This command suspends updates to a zone. No further modifcations (*DNS UPDATE*) can be made. This command has a counterpart: *unfreeze*.

shell example

```
$> yadifa ctrl @192.0.2.1 freeze somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 30001
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    FREEZE

;; ANSWER SECTION:
.                        0       CTRL    FREEZE  somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:16:27 2020
;; MSG SIZE rcvd: 43
```

logreopen

This command reopens all log files.

shell example

```
$> yadifa ctrl @192.0.2.1 logreopen
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 59456
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    LOGREOPEN

;; ANSWER SECTION:

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 526 msec
;; WHEN: Tue Jun 23 10:18:03 2020
;; MSG SIZE rcvd: 17
```

loglevel

This command sets up the maximum level of log [0;15], 6 = INFO, 15 = ALL.

shell example

```
$> yadifa ctrl @192.0.2.1 loglevel -l 15
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 41914
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    LOGLEVEL
```

```
;; ANSWER SECTION:
.                         0       CTRL    LOGLEVEL  0f

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Thu Jul  2 13:53:04 2020
;; MSG SIZE rcvd: 29
```

notify

This command tells the server to send a notification to all the secondary name servers of a specific zone. If no name is provided, it tells the server to send a notification to all the secondary name servers of all of its zones.

shell example

```
$> yadifa ctrl @192.0.2.1 notify somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
(1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 28159
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                                 CTRL    NOTIFY

;; ANSWER SECTION:
.                         0       CTRL    NOTIFY  somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Fri Jun 26 15:35:46 2020
;; MSG SIZE rcvd: 43
```

querylog

This command enables or disables query logs.

shell example

```
$> yadifa ctrl @192.0.2.1 querylog enable
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 38288
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    QUERYLOG

;; ANSWER SECTION:
.                        0       CTRL    QUERYLOG  01

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 2 msec
;; WHEN: Tue Jun 23 10:18:42 2020
;; MSG SIZE rcvd: 29
```

shell example

```
$> yadifa ctrl @192.0.2.1 querylog disable
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 38290
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    QUERYLOG

;; ANSWER SECTION:
.                        0       CTRL    QUERYLOG  00

;; AUTHORITY SECTION:
```

```
;; ADDITIONAL SECTION:

;; Query time: 2 msec
;; WHEN: Tue Jun 23 10:19:42 2020
;; MSG SIZE rcvd: 29
```

reload

This command reloads the zone file from disk. If no parameter is given, '.' will be used as domain name.

shell example

```
$> yadifa ctrl @192.0.2.1 reload somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 8513
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    RELOAD

;; ANSWER SECTION:
.                        0       CTRL    RELOAD  somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:23:12 2020
;; MSG SIZE rcvd: 43
```

shutdown

This command shuts down the server.

shell example

```
$> yadifa ctrl @192.0.2.1 shutdown
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 28755
;; flags: qr ; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    SHUTDOWN

;; ANSWER SECTION:

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:43:25 2020
;; MSG SIZE rcvd: 17
```

sync

This command writes the zone to disk and optionally removes the journal. If no zone is specified, all zones are implied. The extra *clean* option will remove the journal.

shell example

```
$> yadifa ctrl @192.0.2.1 sync somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 36295
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    SYNC

;; ANSWER SECTION:
.                        0       CTRL    SYNC    00 somedomaineu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 2 msec
;; WHEN: Tue Jun 23 10:34:52 2020
;; MSG SIZE rcvd: 44
```

shell example

```
$> yadifa ctrl @192.0.2.1 sync somedomain.eu clean
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 63520
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    SYNC

;; ANSWER SECTION:
.                        0       CTRL    SYNC    01 somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:33:08 2020
;; MSG SIZE rcvd: 44
```

unfreeze

This command enables updates to a zone. Modifications (*DNS UPDATE*) can be done again. This command has a counterpart: *freeze*.

shell example

```
$> yadifa ctrl @192.0.2.1 unfreeze somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 6932
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
```

```
;.                               CTRL    UNFREEZE

;; ANSWER SECTION:
.                        0       CTRL    UNFREEZE  somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:35:43 2020
;; MSG SIZE rcvd: 43
```

zonecfgreload

This command rereads the zone config and reloads the zone file from disk.

shell

```
$> yadifa ctrl @192.0.2.1 zonecfgreload somedomain.eu
```

Gives as a result:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 1853
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    ZONECFGRELOAD

;; ANSWER SECTION:
.                        0       CTRL    ZONECFGRELOAD  somedomain.eu.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Jun 23 10:38:24 2020
;; MSG SIZE rcvd: 43
```

shell

```
$> yadifa ctrl @192.0.2.1 zonecfgreload
```

Gives as a result:

shell output

```
; (1 server found)
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 14197
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;.                               CTRL    ZONECFGRELOAD

;; ANSWER SECTION:

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 1 msec
;; WHEN: Tue Oct  6 11:02:55 2020
;; MSG SIZE rcvd: 17
```

6 Key Roll

6.1 Introduction

On a typical *DNSSEC* setup, the primary name server has access to both the *KSK* and the *Zone Signing Key*[51] (*ZSK*) key pairs.
The private part of the *KSK* is only required when the *ZSK* is replaced.
To avoid leaving the *KSK* key pairs unnecessarily vulnerable, one would only need to generate the *KSK* and *ZSK* in advance, as well as their associated *RRSIG DNSSEC RR*s.
This way, the primary name server needs only access to the *ZSK* key pairs.

The *yakeyrolld* program generates a sequence of *KSK* and *ZSK* for a zone, with all the steps of their lifecycles:

- Time of creation
- Time of publication
- Time of activation
- Time of de-activation
- Time of un-publication.

These times are determined using a *CRON* (*cron*)-like schedule.

For all these steps, it computes the following:

- The expected *DNSSEC* and *RRSIG DNSSEC RR*s on the primary name server before the step is started
- The *ZSK* files to add

- The *ZSK* files to remove
- The *DNSSEC* and *RRSIG DNSKEY RR*s to add
- The *DNSKEY* and *RRSIG DNSKEY RR*s to remove
- The expected *DNSKEY* and *RRSIG DNSKEY RR*s on the *DNS* primary name server after the step has been completed.

Each step is stored as a file. The file contains fields like:

PARAMETER	DESCRIPTION
epochus	An integer with the epoch of the step expressed in microseconds.
dateus	A user-friendly date text matching the epochus field.
actions	A list of actions expected to happen on the step (informational)
debug	A text meant to help understand the step (informational)
update	Each entry is a dynamic update command to be sent to the server.
expect	Each entry defines one record expected to be in the zone on the server prior to executing the current step.
endresult	Each entry defines one record expected to be in the zone on the server after the step has been executed.
add	Defines a key file to create in *keys-path*.
del	Names a key file to delete from *keys-path*.

TIME VALUES

Once this initialization is complete, *yakeyrolld* executes each step at their defined time: files are created or deleted, *RR*s are updated on the primary name server using the dynamic update protocol. If the primary name server's *RR* do not match the expectations, *yakeyrolld* will take measures to simply replace the *ZSK* files, the *DNSKEY* and *RRSIG DNSKEY RR*s on the primary name server completely with the expected ones.

6.2 Configuration

yakeyrolld requires a configuration file. By default, it is ${sysconfigdir}/yakeyrolld.conf ; however it can be changed using the *-c* command line option.

The sections of the configuration file are:

- yakeyrolld

- key
- channels
- loggers
- dnssec-policy
- key-suite
- key-template
- key-roll.

The *main* section is specific to *yakeyrolld*, all the others sections are defined exactly as in *yadifad* (see their description in the *DNSSEC* Policies chapter). The exception is there is no denial option in the dnssec-policy section as there is denial section either.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
damonise	BOOL	FALSE	Daemonises.
domain,domains	FQDN	.	Names one domain to manage, can be used up to 200 times.
generate-from	STRING	"now"	For plan generation, when to start the plan, can be overridden by the command line.
generate-until	STRING	"+1y"	For plan generation, when to stop the plan, can be overridden by the command line.
gid	GID	0	The gid to switch to. This should match the name server's.
keys-path	PATH	undefined	The directory the name server uses to read zone key file.
log-path	PATH	${localstatedir}/zones/keys	The directory that will contain the log files.
match-verify-retries-delay	INT	60	The number of seconds to wait between the match retries.
match-verify-retries	INT	60	If the content of the zone of a domain doesn't match expectations, *yakeyrolld* will retry for this amount of seconds.
pid-file	STRING	yakeyrolld.pid	The name of the pid file.
pid-path	PATH	${localstatedir}/run	The directory of the pid file.

plan-path	PATH	${localstatedir}/zones	The directory of the step files.
policy,policy-name	STRING	undefined	The name of the policy to use when generating the plan.
server	HOST	127.0.0.1	The address of the name server for queries and dynamic updates.
timeout	INT	3	The number of seconds spent trying to communicate with the primary name server until it's considered a time-out.
ttl	INT	600	The default *TTL* value to use when generating *RR*s.
uid	UID	0	The uid to switch to. This should match the name server's.
update-apply-verify-retries	INT	3600	If an update wasn't applied successfully, *yakeyrolld* will retry for this amount of seconds.
update-apply-verify-retries-delay	INT	60	The number of seconds to wait between the update retries.
wait-for-yadifad	BOOL	TRUE	If set to true, a starting-up *yakeyrolld* will wait for *yadifad* to be up and running with the expected domains before starting to send the planned updates.

YAKEYROLLD SECTION

6.3 Generate time format

The generate-from and generate-until time string is able to parse several kind of values:

PARAMETER	DESCRIPTION
now	Right now.
tomorrow	In 86400 seconds.
yesterday	86400 seconds ago.
+INTEGER*unit*	A number of time units after right now.
-INTEGER*unit*	A number of time units before right now.
YYYY-MM-DD	The absolute date at midnight.
YYYYMMDD	The absolute date at midnight.
YYYY-MM-DD_HH:MM:SS.UUUUUU	The absolute date and time to the microsecond.
YYYYMMDDHHMMSSUUUUUU	The absolute date and time to the microsecond.

TIME VALUES

Time units can be: years (366 days), months (31 days), weeks, days or seconds. *yakeyrolld* determines time units by matching the first letters: “s”, “sec”, “second”, “seconds” are equally supported.

Examples:

configuration

```
now
tomorrow
+1y
+1year
+2years
+1m
+1month
+2months
-1y
-1year
-1years
2019-04-16
2019-04-16_12:00:00.123456
20190416
20190416120000123456
```

6.3.1 Command line

short	long	description
-c	--config *configfile*	sets the configuration file to use
-m	--mode *mode*	sets the program mode (generate,play,playloop,print,print-json)
-d	--domain *fqdn*	the domain name
-p	--path *directory*	the directory where to store the keys
-s	--server *address*	the address of the server
-t	--ttl *seconds*	the *TTL* to use for both *DNSKEY* and *RRSIG* *RR*s
	--explain	prints the planned schedule
	--reset	start by removing all the keys and create a new *KSK* and a new *ZSK*. The server will not be queried.
	--policy	name of the policy to use
	--from *time*	the lower time bound covered by the plan (now)
	--until *time*	the upper time bound covered by the plan (+1y)

	--dryrun	do not write files to disk, do not send updates to the server
	--wait	wait for *yadifad* to answer before starting to work (default)
	--nowait	do not wait for *yadifad* to answer before starting to work
	--daemon	daemonise the program for supported modes (default)
	--nodaemon	do not daemonize the program
-Y	--noconfirmation	do not ask for confirmation before doing a data reset
-h	--help	shows the help
-V	--version	prints the version of the software

TIME VALUES

Any parameter given using the command line overrides the value it has in the configuration file.

6.3.2 Primary name server side setup

Every zone whose keys are managed by *yakeyrolld* must be configured on the name server.

- *RRSIG RR*s dynamic updates must be allowed (rrsig-nsupdate-allowed yes)
- updates coming from *yakeyrolld* must be allowed
- *DNSSEC* key files of the zone should be moved or removed as they may interfere with *yakeyrolld*
- the zone file must be setup for the same *DNSSEC* mode configured in *yakeyrolld* (e.g.: *NSEC*, *NSEC3*, *NSEC3-OPTOUT*[4] (*NSEC3-OPTOUT*)).

6.3.3 *yakeyrolld* first sequence

To generate the first sequence, one needs only to give the covered time period and specify to ignore the current content of the zone in the server (*–reset*).

e.g.:

shell

```
$> yakeyrolld -m generate --until +2y --reset
```

This command will generate all the steps required from that point in time and for a period of two years. The first update of the sequence will replace all the keys of the zone. The step files will be stored in the *plan-path/domain* directory. The *KSK* private keys will be stored in the *plan-path/domain.*_SECRET_KEYSIGNINGKEY directory.

6.3.4 *yakeyrolld* runtime usage

Once the step files have been generated, *yakeyrolld* can be started to execute them to make the zone on the server match the step active at the time of the command.

Simply executing the steps can be done using the *play* command.

e.g.:

shell

```
$> yakeyrolld -m play
```

yakeyrolld can be asked to play the sequence to the end, executing each step on time, by using the *playloop* command instead.

shell

```
$> yakeyrolld -m playloop
```

6.3.5 Extend the time covered by the steps

In order to add additional keys, simply call the generation with the appropriate duration and omit the –*reset* parameter.

shell

```
$> yakeyrolld -m generate --until +3y
```

yakeyrolld will add the missing steps to cover the time period and modify the existing ones if needed.

Note that *yakeyrolld* only loads the steps when starting up: any change made to the sequence requires restarting the program.

configuration

```
#
# Example yakeyrolld configuration file.
#

<yakeyrolld>
        domain  "example.eu"

        log-path "/opt/yakeyrolld/var/log/yakeyrolld"
        keys-path "/opt/yadifa/var/zones/keys"

        plan-path "/opt/yakeyrolld/var/zones"

        generate-from "now"

        generate-until "+1y"

        server 127.0.0.1

        policy "primary-policy"
</yakeyrolld>

<key>
        name        primary-secondary
        algorithm   hmac-md5
        secret      PrimaryAndSecondaryKey==
</key>

<channels>
    dnssec     dnssec.log     0644
    system     system.log     0644
    keyroll    keyroll.log    0644
    all        all.log        0644
</channels>

<loggers>
    system     prod system,all
    dnssec     prod dnssec,all
    keyroll    prod keyroll,all
</loggers>

<dnssec-policy>

    # name of the 'dnssec-policy'
    id                      "primary-policy"
    description             "primary zone policy"

    # at least one: key-descriptor "name"
    # they define KSK & ZSK keys

    key-suite               "zsk-2048"
    key-suite               "ksk-2048"
</dnssec-policy>

<key-suite>
    # name of the key-suite
    id                      "zsk-2048"
    key-template            "zsk-rsa-sha256-2048"
    # optional, without it, the keys found in the storage are used
    key-roll                "monthly-calendar"
</key-suite>

<key-suite>
    # name of the key-suite
    id                      "ksk-2048"
    key-template            "ksk-rsa-sha256-2048"
    # optional, without it, the keys found in the storage are used
    key-roll                "yearly-calendar"
</key-suite>

<key-template>
    id                  "zsk-rsa-sha256-2048"
    algorithm           RSASHA256
    size                2048
</key-template>

<key-template>
    id                  "ksk-rsa-sha256-2048"
    ksk                 1
    algorithm           RSASHA256
    size                2048
</key-template>

    #         min  hours days months weekdays weeks

<key-roll>
    id                  "yearly-calendar"

    generate  11   10     *   1      mon      1 # this year   (2018) 15/06 at 00:05
    publish   11   10     *   1      tue      * #                             00:10
    activate  11   10     *   1      wed      * #                    16/06 at 00:15
    inactive  11   10     *   1      mon      * #             (2019) 17/06 at 00:15
    remove    11   10     *   1      wed      * #             (2019) 18/06 at 11:15
</key-roll>

<key-roll>
    id                 "monthly-calendar"

    generate  17   10     *   *      mon      0 # 1 Monday    of month at 10:17
    publish   17   10     *   *      tue      * # 1 Tuesday   of month at 10:17
    activate  17   10     *   *      wed      * # 1 Wednesday of month at 10:17
    inactive  17   10     *   *      wed      * # 1 Wednesday of month at 10:17
    remove    17   10     *   *      thu      * # 1 Thursday  of month at 10:17
</key-roll>
```

7 Domain Name System Security Extensions (DNSSEC)

7.1 Introduction

The *DNS* provides responses without validating their source. This means that it is vulnerable to the insertion of invalid or malicious information, a flaw discovered by Dan Kaminsky in 2008.

This technical report documents the various components of the long-term solution to this kind of cache-poisoning attack: *DNSSEC*.

7.2 DNSSEC overview

In a nutshell, *DNSSEC* adds signatures to regular *DNS* responses in the form of *RRSIG*. A signature covers a resource record set (*RR set*). A *RR set* properly signed by a trusted source can be accepted as valid. Many signatures can cover the same *RR set*.

The *RRSIG RR* is consistent in a hash[1] of the covered *RR* set along with the validity period and other relevant information, signed with the private part of the owner's key pair [2].

To be able to verify whether the response is legitimate, the receiver of a signed response should verify that each *RR set* is verified by at least one of the signatures that covers it.

If this comparison shows no differences, the receiver is sure of two things:

- Integrity - the response has not been modified
- Authenticity - the response comes from the expected source (the only one to possess the private part of the key pair).

[1] A hash of a sequence of characters is the result of a one-way transformation of that sequence into a much smaller, fixed-length sequence by applying a certain mathematical formula. The slightest change of the original sequence changes the resulting hash. Thus, after transmission of the characters, one can detect changes to a sequence by comparing its current hash with the original.

[2] Public/private key encryption is well-known. A message is signed with the private part of a key pair (kept secret). The resulting signed message can only be verified using the public part of the key pair (shared with everybody).

Note that the response itself is not encrypted. *DNSSEC* adds *RRSIG* resource records (*RRs*) to responses, but the records that hold the data remain unaltered. In this way, *DNSSEC* is backwards compatible as non-*DNSSEC*-aware name servers can and should ignore unknown data and continue to function as expected.

The challenge in this scenario is to get the public part of the key pair to the users who need it for verification in a secure way.

The public parts of key pairs are available via the *DNS* as they are published as *DNSKEY RRs*. When querying for *DNSKEY* records, the response to a query also holds a signature for the *DNSKEY* record. But the question remains, should the receiver simply accept that the data is authentic and use it?

The answer is no. To verify the signature of a *DNSKEY* record, the user must consult the parent of the domain name. For domain names, such as *eurid.eu*, the parent is the *Top Level Domain Name[56]* (*TLD*). For a *TLD*, the parent is the *root (.) zone* domain name. To enable users to obtain the public part of a signed domain name in a secure way, a hash of the public key is put in the parent zone as a *DS RR*.

The parent zone signs the *DS RR* with its keys, authentifying the delegation in the process. In the case of *eurid.eu*, a hash of the public key (*DS*) is put in the *.EU* (*.eu zone*) where it is signed with the private key of *.eu zone*. For the *.eu zone* itself, a hash of the *.eu zone* public key (*DS*) is put in the *root (.) zone*, where it is signed with the private key of the *root (.) zone*.

This means that the receiver can obtain the public part of a key pair by querying for its hash in the parent zone, and, verify its signature with the public part of that parent-zone's key pair. This process only takes us up one level in the *DNS* hierarchy.

There the question repeats itself: how can the receiver trust the signature from that parent zone file? The answer lies in applying the same procedure: retrieving the public part of its key, the hash from its parent and the hash's signature.

But ultimately, some trust must be built in.

Herein lies the importance of having a signed Internet *root (.) zone*, because receivers that verify signatures only need to trust the public key of the *root (.) zone*. This is the only public key necessary and it can be obtained outside the *DNS*. It is available for download in several formats together with a signature file at: https://data.iana.org/root-anchors/. Before the *root (.) zone* was signed on 15 July 2010, administrators had to manually configure and maintain public key information from different branches in the *DNS* tree.

Now that the *root (.) zone* is signed, one can imagine how much effort *TLD* operators are putting into enabling *DNSSEC* on the domains they serve. Only a complete chain of trust allows the secure authentication of a domain name.

7.3 Types of key pairs

Two types of keys are used in *DNSSEC*:

- The *KSK* - used only to sign the hash of *DNSKEY* information
- The *ZSK* - used to sign the hashes of all *RR*s (*A*[44] (*A*), *NS*, *MX*[44] (*MX*), etc).

The more signatures generated with a particular key pair, the greater the chance of a successful crypto-attack, in other words deducing the private part of a key pair by using the public part and the available signatures. To prevent the signing of false information, key pairs should not be used indefinitely. Every so often, new key pairs should be generated and used to resign the zone. The frequency of key generation depends on the strength of the algorithm, key length and how often a key is used.

Because strong algorithms and long keys require more resources, such as more CPU, the practice is to use a weaker key pair, the *ZSK*, for all signatures but to change it regularly. Validity of these signatures should be three to six months at most. A stronger key pair, the *KSK*, is only used to sign the public key information. The *KSK* is changed less frequently, every one to two years. Only a hash of the *KSK* appears in the *root (.) zone* (as the *DS RR*). Since this key is changed, or rolled over, less often, interaction with the parent is less frequent.

7.4 Algorithms

Several algorithms for calculating hashes and signatures have been defined. Specific name server implementations or versions may not support all of the algorithms mentioned in the following summary:

RSA/SHA1 Algorithm number 5[2] (*RSASHA1*) is declared mandatory by **RFC_4034**[**51**]. RSASHA1-NSEC3 - SHA1 (algorithm number 7) is defined by **RFC_5155**[**11**]. It is essentially the same algorithm as *RSASHA1*, although the Next SECure records are *NSEC3*. The stronger algorithms, *RSA/SHA256 Algorithm number 8*[34] (*RSASHA256*) and *RSA/SHA512 Algorithm number 10*[34] (*RSASHA512*) are both defined by **RFC_5702**[**34**].

The use of these latter algorithms is recommended, as attacks against *Secure Hash Algorithm 1*[31] (*SHA1*) (used in algorithms 5 and 7) are increasing. Bear in mind that the newer algorithms, numbers 8 and 10, may not be available in older *DNS* server implementations and, as verifying *DNS* name servers that do not recognise an algorithm will treat the data as unsigned. It is unclear at the time of writing whether end users will actually benefit from these stronger algorithms.

8 DNSSEC Policies

8.1 Introduction

The *DNS* infrastructure is an integral and critical part of the Internet. With that said, the introduction of *DNSSEC* did not make life easier for the hostmaster. Generation of *KSK*'s and *ZSK*'s, in addition to signing the zone using "salt" and its iterations cause further complexity. To ensure that the keys will not be compromised, new keys must be generated continuously, at regular intervals, in a process called a "key roll over". When a key-roll over occurs, it is critical to not lose the integrity of the zone information. At no moment in time is it acceptable to have the zone unsigned or the keys, *KSK* and *ZSK*, outdated.

Due to these complex manipulations, especially on large amounts of zones in a portfolio, there is a need for an overall mechanism to facilitate *DNSSEC* enabled zones. Thanks to DNSSEC-Policies (see on page 50) (DNSSEC-Policies) the administrative overhead and complexity for *DNSSEC* enabled zones can be reduced significantly by generating and activating the keys automatically and maintaining the validity of the signatures.

8.2 What is needed for DNSSEC?

To implement *DNSSEC*, the following items are required:

- Keys for signing
- A signed zone
- A delegated zone.

8.2.1 Keys for signing

In *DNSSEC*, there are two different types of keys for signing the zone. The *KSK* and *ZSK*. The only difference in both keys is the use.

The *KSK* is used to sign the *DNSKEY RR* set only and has the *Secure Entry Point*[51] (*SEP*) bit set. The *ZSK* is used to sign each *RR set* of the zone. It is recommended to use a *KSK* in addition to a *ZSK*. The key size *KSK* should be larger, resulting in stronger cryptography and therefore can be rolled-over less often.

Each key consists of two parts: one private the other public.

Private Key

This key is used for signing all the *RR set*s. The signatures are stored in the *RRSIG RR*s and are only valid for a limited amount of time.

The current, most common format used to store a private key is depicted below:

```
Private-key-format: v1.3
Algorithm: 8 (RSASHA256)
Modulus: ...
PublicExponent: AQAB
PrivateExponent: ...
Prime1: ...
Prime2: ...
Exponent1: ...
Exponent2: ...
Coefficient: ...
Created: <create-date>
Publish: <publish-date>
Activate: <activate-date>
Inactive: <inactive-date>
Delete: <delete-date>
```

The fields: Created, Publish, Activate, Inactivate and Delete, indicate when the key must be used and when it must be removed from the zone.

- Created: Date the key was created
- Publish: Date the public part of the key is published in the zone
- Activate: Date the key should start signing the *RR set*s
- Inactivate: Date the key should stop signing the *RR set*s
- Delete: Date the public part of the key is removed from the zone.

Public Key

The public (part of the) key is used to verify the signatures generated by the private (part of the) key. The public key is published in the zone as the *DNSKEY*. The only difference between the

KSK and *ZSK* is the presence of the *SEP* bit, resulting in 257 flags for *KSK* instead of 256 for a *ZSK*.

```
somedomain.eu. IN DNSKEY 257 3 8 AwE...
```

DS

The *DS RR* is the cryptographic glue between the parent and delegated zone. This *RR* needs to be published in the parent zone and needs to correspond with *DNSKEY* in the delegated zone.

```
somedomain.eu.      86400   IN  DS  <keytag> 8 2 <hash_of_key>
```

8.2.2 Signed zone

A zone is signed when all the *RR* sets are signed by a valid *ZSK*. To be valid, the *ZSK* itself needs to be published as a *DNSKEY RR* and is to be signed by a *KSK*, which itself must also be published as a *DNSKEY*. The *KSK* must have a corresponding *DS RR* in the parent zone and must in turn be signed by the parent's *ZSK*.

Depending on your preferences and/or requirements, a choice between *NSEC* and *NSEC3* must be made to prove the Denial of Existence.

Signatures

Signatures are generated by the private key and stored in the zone as *RRSIG RR*s.

```
somedomain.eu.      86400   IN RRSIG   <type_covered> 8 2 86400 (
                              <end_date> <begin_date> <key_tag> <signer>
                              <signature> )
```

Denial of Existence

DNSSEC requires a cryptographic proof of non-existence. The zone is sorted by the labels and *NSEC* or *NSEC3 RR*s are generated representing the gaps between two subsequent labels. When a non-existing *RR* is requested, the *NSEC* or *NSEC3 RR* is returned in between the requested *RR* should have been found. The *NSEC* or *NSEC3 RR* are signed by an *RRSIG*.

For *NSEC*, the non-existence of would result in a reply similar to:

```
eu.                 7200    IN  NSEC    0.eu. NS SOA TXT RRSIG NSEC DNSKEY
somedicprod.eu.     7200    IN  NSEC    somedreams.eu. NS RRSIG NSEC
```

When using *NSEC3*, the mechanism is similar to *NSEC*, but all the *RR* are hashed before being sorted. The hashing algorithm, the salt and the number of times it should be hashed are stored in an *NSEC3PARAM*[11] (*NSEC3PARAM*) *RR* and are copied in each *NSEC3 RR*. In *NSEC3* there is an option to enable the *Opt-Out*[4] (*Opt-Out*). When this flag is set, only the zones for which there is a secure delegation will be considered for generating the *NSEC3 RR*s. Non-secure delegations will be treated as non-existent and will reduce the number of *NSEC3 RR* being created significantly.

```
QBQ65Q60970CPPR0EUCQNSC1FHE073UA.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                QBQ60CGMT2JNIJ4JNF2CCRFI4CE4NUE0
                NS SOA RRSIG DNSKEY NSEC3PARAM )
BKP4A7B3B0FKDVMPFABNCJ046PB2911A.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                BKPDVHUHA3S2PVTPI58DP5I5SABJUIM4
                NS DS RRSIG )
4EIAT7URLC7FMN9AGIJ231E2S7L62TG0.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                4EI0QGMMDB0BP76VHHBDNVEN2UUNABGK
                NS DS RRSIG )
```

8.2.3 Delegated zone

For *DNSSEC* to work, the whole chain up to the root must support *DNSSEC*. If the parent zone does not support *DNSSEC*, the chain cannot be verified and will not work.

8.3 What is needed for yadifa?

As there are a number of parameters to define, the components of *DNSSEC* policies span the following sections:

- *<zone>*
- *<dnssec-policy>*
- *<denial>*
- *<key-suite>*
- *<key-template>*
- *<key-roll>*.

8.3.1 Zone

Any zone can be handled by *DNSSEC* policies.

If a zone is activated to handle *DNSSEC* by DNSSEC-Policy, the keyword **dnssec-policy** with an associated **id** must be added.

configuration example of <*zone*> with **dnssec-policy**

```
<zone>
    domain          somedomain.eu
    file            primaries/somedomain.eu .
    type            primary

    dnssec-policy   "dp-1"
</zone>
```

8.3.2 DNSSEC-Policy

A DNSSEC-Policy configured zone needs <*dnssec-policy*> which has several keywords:

- **id**
- **denial**
- **key-suite**

configuration example of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    # name of the 'dnssec-policy'
    id                      "dp-1"

    denial                  "nsec3-denial"

    # at least one: key-descriptor "name"
    # they define KSK & ZSK keys

    key-suite               "zsk-1024"
    key-suite               "ksk-2048"
</dnssec-policy>
```

At least one <*key-suite*> must be configured. It is also recommended to have one *KSK* and one *ZSK*. *YADIFA* will only read the first four **key-suite**s.

The argument of **key-suite** is a string that must be unique per section type. It is possible, however, to configure several different sections with the same name (id). For example, in one configuration it is possible to have a <*denial*> and a <*key-suite*> with the same "id".

If <*dnssec-policy*> contains two or more **key-suite**s that contain the same content, only one <*key-suite*> will be applied.

Please note that the same algorithm should be used for both key signing and zone signing. This means for example that if you use *KSK* key-suite using the *RSASHA256* algorithm, you also need a *ZSK* key-suite using the *RSASHA256* algorithm.

> `note`
>
> `Clarifying the same content:`
>
> `If two <`*`key-suite`*`> have the same definition about keys in addition to the same time schedule regardless of their names (ids), only one will be applied while the other is silently ignored.`

8.3.3 Denial

The <*denial*> section contains several keywords:

- **id**
- **salt**[1]
- **salt-length**[1]
- **iterations**
- **optout**.

The zone can be signed with *NSEC* or *NSEC3*. If *NSEC3* has been chosen, salt will still need to be used for the *NSEC3PARAM* and the amount of iterations of this salt. In addition, the digest algorithm is also needed and is fixed to *SHA1*. This cannot be changed.

The choice between *NSEC* or *NSEC3* is done in the <*dnssec-policy*>.

Here are two examples:

- An example with the use of *NSEC*

[1] mutually exclusive, if both are defined, the system will refuse to start due to a parsing error

configuration example of <*dnssec-policy*> with *NSEC*

```
<dnssec-policy>
    id              "dp-1"

    denial          "nsec"
    ...
    ...
</dnssec-policy>
```

- An example with the use of *NSEC3*

configuration example of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    id              "dp-2"

    denial          "nsec3"
    ...
    ...
</dnssec-policy>
```

With the latter, “dp-2”, there is still a need for <*denial*>. In <*denial*> you need to add a “salt” which can be blank. The algorithm used for the hashing of the *NSEC3 RR* is always *SHA1* and cannot be changed. The parameters that can be set are: “**iterations**”, which is the amount of iteration done; the salt which can be set with the mutually exclusive: “**salt**” or “**salt-length**”; and “**optout**” to enable or disable the opt-out feature of *NSEC3*. When the opt-out feature is enabled, *RRSIG*s for insecure delegations are not generated, resulting in smaller zones while maintaining the security for secure delegations.

salt is used as keyword with argument a string. This string is *BASE16*[35] (*BASE16*) and is the actual salt. The keyword **salt-length** will generate a random string with the length provided as argument.

configuration example of <*denial*> with keyword **salt**

```
<denial>
    id              "nsec3"

    salt            "BA53BA11"
#   salt-length     4
    iterations      5
    optout          off
</denial>
```

note

Default value of **salt-length's** arguments is ''0''. There is no **salt** if **salt-length** is ''0''.

8.3.4 Key Suite

A zone file can have several keys.

Preferably a zone file is configured with two keys:

- *KSK*
- *ZSK*.

Configuration of the key is done in <*key-suite*>. The section has three keywords:

- **id**
- **key-template**
- **key-roll**.

key-template has the definition of the key and **key-roll** is the time schedule of the key.

configuration example of <*key-suite*>

```
<key-suite>
    id               "ksk-2048"

    key-template     "ksk-2048"
    key-roll         "yearly-schedule"
</key-suite>
```

note

A zone with only a *ZSK* is acceptable, but a zone with only a *KSK* is not.

8.3.5 Key Template

There are two kinds of keys:

- *KSK*

configuration example of <*key-template*> with a *KSK*

```
<key-template>
    id              "ksk-2048"

    ksk             true
    algorithm       8
    size            2048
</key-template>
```

- *ZSK*.

configuration example of <*key-template*> with a *ZSK*

```
<key-template>
    id              "zsk-1024"

    ksk             false
    algorithm       8
    size            1024
</key-template>
```

The arguments of **algorithm** and **size** keywords are referenced in the configuration reference chapter (13.3.12).

note

The **key-suite** is configured in <*dnssec-policy*>.

configuration example with of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    id              "dp-2"
    denial          "nsec3-denial"
    key-suite       "ksk-2048"
</dnssec-policy>
```

8.3.6 Key-roll

A *DNSSEC* key has a life-span. It starts with creating (generating) the key and ends with removing the key from the zone file.

A time schedule has several phases:

- Generate a key
- Publish a key in a zone
- Activate a key
- Inactive a key
- Remove a key from the zone.

The mechanism for changing one key with another is called a key-roll over. Key-roll overs follow the time schedule of a key. There are two kinds of "key-roll" mechanism:

- Relative
- Diary.

Key-roll mechanism "relative" style

configuration example of <*key-roll*> with relative mechanism

```
<key-roll>
    id                  "monthly-schedule"

    create              +30d
    publish             +2h
    activate            +7200 # 2 hours (in seconds)
    inactive            +31d
    delete              +7d

</key-roll>
```

The <*key-roll*> with the relative mechanism has an **id** and time phases.

The time phase keywords are:

- **create**
- **publish**
- **activate**
- **inactive**
- **delete**.

One time phase has a keyword with 2 arguments. The first argument is a time period with a resolution in seconds. The second argument is the dependency of a time phase with a previous one.

For example, **publish** will be done 2 hours after the **generate** time phase. The activate time phase will be done another 2 hours later after the **publish** time phase.

note

The first argument always starts with a plus-sign.

note

The resolution of key rolls are in minutes. The seconds will be rounded up to the minute.

If the second argument is not given default values will be used. (see section 13.3.11)

Key-roll mechanism "diary" style

configuration example of <*key-roll*> with diary mechanism

```
<key-roll>
    id                 "yearly-schedule"
    # this year   (2018) 15/06 at 00:05
    generate     5      0     15      6      *      *
    #                             00:10
    publish     10      0     15      6      *      *
                    
    #                    16/06 at 00:15
    activate    15      0     16      6      *      *
    #             (2019) 17/06 at 00:15
    inactive    15      0     17      6      *      *
    #             (2019) 18/06 at 11:15
    remove      15     11     18      6      *      *
</key-roll>
```

The <*key-roll*> with the diary mechanism has an **id** and time phases.

These "time phases keywords" are the same as those in the relative mechanism.

One time phase has one keyword with 6 arguments. The first argument is the minutes of the hour, the second is the hours of the day. The third argument is the day of the week, and the fourth is the month of the year. The fifth argument can be used to specify a day in a week (e.g. Wed for Wednesday). The last argument is the week number in the month.

note

A mix of relative mechanism and diary mechanism styles in one <*key-roll*> is not allowed.

See section 13.3.11 for further explanation.

9 DNS Name Server Identifier (NSID)

9.1 Introduction

The *DNS* infrastructure is an integral and critical part of the Internet and the robustness of this system has constantly been improved since it was first used. The increased robustness has lead to more complex setups where mechanisms like *DNS* anycast, name server pools and *IP* fail-overs allow different name servers to be available from a single *IP* address. These complex setups can make it very difficult to identify individual name servers. To identify different name servers, one could query for a specific record which is unique to each of the name servers. However, this method will not work for generic queries which comprise the bulk of all requests. *DNS Name Server Identifier*[8] (*NSID*) provides a solution by including a unique identifier within any *DNS* response. This feature is an extension of the *DNS* protocol. To allow backward compatibility, a name server that has the *NSID* extension will only send an *NSID* when it is explicitly asked for. The information, in response to the *NSID* option in the query, can be found in the EDNS OPT pseudo-*RR* in the response.

9.2 NSID payload

The *NSID* is a sequence of up to 512 arbitrary bytes set by the administrator. When queried, the byte sequence is usually represented as an hexadecimal string followed by its corresponding *American Standard Code for Information Interchange* (ASCII) chars, if possible.

The syntax and semantics of the content of the *NSID* option are deliberately left outside the scope of this specification.

Examples of *NSID*:

- It could be the “real” name of the specific name server within the name server pool
- It could be the “real” *IP* address (IPv4 or IPv6) of the name server within the name server pool
- It could be a pseudo-random number generated in a predictable fashion somehow using the server’s *IP* address or name as a seed value

- It could be a probabilistically unique identifier initially derived from a random number generator then preserved across reboots of the name server
- It could be a dynamically generated identifier so that only the name server operator could tell whether any two queries had been answered by the same server
- It could be a blob of signed data, with a corresponding key which might (or might not) be available via *DNS* lookups.

10 DNS Response Rate Limiting (RRL)

10.1 Introduction

A typical *Distributed Denial of Service* (DDoS) attack relies on a great number of hosts to send many requests simultaneously to disrupt a service. *DNS* is at the core of the Internet and when this service is disrupted, many other services are disrupted as well as collateral damage. Therefore, many *DNS* service providers have made major investments in good connectivity to mitigate attacks directed at their infrastructure. A *DNS* amplification attack is a special form of DDoS which takes advantage of the stateless nature of *DNS* queries to create forged *DNS* requests. Answers to these requests are sent to the actual target of the attack. The *DNS* protocol has been designed with efficiency in mind. Therefore, a typical request requires a minimal amount of bandwidth to the name server, but can trigger a huge response which is typically many times larger than the original request. These huge responses allow attackers to hedge their disposable bandwidth with the bandwidth available at some *DNS* servers by making them unwilling participants in this special form of DDoS.

10.2 What is it?

The *DNS* Response Rate Limiting (RRL) is an algorithm that helps mitigating *DNS* amplification attacks. The name servers have no way of knowing whether any particular *DNS* query is real or malicious, but it can detect patterns and clusters of queries when they are abused at high volumes and can so reduce the rate at which name servers respond to high volumes of malicious queries.

10.3 The problem

Any internet protocol based on *User Datagram Protocol*[46] (*UDP*) is suitable for use in a *Denial of Service* (DoS) attack, but *DNS* is especially well suited for such malevolence. There are several reasons:

- Reflected/Spoofed attack *DNS* servers cannot tell by examining a particular packet wether the source address in that packet is real or not. Most *DNS* queries are done by *UDP*. *UDP* does not have source address verification.

- Small *DNS* queries can generate large responses Especially when used with *DNSSEC*, the responses can be 10-20 (or more) times larger than the question.

10.4 A solution

If one packet with a forged source address arrives at a *DNS* server, there is no way for the server to tell it is forged. If hundreds of packets per second arrive with very similar source addresses asking for similar or identical information, there is a very high probability that those packets, as a group, form part of an attack. The RRL algorithm has two parts. It detects patterns in incoming queries, and when it finds a pattern that suggests abuse, it can reduce the rate at which replies are sent.

- Clients are grouped by their masked IPs, using **ipv4-prefix-length** and **ipv6-prefix-length**.
- Clients are kept in a table with a size varying from **min-table-size** to **max-table-size**.
- The **responses-per-second** is the maximum number of "no-error" answers that will be given to a client in the duration of a second.
- The **errors-per-second** is the maximum number of error answers that will be given to a client in the duration of a second.
- The **window** is the period for which the rates are measured. If the client goes beyond any of its allowed rates, then the majority of further answers will be dropped until this period of time has elapsed. Every **slip** dropped answers, a truncated answer may randomly be given, allowing the client to ask the query again using *TCP*.

11 RRSIG Update Allowed

11.1 Introduction

In a normal *DNSSEC* operation, the primary name server has a *KSK* and *ZSK* defined. The *ZSK* is used to sign the zone. To reduce the size and work to generate the *RRSIG*, the *ZSK* is usually a lot smaller than the *KSK*. To mitigate this problem, the *ZSK* is rolled (or replaced) much more often than the *KSK*.

The *KSK* is only used to sign the *DNSKEY RR set* and is usually only rolled once a year or even less frequently.

11.2 The problem

To sign a *RR set* and generate a *RRSIG*, the primary name server needs to have access to the private key of the *KSK* and *ZSK*. If the *ZSK* is compromised due to e.g. a break-in, it can be easily replaced by expediting the keyroll for the *ZSK*. Unfortunately if the primary name server is compromised, it is very likely that the *KSK* has also been compromised. The procedure to roll the *KSK* is a bit more cumbersome and involves making changes in the parent zone by updating the *DS*.

There are methods that the primary name server does not have access to the *KSK* and even the *ZSK*, but these options are often very expensive, slow, a hassle to set up properly and integrate within the existing infrastructure.

11.3 A solution

When an *DNS UPDATE* is sent to the primary name server, (re-)calculated. For this the private key needs to be available.

YADIFA can be configured to accept the *RRSIG* along with the nsupdate to the primary name server, eliminating the need to have access to the private key. The *RRSIG*s can then be computed on a different server.

The intended use for this option is to remove the need to have access to the *KSK* private key. The *KSK* is only required when there is a keyroll, which is only once a month or even less frequently.

Several *ZSK* keys can be pre-generated with the appropriate *RRSIG*s. Once done, the *KSK* can be taken off-line. At the appropriate times, the pre-generated *DNSKEY* updates and *RRSIG*s can be sent to the primary name server.

12 Multi Primary Name Server

12.1 Introduction

A multi primary name server configuration is a setup where more than one primary name server exists for the same zone and a secondary name server is configured to communicate with multiple primary name servers.

The benefit of having a multi primary configuration is that if one of the primary name servers is down or is in a maintenance mode the secondary name server can still request updates.

The secondary name server will listen to the notifications from all the primary name servers, but will always request the updates from the same preferred primary name server. When the preferred name server is unable to provide correct services, the next primary name server in the list of primary name servers (**primaries**) will be used. From then on, this primary name server has the highest priority in the list and becomes the new preferred primary name server.

12.1.1 Design

Whether a secondary name server is configured with a single primary name server or with multiple primary name servers, the design remains similar. The differences for the multiprimary design will be highlighted in this section of the manual.

Single Primary Name Server

When a secondary name server zone has a single primary name server configured, *YADIFA* will check the *SOA* serial on disk and request an *IXFR* from this serial to the (only) primary name server. If no files exist on disk, *YADIFA* will initiate an *AXFR*. When the transfer is successful, the zone is loaded. When notifications are received from the primary name server, it will check the serial in the notification and when the serial is absent or higher, *YADIFA* will initiate an *IXFR* with the current serial to the primary name server.

When a transfer error occurs, *YADIFA* will try to contact the primary name server again after a delay. The backing-off mechanism is explained in a different section.

configuration example

```
<zone>
    domain      somedomain.eu
    file        secondaries/somedomain.eu.
    type        secondary
    primaries   192.0.2.1
</zone>
```

Multiple primary name servers

When a secondary name server zone has multiple primary name servers configured, *YADIFA* will use the first configured primary as the preferred primary name server. In normal operations, it will behave identical to when only a single primary name server is defined with one minor difference. Notifications received from different (not the preferred) primaries, will be trigger the normal transfer procedure to the preferred primary name server. If the preferred primary name server itself is lagged with updates, *YADIFA* will not try to find the most current server (the highest serial), but keep itself in sync with the preferred primary name server. This is a deliberate design decision and will be explained later in this document.

The differences become apparent when a zone transfer fails. When the number of transfer failures exceed the **multiprimary-retries** option, the next primary name server will be selected as the new preferred primary name server. The previous preferred primary name server is added to the end of the list. The backing-off mechanism is explained in a different section.

note

A *Notification of Zone Changes*[50] (*DNS NOTIFY*) from a different primary will trigger the mechanism to update the zone, but *YADIFA* will keep itself in sync with the preferred primary only.

note

Except a reload of the configuration, transfer failures will be considered as the only reason to change the preferred primary name server.

The reason, be it a networking error, *Server Failure (rcode 2)*[44] (*SERVFAIL*), *Server Not Authoritative for zone (rcode 9)*[12] (*NOTAUTH*), *TSIG*, too slow, or anything else causing a transfer failure, is irrelevant for the switching decision. When the transfer is not successful, it is considered a failure.

configuration example

```
<zone>
    domain                    somedomain.eu
    file                      secondaries/somedomain.eu.
    type                      secondary
    primaries                 192.0.2.1,192.0.2.2,192.0.2.3
    multiprimary-retries 2
</zone>
```

In this example:

- The list of primaries is “192.0.2.1,192.0.2.2,192.0.2.3” and the first preferred primary is 192.0.2.1 and will be used to initiate a transfer of the zone
- When a *DNS NOTIFY* is received from any primary (e.g. 192.0.2.2) the *SOA* of the preferred primary name server 192.0.2.1 is checked. If the serial is bigger a transfer will be initiated from 192.0.2.1
- If the transfer from 192.0.2.1 fails 3 times (initial + 2 retries), the next primary in the list (192.0.2.2) will become the new preferred primary and the new list will be “192.0.2.2,192.0.2.3,192.0.2.1”.

note

When **true-multiprimary** is set to false (default), the secondary name server will not perform a (partial) zone transfer when switching to the new preferred primary name server with a lower or identical serial.

True

There are several scenarios in which an organisation runs several independent primary name servers. When there are independent primary name servers, we cannot be sure that the zone content on all the primaries is identical. Differences in update sizes or in jitter may cause differences in the zone content. The flag **true-multiprimary** should be used in this case.

The behavior of *YADIFA* is similar to a regular multi primary setup, with the difference that, when a new preferred primary is taken, the system will request a full zone transfer rather than an incremental. Updates from the same preferred primary will result in an *IXFR*. Switching to a different preferred primary should be avoided as it would otherwise result in a lot of unnecessary strain on the primaries, the secondaries and the network. Therefore, when a *notify* is received from a primary name server which is not the preferred primary, the serial of the preferred primary is checked. And an incremental transfer is initiated from the preferred primary name server when necessary.

> note
>
> When **true-multiprimary** is set to true, the secondary name server will always perform a full zone transfer when switching to the new preferred primary name server regardless of the serial number.

Backing-off mechanism

The back-off time before a new transfer is attempted, can be configured in the <*main*> section with the option **xfr-retry-delay**. A jitter can also be applied with the option **xfr-retry-jitter**. To increase the back-off time between failed transfers, two other parameters can be used: **xfr-retry-failure-delay-multiplier** and **xfr-retry-failure-delay-max**.

The formula for the backing-off mechanism is the following:

xfr-retry-delay + xfr-retry-jitter + min(failed-transfers * xfr-retry-failure-delay-multiplier ; xfr-retry-failure-delay-max)

configuration

```
<main>
    xfr-retry-jitter                    0       # Not possible, minimum 60
                                                # but makes the math clearer
    xfr-retry-delay                     200
    xfr-retry-failure-delay-multiplier  50
    xfr-retry-failure-delay-max         200
</main>

<zone>
    domain                  somedomain.eu
    file                    secondaries/somedomain.eu.
    type                    secondary
    primaries               192.0.2.1,192.0.2.2,192.0.2.3
    multiprimary-retries 6
</zone>
```

In this example:

The **xfr-retry-jitter** is ignored to make the example easier to explain.

Consider the following scenario:

- The preferred primary is 192.0.2.1 is unavailable, as is 192.0.2.2.
- An update to the zone is done.
- A *DNS NOTIFY* is received from 192.0.2.3.

YADIFA will do the following:

1. Check the *SOA* over *UDP* with the preferred primary name server 192.0.2.1 which fails
2. Initiate an *IXFR* with the current serial over *TCP* which also fails
3. *YADIFA* will wait 250 seconds (200 + 1 * 50) (first failure) which also fails
4. *YADIFA* will wait 300 seconds (200 + 2 * 50) (second failure) which also fails
5. *YADIFA* will wait 350 seconds (200 + 3 * 50) (third failure) which also fails
6. *YADIFA* will wait 400 seconds (200 + 4 * 50) (fourth failure) which also fails
7. *YADIFA* will wait 400 seconds (200 + 200) (fifth failure) which also fails
8. *YADIFA* will wait 400 seconds (200 + 200) (sixth failure) which also fails
9. *YADIFA* will wait 400 seconds and switch the preferred primary to 192.0.2.2 and transfer fails
10. *YADIFA* will wait 250 seconds (200 + 1 * 50) (first failure) which also fails
11. *YADIFA* will wait 300 seconds (200 + 2 * 50) (second failure) which also fails
12. *YADIFA* will wait 350 seconds (200 + 3 * 50) (third failure) which also fails
13. *YADIFA* will wait 400 seconds (200 + 4 * 50) (fourth failure) which also fails
14. *YADIFA* will wait 400 seconds (200 + 200) (fifth failure) which also fails
15. *YADIFA* will wait 400 seconds (200 + 200) (sixth failure) which also fails
16. *YADIFA* will wait 400 seconds and switch the preferred primary to 192.0.2.3 and transfer succeeds.

Design reasoning

The design of *YADIFA* takes the following into consideration, in order of importance:

1. The integrity of the zone content
2. The availability of the zone
3. The zone content is up-to-date.

In a secondary name server, there are 9 possible areas in which a zone file can be:

1. True Multiprimary ON in the zone section of the secondary name server

- Zone data, where the primary name server uses *DNS UPDATE* to update content and the zone file is *DNSSEC*
- Zone data, where the primary name server uses *DNS UPDATE* to update content
- Zone data, where the primary name server does not use *DNS UPDATE*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*
- Zone data, where the primary name server does not use *DNS UPDATE* and the zone data is *DNSSEC*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*.

The **true-multiprimary** option in *YADIFA* is used for installations where the zone content of the primary name servers is not identical. The reasons as to why the zone content is not identical is beyond the scope of this document.

Defining multiple primary name servers for a zone file indicates that, if the secondary name server is unable to transfer the zone from the preferred primary name server, the secondary name server will communicate with the next primary name server in its list of primary name servers for reception of its zone content.

As the zone content is not guaranteed to be identical, the only option is to perform a full transfer. With that said, as changing between primary name servers is very costly resource wise, *YADIFA* allows, tunable with several parameters, for the preferred primary to recover from any temporary issues that might otherwise lead to a switch. Altough networks have become very reliable, a *DNS NOTIFY* is sent through *UDP* which does not guarantee delivery. Therefore, when a *DNS NOTIFY* from a different primary is received, *YADIFA* will still check the *SOA* serial of the preferred primary in case the notify was lost.

In all the cases, (dynamic, static, *DNSSEC* or not *DNSSEC*), delaying a switch to a different primary name server will reduce the amount of wasted resources while maintaining the highest operational performance. The connection retries to the primary name server can be configured accordingly. If, after “X” retries no connection can be established with the primary name server, the second primary name server will take its place in the list, resulting in an *AXFR*.

`note`

`In true multi primary setups, the same or a higher serial does not mean that the zone content is more up-to-date.`

2. True Multiprimary OFF

   In this case, *YADIFA* considers all the primary name servers with the same serial as having identical zone data.

   - Zone data, where the primary name server uses *DNS UPDATE* to update content and the zone file is *DNSSEC*
   - Zone data, where the primary name server uses *DNS UPDATE* to update content
   - Zone data, where the primary name server does not use *DNS UPDATE*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*

- Zone data, where the primary name server does not use *DNS UPDATE* and the zone data is *DNSSEC*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*
- Zone data, where there is a single primary name server and intermediary primary name servers.

When *YADIFA* receives a *DNS NOTIFY*, it always communicates with the same primary name server for reception of the changes (*AXFR* or *IXFR*). If *YADIFA* receives a *DNS NOTIFY* that contains a *SOA* resource record with a lesser or equal serial than its own, it ignores the message.

However, for primary name servers using a dynamic zone file with *DNSSEC*, one REALLY cannot be sure, no matter the configuration, that the same *SOA* serial has the same zone data. This is due to jitter in signing the zone, resigning of the zone and dynamic updates which are never completely on the same time on all primary name servers. This results in the content not being 100 percent identical on all the primary name servers. In this case, **true-multiprimary** ON is the best and only choice.

Please Note: This relates to real primary name servers and not intermediary primary name servers.

In the other cases, assuming for the *DNSSEC* enabled zone that all the signatures are pre-calculated and that primary name server(s) are not responsible for maintaining the signatures, which would otherwise result in a scenario where true multiprimary would be preferred, we are absolutely sure that the content is identical. The zone content could be updated quickler by switching to the first primary name server for which a *DNS NOTIFY* is received.

If switching to a different primary name server could be performed with an incremental transfer, the cost of switching would be negligible and would result in the most up-to-date information for the secondary name server. Unfortunately, we cannot be sure that switching to a different primary will result in a small incremental transfer.

Some setups (e.g. without bind's **ixfr-from-differences** yes;) could result in an *AXFR* through an update while others have huge incremental updates. The primary name servers in the configuration may have other paths with different bandwidth restrictions and costs associated with them. Therefor the benefits of quickly switching to a different primary is uncertain therefore, the choice is given to the administrator to specify the most desirable primary. *YADIFA* will respect this choice by only switching when absolutely necessary.

For a hostmaster with thousands of zones to administer, the load between different primary name servers can be distributed by simply rotating the primary name servers in the configuration.

3. Round-robin scheme vs original preference list

   To avoid flapping services, we have opted to implement a round-robin scheme. When the first primary name server is known to be bad (configurable), the next primary name server in the list will become the new preferred primary. When the hostmaster has addressed the issue and wants to switch back to the "first" primary name server, this can be done by issuing a config reload.

12.2 What is needed?

12.2.1 Zone

configuration example of <*zone*> with several **primaries**

```
<zone>
    domain              somedomain.eu
    file                somedomain.eu.
    type                secondary

    primaries           192.0.2.1,192.0.2.2,192.0.2.3
    true-multiprimary   no  # 'no' is default, this line can be left out
</zone>
```

In this example the secondary name server listens to the notifications from the 3 primary name servers (primaries). The secondary name server will always ask for *DNS UPDATE*S from the first in the list. In this example: 192.0.2.1.

If the first primary name server no longer answers, the secondary name server will ask for updates from the second primary name server in the list, 192.0.2.2. From then on the secondary name server continues to ask that primary name server for updates until it no longer answers. Once that happens the secondary name server asks the next one in the list. After the last primary name server stops answering, the secondary name server starts from the first in the list, 192.0.2.1, again.

If the **true-multiprimary** is set to “no”, the secondary name server expects that all primary name servers are in sync and that their zone information is the same.

13 Configuration Reference

13.1 *Layout*

The configuration file has some rules:

- The configuration is read from a simple text file
- A comment starts after the "#" character
- Empty lines have no effect
- A string can be double-quoted, but is not mandatory.

The configuration file is made up of sections. A section starts with a <*name*> line and ends with a </*name*> line.

Currently, the following sections are implemented:

- *"main" section* (see on page 79) (<*main*>)
- *"zone" section* (see on page 86) (<*zone*>)
- *"key" section* (see on page 88) (<*key*>)
- *"acl" section* (see on page 89) (<*acl*>)
- *"channels" section* (see on page 91) (<*channels*>)
- *"loggers" section* (see on page 94) (<*loggers*>)
- *"nsid" section* (see on page 97) (<*nsid*>)
- *"rrl" section* (see on page 98) (<*rrl*>)
- *"dnssec-policy" section* (see on page 99) (<*dnssec-policy*>)
- *"key-suite" section* (see on page 100) (<*key-suite*>)

- *"key-roll" section* (see on page 100) (*<key-roll>*)
- *"key-template" section* (see on page 101) (*<key-template>*)
- *"denial" section* (see on page 102) (*<denial>*)

Unimplemented section names are ignored.

The section order is only of importance for sections of the same type where the principle first-found-first-processed applies. In other words, the last settings will overwrite earlier declarations of the same parameter. One exception is the *<zone>* section, where a declaration for the same domain will result in the error **DATABASE_ZONE_CONFIG_DUP**.

configuration example

```
<zone>
    domain   somedomain.eu
    file     primaries/somedomain.eu.zone
    type     primary
</zone>

<zone>
    domain   somedomain.eu
    file     primaries/someotherdomain.eu.zone
    type     primary
</zone>
```

In this example for the zone somedomain.eu, the *file* will be "primaries/*somedomain.eu*.zone".

The processing order of each section type is determined by the server implementation.
Each section contains settings. A setting is defined on one line but can be spread over multiple lines using parenthesis.

configuration example

```
# comment
# comment
<first>
# commment
    setting0-name value ...
    setting1-name value ...
</first>

<second>
    setting2-name (
        value
        ...
    )
# comment
</second>
```

13.2 Types

Each setting can be one of the following types.

TYPE	DESCRIPTION

ACL	A list of ACL descriptors. User-defined ACLs are found in the <*acl*> section. The “any” and “none” descriptors are always defined. Elements of the list are separated by a “,” or a “;”.
DNSSECTYPE	A *DNSSEC* type name. It can be a DNSSEC-enabled value (“nsec”, “nsec3” or “nsec3-optout”) or a DNSSEC-disabled value (“none”, “no”, “off” or “0”).
ENUM	A word from a specified set.
FLAG	A boolean value. It can be true (“1”, “enable”, “enabled”, “on”, “true”, “yes”) or false (“0”, “disable”, “disabled”, “off”, “false”, “no”).
FQDN	An *FQDN* text string. i.e.: www.eurid.eu.
GID	Group ID. (Can be a number or a name)
HOST(S)	A (list of) host(s). A host is defined by an IP (v4 or v6) and can be followed by the word ‘port’ and a port number. Elements of the list are separated by a ‘,’ or a ‘;’.
INTEGER / INT	A base-ten integer.
NETMOD	“single” or 0: Each working thread reads a single message, processes its answer and replies to it. “buffered” or 1 : Working threads are working by couple. One reads a single message and queues it, one de-queues it, processes its answer and replies to it. “multi” or 2 : Each working thread reads a multiple messages, processes their answers and replies to them.
PATH / FILE	A file or directory path. i.e.: “/var/zones”.
STRING / STR	A text string. Double quotes can be used but are not mandatory. Without quotes the string will be taken from the first non-blank charater to the last non-blank character.
HEXSTR	A hexadecimal even-length text string.
RELDATE	A cron-like date to be matched, relative to another. The columns are minutes [0;59], hours [0;23], days [0;31], months [1;12], weekdays [mon,tue,wed,thu,fri,sat,sun] and week-of-the-month [0;4]. Multiple values can be set in a column cell using ’,’ as a separator. The ’*’ character can be used to set all possible values of its column cell.
RELTIME	A time offset relative to another. It’s written as +integer[unit-character] (e.g.: +24h) where the unit character can be *s*econds, *m*inutes, *h*ours, *d*ays or *w*eeks.
SECONDS	A base-ten integer.
HOURS	A base-ten integer.
DAYS	A base-ten integer.
UID	User ID. (Can be a number or a name)

TYPES

13.3 Sections

13.3.1 <*main*> section

This section defines the global or default settings of the server.

PARAMETER	TYPE	DEFAULT	DESCRIPTION

[1] LOCALSTATEDIR is set at compile time; typically PREFIX/var or /var

allow-control	ACL	none	Default server-control access control list. Only the sources matching the ACL are accepted.
allow-notify	ACL	any	Default notify access control list. Only the servers matching the ACL will be handled.
allow-query	ACL	any	Default query access control list. Only the clients matching the ACL will be replied to.
allow-transfer	ACL	none	Default transfer access control list. Only the clients matching the ACL will be allowed to transfer a zone (*AXFR*/*IXFR*).
allow-update	ACL	none	Default update access control list. Only the clients matching the ACL will be allowed to update a zone.
allow-update-forwarding	ACL	none	Default update-forwarding access control list. Only the sources matching the ACL are accepted.
answer-formerr-packets	FLAG	true	If this flag is disabled; the server will not reply to badly formatted packets.
axfr-compress-packets; axfr-compresspackets; xfr-compresspackets	FLAG	true	Enables the *DNS* packet compression of each *AXFR* packet.
axfr-max-packet-size; axfr-maxpacketsize; xfr-maxpacketsize	INT	4096 bytes	The maximum size of an *AXFR* packet. (MIN: 512; MAX: 65535)
axfr-max-record-by-packet; axfr-maxrecordbypacket; xfr-maxrecordbypacket	INT	0	The maximum number of records in each *AXFR* packet. Older name servers can only handle 1. Set to 0 to disable the limit. (MIN: 0; MAX: 65535)
axfr-memory-threshold	HOST	65536	Any *AXFR* transfer up to that size (in bytes) will not be stored to the disk before being sent (MIN: 0; MAX: 1232896)
axfr-retry-delay;xfr-retry-delay	SECONDS	600	Number of seconds between each retry for the first transfer from the primary name server. (MIN: 60; MAX: 86400)
axfr-retry-jitter;xfr-retry-jitter	SECONDS	180	Jitter applied to axfr-retry-delay. (MIN: 60; MAX: axfr-retry-delay)
axfr-retry-failure-delay-multiplier;xfr-retry-failure-delay-multiplier	INT	5	Linear back-off multiplier. The multiplier times the number of failures is added to the xfr-retry-delay. (MIN: 0; MAX: 86400)

axfr-retry-failure-delay-max;xfr-retry-failure-delay-max	SECONDS	3600	Maximum delay added for the back-off. (MIN: 0; MAX: 604800)
axfr-strict-authority	FLAG	yes	Tells *yadifad* to be strict with the AA flag in *AXFR* answers
chroot	FLAG	off	Enabling this flag will make the server jail itself in the chroot-path directory.
chroot-path; chrootpath	PATH	/	The directory used for the jail.
cpu-count-override	INT	0	Overrides the detected number of logical CPUs. Set to 0 for automatic. (MIN: 0; MAX: 256)
daemon; daemonize	FLAG	false	Enabling this flag will make the server detach from the console and work in background.
data-path; datapath	PATH	zones[1]	The base path were lies the data (zone file path; journaling data; temporary files; etc.)
do-not-listen	HOSTS	-	An exclusion list of addresses to never listen to. If set, 0.0.0.0 and ::0 will always be split by interface to isolate the address.
edns0-max-size	INT	4096	*EDNS0* packets size. (MIN: 512; MAX: 65535)
gid; group	GID	0 (or root)	The group ID that the server will use.
hidden-primary; hidden-master	FLAG	no	As a hidden primary more CPU will be used for various maintenance tasks.
hostname-chaos; hostname	STR	the host name	The string returned by a hostname-chaos TXT CH query.
keys-path; keyspath	PATH	zones/keys[1]	The base path of the *DNSSEC* keys.
listen	HOSTS	0.0.0.0,::0	The list of interfaces to listen to.
log-files-disabled	FLAG	no	If set, disables checking the log-path directory for existence and writing rights.
log-path; logpath	PATH	log[1]	The base path where the log files are written.
log-unprocessable	FLAG	off	Enabling this flag will make the server log unprocessable queries.
max-tcp-queries; max-tcp-connections	INT	128	The maximum number of parallel *TCP* queries; allowed. (MIN: 1; MAX: 255)
network-model	NETMOD	multi	Sets the networking model of *YADIFA*.
pid-file; pidfile	STR	run/yadifad.pid[1]	The pid file name.

queries-log-type	INT	1	Query log format. (0: none; 1: *YADIFA* format; 2: BIND format; 3: *YADIFA* and BIND format at once)
serverid-chaos; serverid	STR	-	The string returned by a id.server. TXT CH query. If not set; REFUSED is answered.
server-port; port	INT	53	The default *DNS* port. (MIN: 1; MAX: 65535)
sig-validity-interval	DAYS	30	The number of days for which an automatic signature is valid. (MIN: 7 days; MAX: 30 days)
sig-validity-jitter; sig-jitter	SECONDS	3600	The signature expiration validity jitter in seconds (1 hour). (MIN: 0 sec; MAX: 86400 sec)
sig-validity-regeneration	HOURS	automatic	Signatures expiring in less than the indicated amount of hours will be recomputed. The default will be chosen by *YADIFA*. (MIN: 24 hours; MAX: 168 hours)
statistics	FLAG	true	The server will log a report line about some internal statistics.
statistics-max-period	SECONDS	60	The period in seconds between two statistics log lines. (MIN: 1 sec; MAX: 31 * 86400 seconds (31 days))
tcp-query-min-rate	INT	512 bytes/second	The minimum transfer rate required in a *TCP* connection (read and write). Slower connections are closed. The units are bytes per second. (MIN: 0; MAX: 4294967295)
thread-affinity-base	INT	0	Sets the first CPU to set affinity for. Set it to the real CPU of a core. (MIN: 0; MAX: 3)
thread-affinity-multiplier	INT	0	Sets the multiplier choosing CPU to set affinity for. Allows avoiding hyperthread cores. Set to 0 for automatic avoiding. (MIN: 0; MAX: 4)
thread-count-by-address	INT	-1	Number of independent threads used to process each listening address. Set to -1 for automatic. Set to 0 for single threaded. (MIN: -1; MAX: number of CPU's)
transfer-source	HOST	not set	If set, *AXFR* and *IXFR* queries will be made specifically from this address
uid; user	UID	0 (or root)	The user ID that the server will use.
version-chaos; version	STR	yadifa version#	The text to include in the version TXT CH query.

xfr-connect-timeout	SECONDS	5	Timeout for establishing a connection for *AXFR* and *IXFR* transfers. Set to 0 to disable. (MIN: 0; MAX: 4294967295)
xfr-path; xfrpath	PATH	zones/xfr[1]	The base path used for *AXFR* and journal storage.
zone-download-thread-count	INT	4	Number of independent threads used to download the zones. (MIN: 0; MAX: 255)
zone-load-thread-count	INT	1	Number of independent threads used to process loading of the zones. (MIN: 0; MAX: 255)
zone-store-thread-count	INT	1	Sets the number of threads used to store a zone on disk (MIN: 1, MAX: 4).
zone-unload-thread-count	INT	1	Sets the number of threads used to delete a zone from memory (MIN: 1, MAX: 4).
worker-backlog-queue-size	INT	16384	For network-model 1, sets the size of the backlog queue (MIN: 4096, MAX: 1048576).

MAIN SECTION

The dnssec-mode configuration parameter is a hint to yadifad on how the zone should be configured.

When YADIFA loads a zone file, it automatically detects its DNSSEC mode. The hint helps to decide how to maintain the zone and to issue warnings when expectation differs.

Detecting the DNSSEC mode of a zone file is often a relatively trivial task. Some cases may not be as straightforward:

When yadifa loads a zone, the dnssec-mode guessing works like this:

- If the zone file contains a DNSKEY with an NSEC3-capable algorithm (All algorithms from 6 and above) contains an NSEC3PARAM, it is considered to be NSEC3. The optout variant is detected if there is at least one delegation without a DS resource record set that is not being covered by an NSEC3 record.
- If the zone file contains a DNSKEY with an NSEC3-only algorithm (6 and 7, a.k.a. DSA-NSEC3-SHA1 and RSASHA1-NSEC3-SHA1) but not NSEC3PARAM, an error will be logged and the zone will be DNSSEC none.
- If the zone file contains a DNSKEY with an NSEC-only algorithm (3 and 5, a.k.a. DSA and RSASHA1) but has no NSEC records, its mode is set to NSEC. The presence of an NSEC3PARAM in such a zone will log an error and will not be loaded.
- If the zone file contains a DNSKEY with a modern algorithm (All algorithms from 8 and above) and has no NSEC3PARAM, its mode is set to NSEC.

- If the zone file contains a DNSKEY with a modern algorithm (All algorithms from 8 and above) and has no NSEC3PARAM, but NSEC3 records exist, the zone will log an error and the will not be loaded.

configuration example

```
<main>
    chroot                      on
    daemonize                   true
    chroot-path                 /srv/yadifa/var
    keys-path                   /zones/keys
    data-path                   /zones
    log-path                    /log
    pid-path                    /run
    pid-file                    yadifad.pid

    cpu-count-override          6
    dnssec-thread-count         10
    max-tcp-queries             100
    tcp-query-min-rate          6000

    additional-from-auth        yes
    authority-from-auth         yes
    answer-formerr-packets      no

    listen                      192.0.2.53, 192.0.2.153 port 8053

    hostname                    my-shown-hostname
    serverid                    ns-loc-01

    user                        yadifad
    group                       yadifad

    statistics                  yes
    statistics-max-period       60

    # could have been written as: 'version not disclosed' without the '
    version                     "not disclosed"

    # note: Any is default anyway
    allow-query                 any
    allow-update                operations-network ; public-network
    allow-transfer              secondaries ; operations-network ; public-network

    sig-validity-interval       360
    sig-validity-regeneration   48
    sig-validity-jitter         1800

    axfr-max-record-by-packet   0
    axfr-max-packet-size        32768
    axfr-compress-packets       true
</main>
```

PARAMETER	TYPE	DEFAULT	DESCRIPTION
enabled	FLAG	YES	If enabled, yadifad will process controller queries.

MAIN SECTION

13.3.2 <*zone*> sections

Each zone is defined by one section only.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
allow-control	ACL	as main	Control commands control list. Only the matching sources are allowed.
allow-notify	ACL	as main	Notify access control list. Only the servers matching the ACL will be handled.
allow-query	ACL	as main	Query access control list. Only the clients matching the ACL will be replied to.
allow-transfer	ACL	as main	Tansfer access control list. Only the clients matching the ACL will be allowed to transfer a zone (*AXFR/IXFR*
allow-update	ACL	as main	Update access control list. Only the clients matching the ACL will be allowed to update a zone.
allow-update-forwarding	ACL	as main	Update forwarding control list. Only the matching sources are allowed.
dnssec-mode; dnssec	DNSSEC-TYPE	off	Type of *DNSSEC* used for the zone. As primary name sever; *YADIFA* will try to maintain that state.
dnssec-policy	STR	-	Sets the dnssec-policy id to be used.
domain	FQDN	-	Mandatory. Sets the domain of the zone (i.e.: eurid.eu).
drop-before-load	FLAG	off	Enabling this flag will make the server drop the zone before loading the updated zone from disk. Use this on systems constrained for *Random-access memory* (RAM).

file-name; file	FILE	-	Sets the zone file name. Only mandatory for a primary zone. Relative paths to <*main*> **data-path**
journal-size-kb; journal-size	INT	0	Puts a soft limit on the size of the journal; expressed in KB. (MIN: 0; MAX: 3698688 (3GB))
keys-path; keyspath	PATH	as main	The base path of the *DNSSEC* keys.
maintain-dnssec	FLAG	true	Enabling this flag will cause the server to try and maintain *RRSIG* records
primaries; primary; masters; master	HOSTS	-	Mandatory for a secondary. Sets the primary server(s). Multiple primaries are supported.
multiprimary-retries; multimaster-retries	INT	0	The number of times the primary is unreachable before switching to a different primary. (MIN: 0; MAX: 255)
no-primary-updates; no-master-updates	FLAG	false	Enabling this flag will prevent the server from probing or downloading changes from the primary.
notifies; also-notify; notify	HOSTS	-	The list of servers to notify in the event of a change. Currently only used by primaries when a dynamic update occurs.
notify-retry-count; retry-count	INT	5	Number of times *YADIFA* tries to send a *DNS NOTIFY*. (MIN: 0; MAX: 10)
notify-retry-period; retry-period	INT	1	Time period in minutes between two *DNS NOTIFY* attempts. (MIN: 1; MAX: 600)
notify-retry-period-increase; retry-period-increase	INT	0	Increase of the time period in minutes between two *DNS NOTIFY* attempts. (MIN: 0; MAX: 600)
rrsig-nsupdate-allowed; rrsig-push-allowed	FLAG	false	If this flag is set the server allows to edit *RRSIG* records using dynamic updates.
sig-validity-interval; signature-validity-interval	DAYS	as main	The number of days for which an automatic signature is valid. (MIN: 7 days; MAX: 30 days)
sig-validity-regeneration; signature-regeneration	HOURS	as main	The signatures expiring in less than the indicated amount of hours will be recomputed. (MIN: 24 hours; MAX: 168 hours)
sig-validity-jitter; signature-sig-jitter; signature-jitter; sig-jitter	SECONDS	as main	The signature expiration validity jitter in seconds. (MIN: 0 sec; MAX: 86400 sec)

true-multiprimary; true-multimaster	FLAG	off	Enabling this flag will make the server use *AXFR* when switching to a new primary.
transfer-source	HOST	not set	if set, *AXFR* and *IXFR* queries will be made specifically from this address
type	ENUM	-	Mandatory. Sets the type of zone : either **primary/master** or **secondary/slave**.

ZONE SECTION

sig-* and allow-* settings defined here have precedence over those in the <*main*> section.

configuration example

```
<zone>
    domain                       somedomain.eu.
    type                         primary
    file-name                    primaries/somedomain.eu-signed.txt

    # The rest is not mandatory ...

    also-notify                  192.0.2.194, 192.0.2.164

    # Doing this is pointless since it's both the global setting AND
    # the default one

    allow-query                  any
    allow-update                 my-network; 127.0.0.1
    allow-transfer               my-secondaries

    # Same as global setting
    sig-validity-interval        720             # 30 days is enough
    sig-validity-regeneration    12
    sig-validity-jitter          7200

    journal-size-kb              64               # 64 KB
</zone>

<zone>
    domain                       someotherdomain.eu
    type                         secondary
    primary                      192.0.2.53
</zone>
```

13.3.3 <*key*> sections

Each TSIG key must be defined by one section.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
algorithm	ENUM	-	Mandatory. Sets the algorithm of the key. Supported values are : 'hmac-md5', 'hmac-sha1', 'hmac-sha224', 'hmac-sha256', 'hmac-sha384', 'hmac-sha512' (the algorithm names are case insensitive)
name	FQDN	-	Mandatory. Sets the name of the key.
secret	TEXT	-	Mandatory. Sets the value of the key. BASE64 encoded.

KEY SECTION

configuration example

```
<key>
    name            yadifa
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>

<key>
    name            eu-secondary1
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>

<key>
    name            eu-secondary2
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>
```

13.3.4 <*acl*> section

Each entry of the acl section defines a rule of access. Each rule is a name (a single user-defined word) followed by a rule in the form of a list of statements. The separator can be "," or ";". The "any" and "none" names are reserved. A statement tells if a source is accepted or rejected. Reject statements are prefixed with "!". Statements are evaluated in the following order: first the IP addresses are checked using a fallthrough method. If a statement matches, the evaluation will stop and accordingly accept or reject the source. When there is no IP match, the keys are checked. If no statement matches, then the source is rejected.

Note: a key can not be limited to a single IP address or range.

A statement can be either:

- An IPv4 or an IPv6 address followed (or not) by a mask.
  [!]ipv4|ipv6[/mask]

  For example:

  configuration sample

  ```
  internal-network            192.0.2.128/26;2001:DB8::/32
  ```

- The word 'key' followed by the name of a TSIG key.
  key key-name

  For example:

  configuration sample

  ```
  secondaries                 key public-secondary;key hidden-secondary
  ```

- An ACL statement name from the <*acl*> section. Note that negation and recursion are forbidden and duly rejected.
  acl-name

  For example:

  configuration sample

  ```
  who-can-ask-for-an-ixfr     internal-network;secondaries;127.0.0.1
  ```

configuration example

```
<acl>
    # user-defined-name     rule-statements

    # rule to accept this TSIG key

    secondary1              key eu-secondary1
```

```
    # rule to accept that TSIG key

    secondary2              key eu-secondary2

    # rule to accept what the secondary1 and secondary2 rules are accepting

    secondaries             secondary1;secondary2

    # rule to accept this IP
    primary                 192.0.2.2

    # rule to accept both this IPv4 network and that IPv6 network
    operations              192.0.2.128/28;2001:DB8::/32

    # the order of each ACL statement is important: the following rule

    order-example-1         key eu-secondary1; 192.0.2.128/26 ; ! 192.0.2.133 ; 192.0.2.5; !
    ↪ 192.0.2.0/26

    # will be understood internally the same way as this one

    order-example-2         192.0.2.128/26 ; 192.0.2.5 ; ! 192.0.2.0/26 ; key eu-secondary1

    # Note that '! 192.0.2.133' will not be considered as the IP is included in the
    ↪ 192.0.2.128/26 range declared earlier.
    # To have individual IP addresses considered, they need to be declared before the block,
    ↪ as was done with 192.0.2.5

    # Using this acl example, access is granted to all hosts from 192.0.2.128/26 and
    ↪ 192.0.2.5, regardless if key eu-secondary1
    # is used or not. Hosts from 192.0.2.0/26 are denied regarless if key eu-secondary1 is
    ↪ used or not.
    # Any host not matching the IP ranges 192.0.2.128/26 or 192.0.2.0/26 will only be
    ↪ accepted if key eu-secondary1 is used.


</acl>
```

13.3.5 <*channels*> section

Channels are loggers output stream definitions. Three types are supported:

- file
- STDOUT, STDERR
- syslog.

Each channel is a name (a single user-defined word) followed by:

- the “syslog” keyword, defining a channel to the syslog daemon. The keyword can be followed by case-insensitive facilities and options arguments. These arguments will be given to syslog. Note that only one facility should be given.

  Supported facilities:

PARAMETER	DESCRIPTION
auth	Security/authorisation messages (DEPRECATED: use authpriv)
authpriv	Security/authorisation messages (private)
cron	Clock daemon (cron and at)
daemon	System daemons without separate facility value
ftp	Ftp daemon
local0	Reserved for local use
local1	Reserved for local use
local2	Reserved for local use
local3	Reserved for local use
local4	Reserved for local use
local5	Reserved for local use
local6	Reserved for local use
local7	Reserved for local use
lpr	Line printer subsystem
mail	Mail subsystem
news	USENET news subsystem
syslog	Messages generated internally by syslogd(8)
user	Generic user-level messages
uucp	UUCP subsystem

CHANNELS SECTION

  Supported options:

PARAMETER	DESCRIPTION
cons	Write directly to system console if there is an error while sending to system logger.
ndelay	Open the connection immediately (normally, the connection is opened when the first message is logged).
nowait	Don’t wait for child processes that may have been created while logging the message (On systems where it is relevant).
odelay	Opening of the connection is delayed until syslog() is called (This is the default, and need not be specified).
perror	(Not in POSIX.1-2001.) Print to stderr as well.
pid	Include PID with each message.

note

For more information: man syslog

For example:

configuration sample

```
syslog  syslog  CRON,PID
```

- The “STDOUT” case-sensitive keyword, defining a channel writing on the standard output.

  For example:

  configuration sample

  ```
  default-output STDOUT
  ```

- The “STDERR” case-sensitive keyword, defining a channel writing on the standard error.

  For example:

  configuration sample

  ```
  default-error STDERR
  ```

- A relative file path, defining a channel writing on a file (append at the end). The file is followed by the file rights as an octal number.

  For example:

  configuration sample

  ```
  yadifa yadifa.log 0644
  ```

configuration example

```
<channels>
    # user-defined-name      parameters

    # channel 'statistics': a file called stats.log
    #                        with 0644 access rights
    #
    statistics               stats.log 0644
```

```
    # channel 'syslog' :    a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa': a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>
```

13.3.6 <*loggers*> section

YADIFA has a set of log sources, each of which can have their output filtered (or ignored) and sent to a number of channels.

A logger line is defined as the source name followed by the list of levels and then the list of channels. The lists are "," separated.

The current set of sources is:

SOURCES	DESCRIPTION
database	Database output (incremental changes, integrity checks, etc.)
dnssec	*DNSSEC* output (*NSEC*, *NSEC3*, signatures events)
server	Server actions output (network setup, database setup, queries, etc.)
stats	Internal statistics periodic output
system	Low-level output (thread management, task scheduling, timed events)
zone	Internal zone loading output
queries	Queries output

LOGGERS SECTION

The current set of levels is:

LEVELS	DESCRIPTION
emerg	System is unusable
alert	Action must be taken immediately
crit	Critical conditions
err	Error conditions
warning	Warning conditions
notice	Normal, but significant, condition
info	Informational message
debug	Debug-level 0 message
debug1	Debug-level 1 message
debug2	Debug-level 2 message
debug3	Debug-level 3 message
debug4	Debug-level 4 message
debug5	Debug-level 5 message
debug6	Debug-level 6 message
debug7	Debug-level 7 message
prod	All non-debug levels
all	All levels
*	All levels

LEVELS

note

Messages at the ''crit'', ''alert'' and ''emerg'' levels do trigger an automatic shutdown of the server.

If the logger section is omitted completely, everything is logged to the STDOUT channel. Negations are not allowed.

configuration

```
<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>
```

The defined loggers are:

system contains low level messages about the system such as memory allocation, threading, IOs, timers and cryptography, . . .

database contains messages about most lower-level operations in the *DNS* database. ie: journal, updates, zone loading and sanitization, *DNS* message query resolution, . . .)

dnssec contains messages about lower-level dnssec operations in the *DNS* database. ie: status, maintenance, verification, . . .

server contains messages about operations in the *DNS* server. i.e.: startup, shutdown, configuration, transfers, various services status (database management, network management, *DNS* notification management, dynamic update management, resource rate limiting, . . .)

zone contains messages about the loading of a zone from a source (file parsing, transferred binary zone reading, . . .)

stats contains the statistics of the server. (See chapter 16)

queries contains the queries on the server. Queries can be logged with the bind and/or with the *YADIFA* format.

bind format:

client sender-ip#port: query: fqdn class type +SETDC (listen-ip)

***YADIFA* format:**

query [id] {+SETDC} fqdn class type (sender-ip#port)

where:

id is the query message id
+ means the message has the Recursion Desired flag set
S means the message is signed with a *TSIG*
E means the message is EDNS
T means the message was sent using *TCP* instead of *UDP*
D means the message has the DNSSEC OK flag set
C means the message has the Checking Disabled flag set
fqdn is the queried *FQDN*
class is the queried class
type is the queried type
sender-ip is the IP of the client that sent the query
port is the port of the client that sent the query
listen-ip is the listen network interface that received the message

Note that on *YADIFA* any unset flag is replaced by a "-", on bind only the "+" follows that rule.

System operators will mostly be interested in the info and above messages of queries and stats, as well as the error and above messages of the other loggers.

13.3.7 <*nsid*> section

note

If you want to have *NSID* support in *YADIFA* you need to enable this function before compiling the sources.

shell

```
$> ./configure --enable-nsid
```

After the ‘‘configure’’, you can do the normal ‘‘make’’ and ‘‘make install’’.

shell

```
$> make
$> make install
```

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ascii hex	STR	“” “”	The string can be 512 characters long.

NSID SECTION

configuration example ascii

```
<nsid>
    ascii belgium-brussels-01
</nsid>
```

configuration example hex

```
<nsid>
    hex   00320201
</nsid>
```

13.3.8 *<rrl>* section

YADIFA has support for RRL enabled by default.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
enabled	FLAG	false	Enables the RRL
errors-per-second	INT	5	Allowed error rate
exempt-clients,exempted	ACL	none	Clients matching this rule are not subject to the RRL
ipv4-prefix-length	INT	24	Mask applied to group the IPv4 clients
ipv6-prefix-length	INT	56	Mask applied to group the IPv6 clients
log-only	FLAG	false	If set to true, logs what it should do without doing it
max-table-size	INT	16384	RRL buffer maximum size
min-table-size	INT	1024	RRL buffer minimum size
responses-per-second	INT	5	Allowed response rate
slip	INT	2	Random slip parameter
window	INT	15	RRL sliding window size in seconds

RRL SECTION

configuration example

```
<rrl>
    responses-per-second        5
    errors-per-second           5
    slip                        10
    log-only                    off
    ipv4-prefix-length          24
    ipv6-prefix-length          56
    exempt-clients              none
    enabled                     yes
</rrl>
```

13.3.9 <*dnssec-policy*> section

The dnssec-policy section binds key suites and a denial mode. It is meant to be used as a dnssec-policy parameter in a zone section. Usually two key-suite will be given: one for a *KSK*, and one for a *ZSK*.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the dnssec-policy section.
description	STR	-	Description for the dnssec-policy section.
key-suite	STR	-	**id** of the <*key-suite*> to be used. Usually both a *KSK*, and a *ZSK* suites are given.
denial	STR	nsec	**id** of the <*denial*> to be used for *NSEC3* or the argument 'nsec' to use *NSEC*.

DNSSEC-POLICY SECTION

configuration example with <*denial*>

```
<dnssec-policy>
    id                    "dnssec-policy-nsec3"

    description           "Example of ZSK and KSK"
    denial                "nsec3-resalting-on"
    key-suite             "zsk-1024"
    key-suite             "ksk-2048"
</dnssec-policy>
```

configuration example without <*denial*>

```
<dnssec-policy>
    id                  "dp-nsec"

    description         "Example of ZSK and KSK"
    denial              "nsec"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>
```

13.3.10 <*key-suite*> section

The key-suite section is used by dnssec policies and is meant to be referenced by a dnssec-policy section. A key-suite links a key definition (key-template) with a deployment calendar (key-roll).

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-suite section.
key-template	STR	-	**id** of the <*key-template*> to be used.
key-roll	STR	-	**id** of the <*key-roll*> to be used.

KEY-SUITE SECTION

configuration example <*key-suite*>

```
<key-suite>
    id                  "ksk-2048"

    key-template        "ksk-2048"
    key-roll            "yearly-schedule"
</key-suite>
```

13.3.11 <*key-roll*> section

The key-roll section is used by dnssec policies and is meant to be referenced by a key-suite section. It's essentially a deployment calendar. Each event is computed relatively to another. Dates are chosen so that there is always a key in an active state. Please look at the examples as a misconfiguration could easily span the life of a key over several years. If the RELDATE format is being used, the first valid date matching the line is used. (e.g.: by being too restrictive on the matching conditions) Usage of the RELDATE format is recommended over the RELTIME one.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-roll section.
generate; generated; create	RELTIME | RELDATE	-	Time when the key must be generated. Pre-dated before so it's active right now if it's the first one. Always computed so that the next activation happens before the last deactivation.
publish	RELTIME | RELDATE	-	Time when the key must be published in the zone. Relative to the generation.
activate	RELTIME | RELDATE	-	Time when the key will be used for signing the zone or apex of the zone. Relative to the publication.
inactive	RELTIME | RELDATE	-	Time when the key will not be used anymore for signing. Relative to the activation.
delete	RELTIME | RELDATE	-	Time when the key will be removed out of the zone. Relative to the deactivation.

KEY-ROLL SECTION

configuration example **section-key-roll**

```
<key-roll>
    id                 "yearly-schedule"
    # this year   (2018) 15/06 at 00:05
    generate     5      0     15      6      *      *
    #                             00:10
    publish     10      0     15      6      *      *
    #                    16/06 at 00:15
    activate    15      0     16      6      *      *
    #             (2019) 17/06 at 00:15
    inactive    15      0     17      6      *      *
    #             (2019) 18/06 at 11:15
    remove      15     11     18      6      *      *
</key-roll>
```

13.3.12 *<key-template>* section

The key-template section is used by dnssec policies and is meant to be referenced by a key-suite section. It contains the various parameters of a key for its generation.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-template section.
ksk	FLAG	false	When this flag is enabled a *KSK* will be generated. When disabled a *ZSK* will be generated.
algorithm	ENUM	7	Sets the algorithm of the key. Supported values are: ’DSA’; 3; ’RSASHA1’; 5; ’NSEC3DSA’; 6; ’NSEC3RSASHA1’; 7; ’RSASHA256’; 8; ’RSASHA512’; 10; ’ECDSAP256SHA256’; 13; ’ECDSAP384SHA384’; 14. ; ’ED25519’ ; 15 ; ’ED448’ ; 16
size	INT	0	The length of the key in bits (incompatible sizes will be rejected). (MIN: 0; MAX: 4096)

KEY-TEMPLATE SECTION

configuration example **section-key-template**

```
<key-template>
    id                  "ksk-2048"
    ksk                 true
    algorithm           8
    size                2048
    engine              default
</key-template>
```

13.3.13 <*denial*> section

The denial section is used by dnssec policies and is meant to be referenced by a dnssec-policy section. It is used to define the *NSEC3* denial parameters of a dnssec policy. Policies using a *NSEC* denial don’t need to use this section.

PARAMETER	TYPE	DEFAULT	DESCRIPTION

id	STR	-	**id** of the denial section.
salt	HEXSTR	-	The actual salt to use. Mutually exclusive with the **salt-length** option.
salt-length	INT	0	The system will generate a random salt with this length. Mutually exclusive with the **salt** option. (MIN: 0; MAX: 256)
iterations	INT	1	The number of iterations the salt and hash should be applied to the label. (MIN: 0; MAX: 65535)
optout	FLAG	false	When this flag is enabled only delegations which have a *DS* record will be considered for *NSEC3* record generation.

DENIAL SECTION

configuration example <*denial*>

```
<denial>
    id                  "nsec3-resalting-on"

    salt                "ABCD"
    #salt-length        4
    iterations          5
    optout              off
</denial>
```

14 Zones

Only textual zones are implemented.

The format of a zone file is defined in **RFC_1034**[43] and **RFC_1035**[44].

zone file sample

```
;; Example domain
$TTL    86400   ; 24 hours
$ORIGIN somedomain.eu.

somedomain.eu.      86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )

                    86400   IN  MX  10 mail.somedomain.eu.
                    86400   IN  NS  ns1.somedomain.eu.

ns1.somedomain.eu.  86400   IN  A   192.0.2.2
mail.somedomain.eu. 86400   IN  A   192.0.2.3
www.somedomain.eu.  86400   IN  A   192.0.2.4
```

14.1 Macros

Some macros are implemented:

- @
- $INCLUDE
- $ORIGIN
- $TTL

14.1.1 @

Use as a name, the @ symbol is replaced by the current origin.
The initial value is the **domain** field of the <zone> section.

For example:

configuration sample

```
<zone>
    domain somedomain.eu
    ...
</zone>
```

zone file sample

```
;; The following @ is seen as somedomain.eu.

@                   86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )
```

14.1.2 $INCLUDE

The value of this macro must be the name of an existing file. The contents of this file will be inserted into the zone at the place of the macro. It is allowed to have multiple **$INCLUDE** macros in the same zone file and/or use them in the included file.

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600
$ORIGIN somedomain.eu.
somedomain.eu.              86400   IN  SOA ns1 info (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )
ns1                        86400 A   192.0.2.2
$INCLUDE mailserver.zone
```

```
www                        86400 A   192.0.2.4
```

zone file sample

```
;; The file mailserver.zone
mail                       86400 A   192.0.2.3
```

Would be identical to a single zone file below:

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600
$ORIGIN somedomain.eu.
somedomain.eu.             86400   IN  SOA ns1 info (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )
ns1                        86400 A   192.0.2.2
;; The file mailserver.zone
mail                       86400 A   192.0.2.3
www                        86400 A   192.0.2.4
```

14.1.3 $ORIGIN

The value of this macro is appended to any following domain name not terminating with a “.”. The initial value is the **domain** field of the <zone> section.

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600
$ORIGIN somedomain.eu.
somedomain.eu.             86400   IN  SOA ns1 info (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )
ns1                        86400 A   192.0.2.2
mail                       86400 A   192.0.2.3
www                        86400 A   192.0.2.4
```

14.1.4 $TTL

This macro is the **TTL** value that is to be set for the resource records with an undefined **TTL**.

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600

somedomain.eu.        86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                     1
                                     3600
                                     1800s
                                     3600000s
                                     600
                                     )
ns1.somedomain.eu.    86400  A   192.0.2.2

mail.somedomain.eu.   86400  A   192.0.2.3

www.somedomain.eu.    86400  A   192.0.2.4
                             A   192.0.2.5

ftp.somedomain.eu.           A   192.0.2.6 ;; The TTL will be set using $TTL
```

14.2 Classes

YADIFA knows only one class:

- IN [44].

14.3 Resource record types

As primary name server, *YADIFA* knows only the following *RR* types. Everything else will give an error and be ignored.

TYPE	VALUE	REFERENCE	SUPPORTED

A	1	**RFC_1035**[44]	Y
NS	2	**RFC_1035**[44]	Y
MD	3	**RFC_1035**[44]	N
MF	4	**RFC_1035**[44]	N
CNAME	5	**RFC_2308**[?] **RFC_1035**[44]	Y
SOA	6	**RFC_1035**[44]	Y
MB	7	**RFC_1035**[44]	N
MG	8	**RFC_1035**[44]	N
MR	9	**RFC_1035**[44]	N
NULL	10	**RFC_1035**[44]	N
WKS	11	**RFC_1035**[44]	Y
PTR	12	**RFC_1035**[44]	Y
HINFO	13	**RFC_1035**[44]	Y
MINFO	14	**RFC_1035**[44]	N
MX	15	**RFC_1035**[44]	Y
TXT	16	**RFC_1035**[44]	Y
RP	17	**RFC_1183**[55]	N
AFSDB	18	**RFC_1183**[55] **RFC_5864**[5]	N
X25	19	**RFC_1183**[55]	N
ISDN	20	**RFC_1183**[55]	N
RT	21	**RFC_1183**[55]	N
NSAP	22	**RFC_1706**[15]	N
NSAP-PTR	23	**RFC_1348**[14] **RFC_1637**[16] **RFC_1706**[15]	N
SIG	24	**RFC_4034**[51] **RFC_3755**[60] **RFC_2535**[21] **RFC_2536**[22] **RFC_2537**[23] **RFC_2931**[1] **RFC_3110**[2] **RFC_3008**[61]	N
KEY	25	**RFC_4034**[51] **RFC_3755**[60] **RFC_2535**[21] **RFC_2536**[22] **RFC_2537**[23] **RFC_2539**[24] **RFC_3008**[61] **RFC_3110**[2]	N
PX	26	**RFC_2163**[6]	N
GPOS	27	**RFC_1712**[9]	N
AAAA	28	**RFC_3596**[54]	Y
LOC	29	**RFC_1876**[19]	N
NXT	30	**RFC_3755**[60] **RFC_2535**[21]	N
EID	31	DNS Resource Records for Nimrod Routing Architecture	N
NIMLOC	32	DNS Resource Records for Nimrod Routing Architecture	N
SRV	33	**RFC_2782**[25]	Y
ATMA	34	ATM Name System V2.0	N
NAPTR	35	**RFC_2915**[18] **RFC_2168**[42] **RFC_3403**[41]	Y

KX	36	**RFC_2230**[7]	N
CERT	37	**RFC_4398**[36]	N
A6	38	**RFC_3226**[27] **RFC_2874**[33] **RFC_6563**[13]	N
DNAME	39	**RFC_6672**[62]	N
SINK	40	The Kitchen Sink Resource Record	N
OPT	41	**RFC_6891**[57] **RFC_3225**[17]	N
APL	42	**RFC_3123**[38]	N
DS	43	**RFC_4034**[51] **RFC_3658**[28]	Y
SSHFP	44	**RFC_4255**[26]	Y
IPSECKEY	45	**RFC_4025**[49]	N
RRSIG	46	**RFC_4034**[51] **RFC_3755**[60]	Y
NSEC	47	**RFC_4034**[51] **RFC_3755**[60]	Y
DNSKEY	48	**RFC_4034**[51] **RFC_3755**[60]	Y
DHCID	49	**RFC_4701**[29]	N
NSEC3	50	**RFC_5155**[11]	Y
NSEC3PARAM	51	**RFC_5155**[11]	Y
TLSA	52	**RFC_6698**[53]	Y
HIP	55	**RFC_5205**[40]	N
NINFO	56	The Zone Status (ZS) DNS Resource Record	N
RKEY	57	ENUM Encryption	N
TALINK	58	talink-completed-template	N
CDS	59	**RFC_7344**[10]	N
CDNSKEY	60	**RFC_7344**[10]	N
OPENPGPKEY	61	Using DANE to Associate OpenPGP public keys with email addresses	N
CSYNC	62	**RFC_7477**[32]	N
SPF	99	**RFC_7208**[37]	N
UINFO	100	[IANA-Reserved]	N
UID	101	[IANA-Reserved]	N
GID	102	[IANA-Reserved]	N
UNSPEC	103	[IANA-Reserved]	N
NID	104	**RFC_6742**[52]	N
L32	105	**RFC_6742**[52]	N
L64	106	**RFC_6742**[52]	N
LP	107	**RFC_6742**[52]	N
EUI48	108	**RFC_7043**[3]	N
EUI64	109	**RFC_7043**[3]	N
DLV	32769	**RFC_4431**[59]	N

SUPPORTED TYPES

15 Journal

YADIFA has got an updated journaling system since the release of version 2.4.0.

Before *YADIFA* 2.1.0:

- Is based on a append-only file
- Has a linear access time (except the last few entries) which was not ideal for random access on big journals
- Could only be limited in growth by emptying it completely.

Before *YADIFA* 2.4.0:

- Is based on a file that is being written in a cyclic fashion
- Has a relatively constant access time
- Can be limited in size, although it is not a hard limit.

From *YADIFA* 2.4.0, the index table appendix is dropped. The manually-set journal size is a hard limit. The first time *YADIFA* 2.4.0 finds a pre-2.4.0 .cjf journal file, it plays it, stores the resulting zone on disk and deletes the file so the updated *.cjf* version can be used.

The journal size is automatically set by *YADIFA* at around half the size of the zone size, but it can be set to an arbitrary value through configuration. To do this, one merely needs to set journal-size-kb in the <*zone*> section of the zone. The value range for version 2.4.0 is from 64 kB to 3 GB. It is recommended to set it to a multiple of 64.

configuration example

```
<zone>
  domain somedomain.eu
    ...
    journal-size-kb 64
</zone>
<zone>
    domain someotherdomain.eu
    ...
    journal-size-kb 256000
</zone>
```

In order to reduce the size of the journal after reconfiguring it, it is recommended that one uses the command line to synchronize the zone and wipe the journal empty.

16 Statistics

YADIFA has a range of statistics available with one configuration setting. The statistics logger values are grouped into inputs, outputs and the RRL. Groups are composed of a name followed by an open parenthesis containing several space-separated event=count fields and ending in a closed parenthesis.

A single line of statistics looks as follows:

shell output

```
udp (in=303 qr=303 ni=0 up=0 dr=0 st=91191 un=0 rf=0)
    ↪ tcp (in=369 qr=368 ni=0 up=0 dr=0 st=82477 un=0 rf=0 ax=0 ix=0 ov=0)
    ↪ udpa (OK=242 FE=0 SF=0 NE=0 NI=0 RE=61 XD=0 XR=0 NR=0 NA=0 NZ=0 BV=0 BS=0 BK=0 BT=0BM
    ↪ =0 BN=0 BA=0 TR=0)
    ↪ tcpa (OK=209 FE=0 SF=0 NE=0 NI=0 RE=159 XD=0 XR=0 NR=0 NA=0 NZ=0 BV=0 BS=0 BK=0 BT=0
    ↪ BM=0 BN=0 BA=0 TR=0) rrl (sl=0 dr=0)
```

You can clearly see the groups containing the event=count fields. There are currently 5 groups defined:

- udp(. . .) covers the *UDP* messages
- udpa(. . .) covers the *UDP* messages answers
- tcp(. . .) covers the *TCP* messages
- tcpa(. . .) covers the *TCP* messages answers
- rrl(. . .) covers the RRL events

The statistics logger counts the various events about the messages from the clients.

in input count
    counts the number of *DNS* messages received

qr query count
    counts the number of queries among the *DNS* messages

ni notify count
counts the number of notifications among the *DNS* messages

up update count
counts the number of updates among the *DNS* messages

dr dropped count
counts the number of *DNS* messages dropped

st total bytes sent (simple queries only)
counts the total number of bytes sent

un undefined opcode count
counts the number of undefined opcodes among the *DNS* messages

rf referral count
counts the number of referrals among the *DNS* queries

ax *AXFR* query count (*TCP* only)
counts the number of full zone transfers queried

ix *IXFR* query count (*TCP* only)
counts the number of incremental zone transfers queried

ov connection overflow (*TCP* only)
counts the number of times the *TCP* pool has been full when a new connection came in

The statistics logger answers counts the status of *DNS* answers sent to the clients.

OK NOERROR answer count

FE FORMERR answer count

SF SERVFAIL answer count

NE NXDOMAIN answer count

NI NOTIMP answer count

RE REFUSED answer count

XD YXDOMAIN answer count

XR YXRRSET answer count

NR NXRRSET answer count

NA NOTAUTH answer count

NZ NOTZONE answer count

BV BADVERS answer count

BS BADSIG answer count

BK BADKEY answer count

BT BADTIME answer count

BM BADMODE answer count

BN BADNAME answer count

BA BADALG answer count

TR BADTRUNC answer count

The RRL group only counts the two main events of the Response Rate Limiter.

dr dropped answer count
counts the number of times an answer has been dropped

sl truncated answer count
counts the number of times an answer that should have been dropped has been sent truncated instead

17 Configuration Examples

17.1 Introduction

```
;; Example domain
$TTL    86400   ; 24 hours
$ORIGIN somedomain.eu.

somedomain.eu.      86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )

                    86400   IN  MX  10 mail.somedomain.eu.
                    86400   IN  NS  ns1.somedomain.eu.

ns1.somedomain.eu.  86400   IN  A   192.0.2.2
mail.somedomain.eu. 86400   IN  A   192.0.2.3
www.somedomain.eu.  86400   IN  A   192.0.2.4
```

17.2 YADIFA as a primary name server

17.2.1 The One That is Really Easy

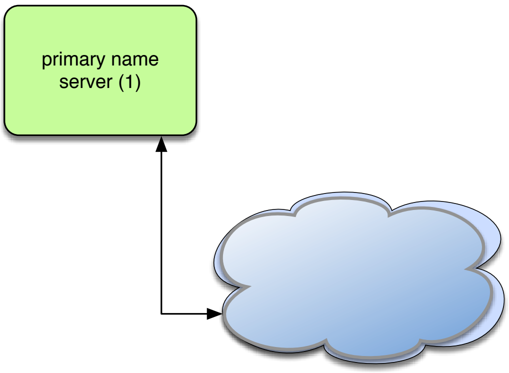

Figure 17.1: Primary name server (simple configuration)

configuration example of That is Really Easy

```
<zone>
    domain                  somedomain.eu
    file                    "primaries/somedomain.eu."
    type                    "primary"
</zone>
```

17.2.2 The One With Activation of Logging

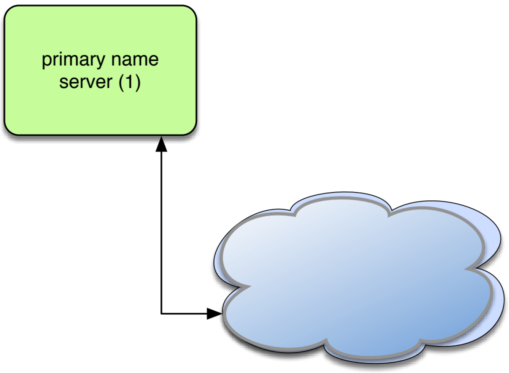

Figure 17.2: Primary name server with logging

configuration example of Activation of Logging

```
<channels>
    # user-defined-name     parameters
    # channel 'statistics': a file called stats.log
    #                       with 0644 access rights
    #
    statistics              stats.log 0644

    # channel 'syslog'    : a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa'    : a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>
<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>
<zone>
```

```
    domain              somedomain.eu
    file                "primaries/somedomain.eu."
    type                "primary"
</zone>
```

17.2.3 The One With NSID

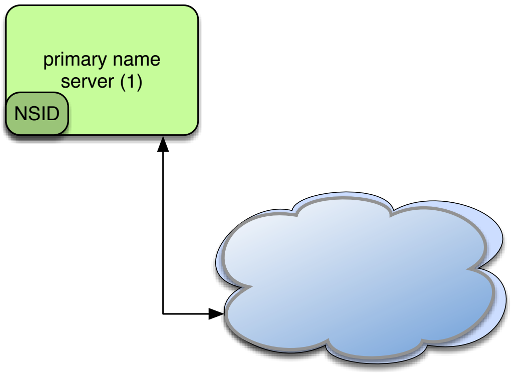

Figure 17.3: Primary name server with NSID

configuration example of NSID

```
<nsid>
    ascii "yadifad example NSID"
    # alternatively, an hexadecimal format can be used
    # hex 796164696661642065786616d706c65204e5349440a
</nsid>

<zone>
    domain                  somedomain.eu
    file                    "primaries/somedomain.eu."
    type                    "primary"
</zone>
```

17.2.4 The One With RRL

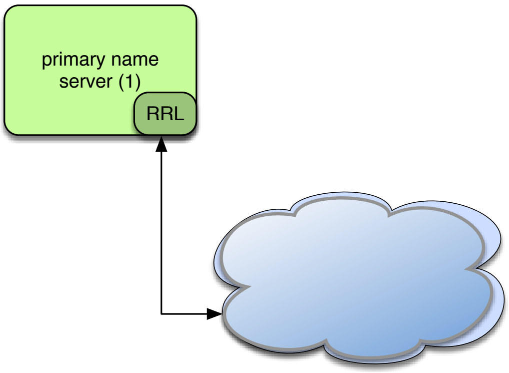

Figure 17.4: Primary name server with RRL

configuration example of RRL

```
# If YADIFA has been compiled with the Response Rate Limiter (default)
<rrl>
    # enable the RRL
    enabled             true

    # don't actually limit the response rate, only log what the filter
    # would do
    log-only            false

    # how many responses per second are allowed for a client
    # (masked with the prefix)
    responses-per-second 5

    # how many errors per second are allowed for a client
    # (masked with the prefix)
    errors-per-second   5

    # window of time in which the rates are measured, expressed in seconds
    window              15

    # every "slip" dropped answers, a truncated answer may randomly be
    # given so the client can ask again using TCP
    slip                2

    # the min size of the table storing clients(masked with the prefix)
    min-table-size      1024

    # the max size of the table storing clients(masked with the prefix)
    max-table-size      16384

    # IPv4 clients are masked with this prefix
    ipv4-prefix-length  24

    # IPv6 clients are masked with this prefix
    ipv6-prefix-length  56
    # the list of IP/networks (Access Control List) not impacted by
    # the RRL
    exempted            none
</rrl>

<zone>
    domain              somedomain.eu
    file                "primaries/somedomain.eu."
    type                "primary"
</zone>
```

17.2.5 The One With ACL

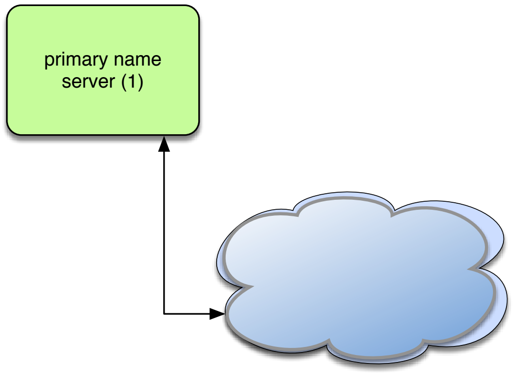

Figure 17.5: Primary name server with ACL

configuration example of ACL

```
<key>
    name                primary-secondary
    algorithm           hmac-md5
    secret              PrimaryAndSecondaryKey==
</key>

<key>
    name                update-key
    algorithm           hmac-md5
    secret              UpdateKeyVerySecretKey==
</key>

<acl>
    transferer          key primary-secondary
    updaterkey          key update-key
    # 192.0.2.72-192.0.2.75 and 2001:db8::74/128
    updaters            192.0.2.72/30, 2001:db8::74
    # default /32 for IPv4
    secondary           192.0.2.2
</acl>

<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "primary"
    # send notifies even if not in the NS-set
    notify-also         192.0.2.2
    # transfers allowed by IP from secondary (unless an invalid key is used)
    # transfers allowed by key from transferer (from any IP address)
    allow-transfer      secondary, transferer
    # allow updates by IP from updaters (unless an invalid key is used)
    # allow updates using the update-key (from any IP address)
    allow-update        updaters, updatekey
    # only secondary is allowed to query the primary name server
    allow-query         secondary
</zone>
```

17.2.6 The One With DNSSEC Policy ‘diary’ style

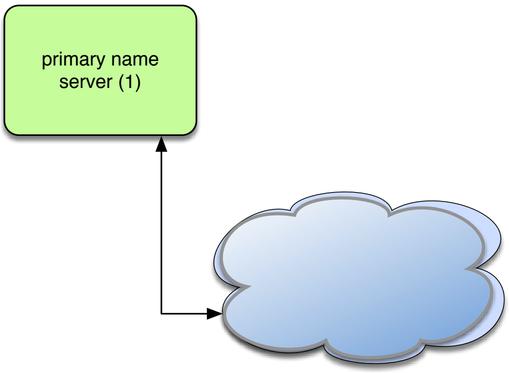

Figure 17.6: Primary name server (DNSSEC policy ‘diary’ style)

configuration example of DNSSEC policy ’diary’ style

```
<key-roll>
    id                 "yearly-schedule"

    generate            5          0           15                  6            *
    ↪       * # this year   (2018) 15/06 at 00:05
    publish             10         0           15                  6            *
    ↪       * #                             00:10
    activate            15         0           16                  6            *
    ↪       * #                    16/06 at 00:15
    inactive            15         0           17                  6            *
    ↪       * #             (2019) 17/06 at 00:15
    remove              15        11           18                  6            *
    ↪       * #             (2019) 18/06 at 11:15
</key-roll>

<key-roll>
    id                 "monthly-schedule"

    generate            5          0           *                   *            tue
    ↪       0 # 1 tuesday   of the month at 00:05
    publish             10         0           *                   *            tue
    ↪       0 #                             00:10
    activate            15         0           *                   *            wed
    ↪       0 # 1 wednesday of the month at 00:15
    inactive            15         0           *                   *            thu
    ↪       0 # 1 thursday  of the month at 00:15
    remove              15        11           *                   *            fri
    ↪       0 # 1 friday    of the month at 11:15
</key-roll>

<key-suite>
    id                  "ksk-2048"
    key-template        "ksk-2048"
    key-roll            "yearly-schedule"
</key-suite>
```

```
<key-suite>
    id                  "zsk-1024"
    key-template        "zsk-1024"
    key-roll            "monthly-schedule"
</key-suite>

<dnssec-policy>
    id                  "dp-nsec"

    description         "Example of ZSK and KSK"
    denial              "nsec"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>

<zone>
    domain              somedomain.eu
    file                primaries/somedomain.eu.
    type                "primary"
    dnssec-policy       "dp-nsec"
</zone>
```

17.2.7 The One With DNSSEC Policy ‘relative’ style

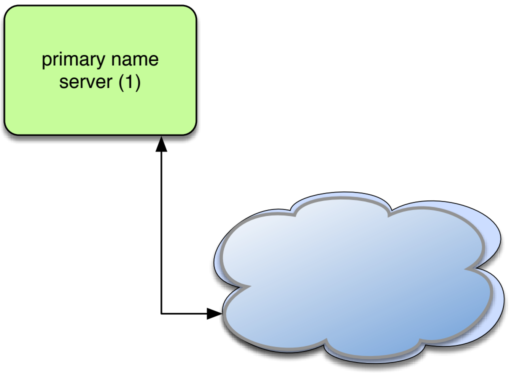

Figure 17.7: Primary name server (DNSSEC policy ‘relative’ style)

configuration example of DNSSEC policy ’relative’ style

```
<key-roll>
    id                  "yearly-schedule"

    create              +355d
    publish             +4h
    activate            +10d
    inactive            +366d
    delete              +7d
</key-roll>

<key-roll>
    id                  "monthly-schedule"

    create              +30d
    publish             +2h
    activate            +7200 # 2 hours (in seconds)
    inactive            +31d
    delete              +7d
</key-roll>

<key-suite>
    id                  "ksk-2048"
    key-template        "ksk-2048"
    key-roll            "yearly-schedule"
</key-suite>

<key-template>
    id                  "ksk-2048"
    ksk                 true
    algorithm           8
    size                2048
    engine              default
</key-template>

<key-suite>
    id                  "zsk-1024"
    key-template        "zsk-1024"
    key-roll            "monthly-schedule"
</key-suite>

<key-template>
    id                  "zsk-1024"
    algorithm           8
    size                1024
    engine              default
</key-template>

<denial>
    id                  "nsec3-resalting-on"

    salt                "ABCD"
    #salt-length        4
    iterations          5
    optout              off
</denial>

<dnssec-policy>
    id                  "dnssec-policy-nsec3"

    description         "Example of ZSK and KSK"
    denial              "nsec3-resalting-on"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>

<zone>
    domain              somedomain.eu
    file                primaries/somedomain.eu.
    type                "primary"
    dnssec-policy       "dp-nsec"
</zone>
```

17.2.8 The One With RRSIG Update Allowed

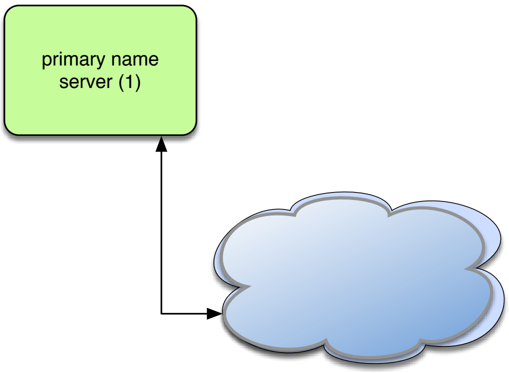

Figure 17.8: Primary name server (RRSIG Update Allowed)

configuration example of RRSIG Update Allowed

```
<key-roll>
    id                  "yearly-schedule"

    create              +355d
    publish             +4h
    activate            +10d
    inactive            +366d
    delete              +7d
</key-roll>

<key-roll>
    id                  "monthly-schedule"

    create              +30d
    publish             +2h
    activate            +7200 # 2 hours (in seconds)
    inactive            +31d
    delete              +7d
</key-roll>

<key-suite>
    id                  "ksk-2048"
    key-template        "ksk-2048"
    key-roll            "yearly-schedule"
</key-suite>

<key-template>
    id                  "ksk-2048"
    ksk                 true
    algorithm           8
    size                2048
    engine              default
</key-template>

<key-suite>
    id                  "zsk-1024"
    key-template        "zsk-1024"
    key-roll            "monthly-schedule"
</key-suite>

<key-template>
    id                  "zsk-1024"
    algorithm           8
    size                1024
    engine              default
</key-template>

<denial>
    id                  "nsec3-resalting-on"

    salt                "ABCD"
    #salt-length        4
    iterations          5
    optout              off
</denial>

<dnssec-policy>
    id                  "dnssec-policy-nsec3"

    description         "Example of ZSK and KSK"
    denial              "nsec3-resalting-on"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>

<zone>
    domain                  somedomain.eu
    file                    primaries/somedomain.eu.
    type                    "primary"
    dnssec-policy           "dp-nsec"
    rrsig-nsupdate-allowed  true
</zone>
```

17.2.9 The One With the Controller

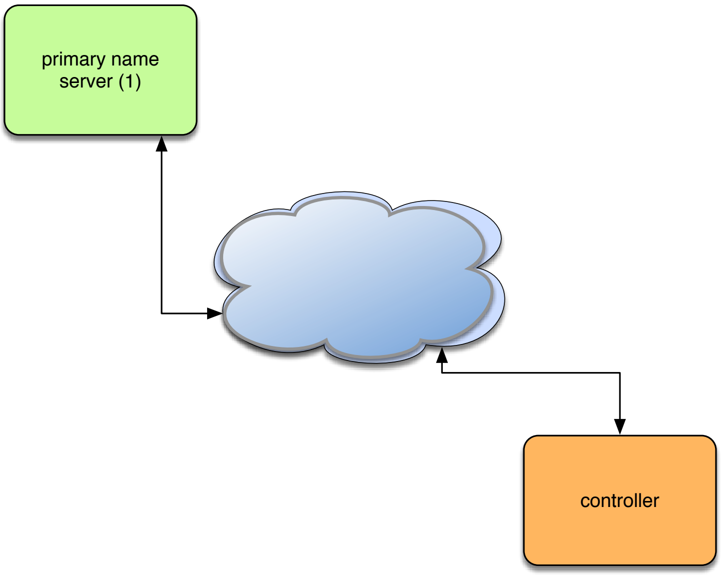

Figure 17.9: Primary name server with controller

On the primary name server (${SYSCONFDIR}/yadifad.conf):

configuration example of controller (server)

```
<main>
    allow-control       "yadifa-controller"
</main>

<acl>
    yadifa-controller   key "controller-key"
</acl>

<key>
    name                "controller-key"
    algorithm           "hmac-md5"
    secret              "ControlDaemonKey"
</key>
```

On the controller (${HOME}/.yadifa.rc or ${SYSCONFDIR}/yadifa.conf):

configuration example of controller (client)

```
<yadifa-ctrl>
    server              192.0.2.1
    tsig-key-name       "controller-key"
</yadifa-ctrl>

<key>
    name                "controller-key"
    algorithm           "hmac-md5"
    secret              "ControlDaemonKey"
</key>
```

17.3 YADIFA as a secondary name server

17.3.1 The One With One Primary

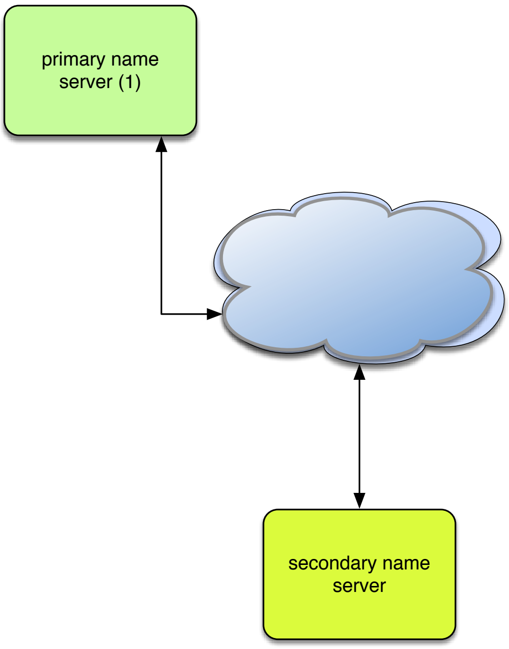

Figure 17.10: Secondary name server (one primary)

configuration example of One Primary

```
<zone>
    domain                  somedomain.eu
    file                    "secondaries/somedomain.eu."
    type                    "secondary"
    primary                 192.0.2.1
</zone>
```

17.3.2 The One With Several Primaries

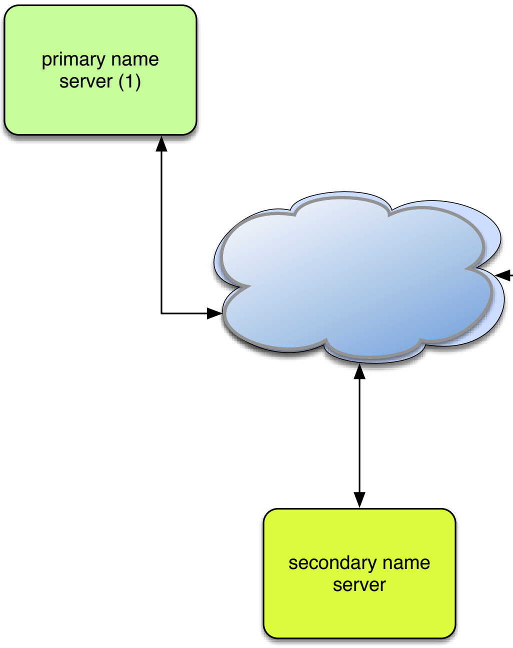

Figure 17.11: Secondary name server (several primaries)

configuration example of Several Primaries

```
<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "secondary"

    primaries           192.0.2.1,192.0.2.2
    true-multiprimary   yes
</zone>
```

17.3.3 The One With Activation of Logging

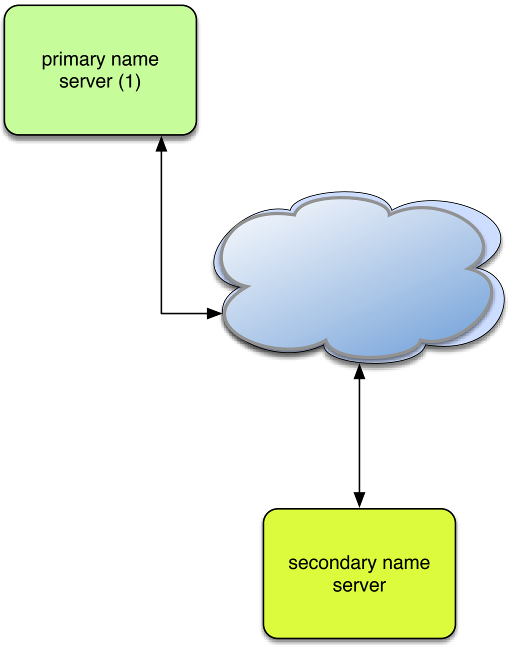

Figure 17.12: Secondary name server with logging

configuration example of Activation of Logging

```
<channels>
    # user-defined-name     parameters
    # channel 'statistics': a file called stats.log
    #                       with 0644 access rights
    #
    statistics              stats.log 0644

    # channel 'syslog'    : a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa'    : a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>

<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>

<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "secondary"
    primary             192.0.2.1
</zone>
```

17.3.4 The One With NSID

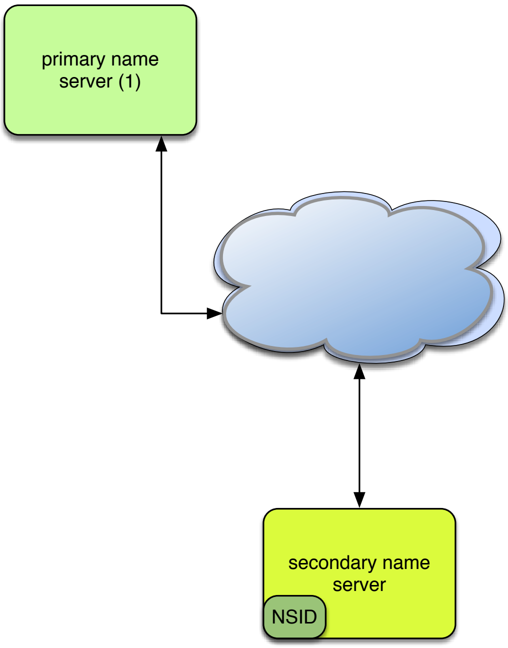

Figure 17.13: Secondary name server with NSID

configuration example of NSID

```
<nsid>
    ascii "yadifad example NSID"
    # alternatively, an hexadecimal format can be used
    # hex 79616469666164206578616d706c65204e5349440a
</nsid>

<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "secondary"
    primary             192.0.2.1
</zone>
```

17.3.5 The One With RRL

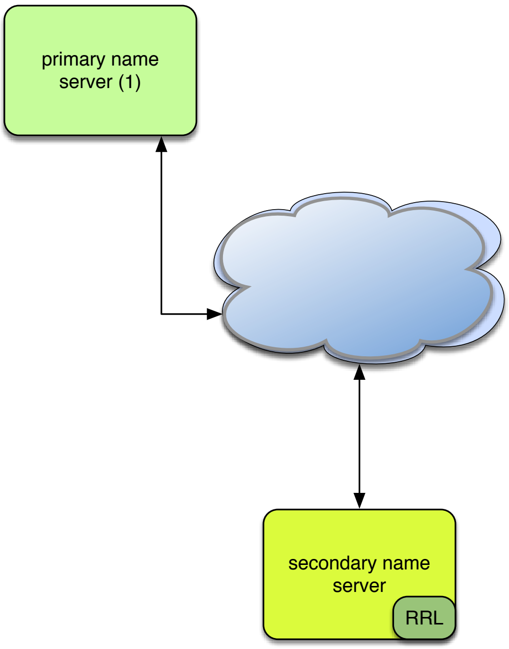

Figure 17.14: Secondary name server with RRL

configuration example of RRL

```
# If YADIFA has been compiled with the Response Rate Limiter (default)
<rrl>
    # enable the RRL
    enabled             true

    # don't actually limit the response rate, only log what the filter
    # would do
    log-only            false

    # how many responses per second are allowed for a client
    # (masked with the prefix)
    responses-per-second 5

    # how many errors per second are allowed for a client
    # (masked with the prefix)
    errors-per-second   5

    # window of time in which the rates are measured, expressed in seconds
    window              15

    # every "slip" dropped answers, a truncated answer may randomly be
    # given so the client can ask again using TCP
    slip                2

    # the min size of the table storing clients(masked with the prefix)
    min-table-size      1024
    # the max size of the table storing clients(masked with the prefix)
    max-table-size      16384

    # IPv4 clients are masked with this prefix
    ipv4-prefix-length  24

    # IPv6 clients are masked with this prefix
    ipv6-prefix-length  56

    # the list of IP/networks (Access Control List) not impacted by
    # the RRL
    exempted            none
</rrl>

<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "secondary"
    primary             192.0.2.1
</zone>
```

17.3.6 The One With ACL

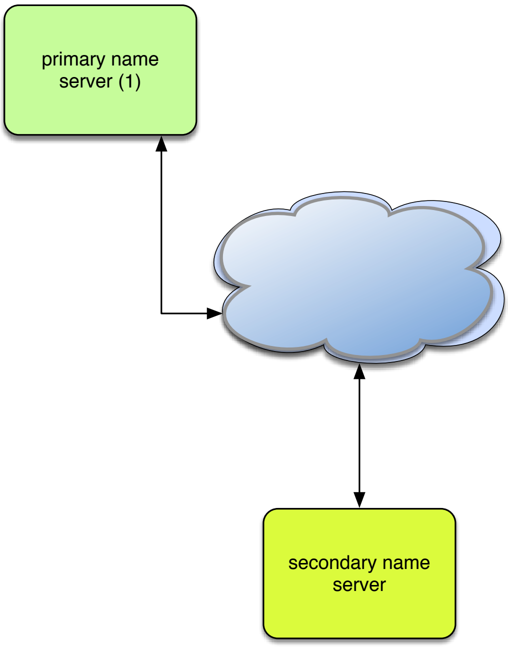

Figure 17.15: Secondary name server with ACL

configuration example of ACL

```
<key>
    name                primary-secondary
    algorithm           hmac-md5
    secret              PrimaryAndSecondaryKey==
</key>

<acl>
    # default /32 for IPv4
    primary             192.0.2.1
</acl>

<zone>
    domain              somedomain.eu
    file                "secondaries/somedomain.eu."
    type                "secondary"
    primary             192.0.2.1 key primary-secondary
    # allow notifies coming from the primary only
    allow-notify        primary
    # do not allow transfers
    allow-transfer      none
    # do now allow updates
    allow-update        none
    # only secondary is allowed to query the primary name server
    allow-query         any
</zone>
```

18 Troubleshooting

By default, *YADIFA* logs everything on the standard output. Warnings or errors may point to the issue. When configuring the logging to suit your needs, it is recommended one keeps the levels: warning,err,crit,alert and emerg for everything but the queries.

18.1 Submitting a bug report

If you are unable to fix the issue yourself, you can submit a bug report to the *YADIFA* team.
For critical issues (i.e.: crash), please use bugreport@yadifa.eu.
For any other issue or question, you can use yadifa-users@mailinglists.yadifa.eu.

The report should contain:

- The operating system type and version
- The version of *YADIFA* and how it was installed.
  - If you configured it yourself : the *./configure* parameters
  - If you used a package : where from and what version
- What machine it is running on
- All the log output, preferrably with all levels enabled (* or any in the configuration file).
- If you know them: the steps to reproduce the issue
- If possible, the zone files and as much of the configuration file you can give (i.e.: everything but the *TSIG* keys)

Please find enclosed two short scripts you can run on the server to retrieve most of the information we need.

System information (some programs or files will not exist on your system):

script

```
#!/bin/sh

# basic system information
echo uname:
echo ------
uname -a
# OS
cat /etc/lsb_release
cat /etc/redhat-release
cat /etc/slackware-version
cat /etc/os-release
cat /etc/defaults/pcbsd
cat /etc/defaults/trueos
echo mount:
echo ------
mount
# available disk space
echo df:
echo ---
df -h

# available memory space
echo free:
echo -----
free -h
```

Hardware information:

script

```
#!/bin/sh
# various hardware information

echo lscpu:
echo ------
lscpu

echo lspci:
echo ------
lspci

echo lshw:
echo -----
lshw

echo hwinfo:
echo -------
hwinfo

echo lsscsi:
echo -------
lsscsi

echo lsusb:
echo ------
lsusb

echo lsblk:
echo ------
lsblk

echo pciconf:
echo --------
pciconf -lvcb
```

Please find enclosed a short script you can run on the build machine to retrieve information about the compiler:

script

```
#/bin/sh

# compiler info (if you compiled yadifad yourself)
# to run on the build machine

echo gcc:
echo ----
gcc -v -v
gcc -dM -E - < /dev/null

echo clang:
echo ------
clang -v -v
clang -dM -E - < /dev/null
```

18.2 Stacktrace

In the case of a crash, generating a stacktrace at the time of the problem arises may help to understand the issue. Please note that it is best to do this with the debug symbols for the package installed or with a binary that has not been stripped.

To generate the stacktrace, you can either use a generated core dump, or run yadifad in the debugger.

Please note that the way to enable unlimited-size core dumps varies with your OS flavor. On some linux, you can get its location by executing:

shell

```
$> cat /proc/sys/kernel/core_pattern
```

And enable it typing, as root:

shell

```
$> ulimit -c unlimited
```

Be sure the command worked:

shell

```
$> ulimit -c
```

Should print:

shell output

```
unlimited
```

18.2.1 Using a core dump

With a core dump at hand, you can start the debugger like this:

gdb */path-to-yadifad/yadifad /path-to-yadifad-core-dump/yadifad-core-dump-file*

For example:

shell

```
$> gdb /usr/local/sbin/yadifad /var/cache/abrt/yadifad.core
```

Then on the debugger prompt:

gdb

```
set logging file /tmp/yadifad-stacktrace.txt
set logging on
thread apply all bt
```

You can keep pressing the [enter] key until you are back to an empty (gdb) prompt

gdb

```
quit
```

The file /tmp/yadifad-stacktrace.txt will contain the stacktraces.

18.2.2 Running *yadifad* in the debugger

You can start the debugger like this:

gdb */path-to-yadifad/yadifad*

shell

```
$> gdb /usr/local/sbin/yadifad
```

Or, if *yadifad* is already running, like this:

1. search for pid of yadifad (e.g.: 12345)
2. gdp -p 12345

shell

```
$> gdb -p 12345
```

Then on the debugger prompt:

gdb

```
handle SIGUSR1 noprint pass
handle SIGUSR2 noprint pass
handle SIGTERM noprint pass
handle SIGINT noprint pass
handle SIGPIPE noprint pass
handle SIGHUP noprint pass
handle SIG33 noprint pass
set follow-fork-mode child
run
```

When the debugger stops with an error (i.e.: SIGSEGV, SIGABRT):

gdb

```
set logging file /tmp/yadifad-stacktrace.txt
set logging on
thread apply all bt
```

You can keep pressing the [enter] key until you get an empty (gdb) prompt.

gdb

```
quit
```

The file /tmp/yadifad-stacktrace.txt will contain the stacktraces.

18.3 Building yadifad with even more debugging information

When preparing to build *yadifad*, there are *./configure* options that increase the debugging information available.

The stacktrace information in the logs can be improved using *–enable-bfd-debug*. The cost of this option can be considered negligible.
Please note that although very useful in some cases, the mutexes monitoring feature (enabled using *–enable-mutex-debug*) is extremely expensive and should only be used in very specific cases.

In order to enable more debugging information, the make target “debug” greatly increases logging and activates many runtime checks. All internal libraries must be compiled with the same target so start from a clean source.

shell

```
$> make clean
$> make debug
$> sudo make install
```

Note that this kind of build may generate extremely huge log files. The increased logging is still subject to the settings in *yadifad.conf*, so it is still possible to tune the flow.

Bibliography

[1] D. Eastlake 3rd. *DNS Request and Transaction Signatures (SIG(0)s)*, September 2000. **RFC 2931**.

[2] D. Eastlake 3rd. *RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)*, May 2001. **RFC 3110**.

[3] J. Abley. *Resource Records for EUI-48 and EUI-64 Addresses in the DNS*, October 2013. **RFC 7043**.

[4] J. Abley. *Authenticated Denial of Existence in the DNS*, February 2014. **RFC 7129**.

[5] R. Allbery. *DNS SRV Resource Records for AFS*, April 2010. **RFC 5864**.

[6] C. Allocchio. *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)*, January 1998. **RFC 2163**.

[7] R. Atkinson. *Key Exchange Delegation Record for the DNS*, November 1997. **RFC 2230**.

[8] R. Austein. *DNS Name Server Identifier (NSID) Option*, August 2007. **RFC 5001**.

[9] C. Farrell / M. Schulze / S. Pleitner / D. Baldoni. *DNS Encoding of Geographical Location*, November 1994. **RFC 1712**.

[10] W. Kumari / O. Gudmundsson / G. Barwood. *Automating DNSSEC Delegation Trust Maintenance*, September 2014. **RFC 7344**.

[11] B. Laurie / G. Sisson / R. Arends / D. Blacka. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, March 2008. **RFC 5155**.

[12] Paul Vixie / S. Thomson / Y. Rekhter / J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*, April 1997. **RFC 2136**.

[13] S. Jiang / D. Conrad / B. Carpenter. *Moving A6 to Historic Status*, March 2012. **RFC 6563**.

[14] B. Manning / R. Colella. *DNS NSAP Resource Records*, July 1992. **RFC 1348**.

[15] B. Manning / R. Colella. *DNS NSAP Resource Records*, October 1994. **RFC 1706**.

[16] B. Manning / R. Colella. *DNS NSAP Resource Records*, June 1994. **RFC 1637**.

[17] D. Conrad. *Indicating Resolver Support of DNSSEC*, December 2001. **RFC 3225**.

[18] M. Mealling / R. Daniel. *The Naming Authority Pointer (NAPTR) DNS Resource Record*, September 2000. **RFC 2915**.

[19] C. Davis / Paul Vixie / T. Goodwin / I. Dickinson. *Common DNS Operational and Configuration Errors*, January 1996. **RFC 1912**.

[20] Ed. E. Lewis / A. Hoenes. *DNS Zone Transfer Protocol (AXFR)*, June 2010. **RFC 5936**.

[21] D. Eastlake. *Domain Name System Security Extensions*, March 1999. **RFC 2535**.

[22] D. EastLake. *DSA KEYs and SIGs in the Domain Name System (DNS)*, March 1999. **RFC 2536**.

[23] D. Eastlake. *RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)*, March 1999. **RFC 2537**.

[24] D. Eastlake. *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)*, March 1999. **RFC 2539**.

[25] A. Gulbrandsen / Paul Vixie / L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*, February 2000. **RFC 2782**.

[26] J. Schlyter / W. Griffin. *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints*, January 2006. **RFC 4255**.

[27] O. Gudmundsson. *DNSSEC and IPv6 A6 aware server/resolver message size requirements*, December 2001. **RFC 3226**.

[28] O. Gudmundsson. *Delegation Signer (DS) Resource Record (RR)*, December 2003. **RFC 3658**.

[29] M. Stapp / T. Lemon / A. Gustafsson. *DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR)*, October 2006. **RFC 4701**.

[30] S. Kwan / P. Garg / J. Gilroy / L. Esibov / J. Westhead / R. Hall. *Secret Key Transaction Authentication for DNS (GSS-TSIG)*, October 2003. **RFC 3645**.

[31] T. Hansen. *US Secure Hash Algorithms*, May 2011. **RFC 6234**.

[32] W. Hardaker. *Child-to-Parent Synchronization in DNS*, March 2015. **RFC 7477**.

[33] M. Crawford / C. Huitema. *DNS Extensions to Support IPv6 Address Aggregation and Renumbering*, July 2000. **RFC 2874**.

[34] J. Jansen. *Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*, October 2009. **RFC 5702**.

[35] S. Josefsson. *Base-N Encodings*, October 2006. **RFC 4648**.

[36] S. Josefsson. *Storing Certificates in the Domain Name System (DNS)*, March 2006. **RFC 4398**.

[37] S. Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*, April 2014. **RFC 7208**.

[38] P. Koch. *A DNS RR Type for Lists of Address Prefixes (APL RR)*, June 2001. **RFC 3123**.

[39] A. Freier / P. Karlton / P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*, August 2011. **RFC 6101**.

[40] P. Nikander / J. Laganier. *Host Identity Protocol (HIP) Domain Name System (DNS) Extension*, April 2008. **RFC 5205**.

[41] M. Mealling. *Dynamic Delegation Discovery System (DDDS)*, October 2002. **RFC 3403**.

[42] R. Daniel / M. Mealling. *Resolution of Uniform Resource Identifiers using the Domain Name System*, June 1997. **RFC 2168**.

[43] P. Mockapetris. *DOMAIN NAMES - CONCEPTS AND FACILITIES*, November 1987. **RFC 1034**.

[44] P. Mockapetris. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*, November 1987. **RFC 1035**.

[45] M. Ohta. *Incremental Zone Transfer in DNS*, August 1996. **RFC 1995**.

[46] John Postel. *USER DATAGRAM PROTOCOL*, August 1980. **RFC 768**.

[47] John Postel. *INTERNET PROTOCOL*, September 1981. **RFC 791**.

[48] John Postel. *TRANSMISSION CONTROL PROTOCOL*, September 1981. **RFC 793**.

[49] M. Richardson. *A Method for Storing IPsec Keying Material in DNS*, February 2005. **RFC 4025**.

[50] R. Arends / R. Austein / M. Larson / D. Massey / S. Rose. *DNS Security Introduction and Requirements*, March 2005. **RFC 4033**.

[51] R. Arends / R. Austein / M. Larson / D. Massey / S. Rose. *Resource Records for the DNS Security Extensions*, March 2005. **RFC 4034**.

[52] RJ. Atkinson / SN. Bhatti / S. Rose. *DNS Resource Records for the Identifier-Locator Network Protocol (ILNP)*, November 2012. **RFC 6742**.

[53] P. Hoffman / J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, August 2012. **RFC 6698**.

[54] S. Thomson / C. Huitema / V. Ksinant / M. Souissi. *DNS Extensions to Support IP Version 6*, October 2003. **RFC 3596**.

[55] C. Everhart / L. Mamakos / R. Ullmann. *New DNS RR Definitions*, October 1990. **RFC 1183**.

[56] O. Vaughan. *A Legal basis for Domain Name Allocation*, November 1997. **RFC 2240**.

[57] J. Damas / M. Graff / Paul Vixie. *Extension Mechanisms for DNS (EDNS(0))*, April 2013. **RFC 6891**.

[58] Paul Vixie. *Extension Mechanisms for DNS (EDNS0)*, August 1999. **RFC 2671**.

[59] M. Andrews / S. Weiler. *The DNSSEC Lookaside Validation (DLV) DNS Resource Record*, February 2006. **RFC 4431**.

[60] S. Weiler. *Legacy Resolver Compatibility for Delegation Signer (DS)*, May 2004. **RFC 3755**.

[61] B. Wellington. *Domain Name System Security (DNSSEC) Signing Authority*, November 2000. **RFC 3008**.

[62] S. Rose / W. Wijngaards. *DNAME Redirection in the DNS*, June 2012. **RFC 6672**.

Index

algorithms
  rsasha1, 49
  rsasha256, 49, 55
  rsasha512, 49
ascii, 61
AXFR, 7, 25, 67, 72, 73, 80–83, 86, 88, 113

command
  bin
    ./configure, 139
    configure, 14, 27
    kill, 24
    make, 14, 27
    make install, 14, 27
    yadifa, 7, 13, 26, 27
  sbin
    yadifad, 7, 13, 26, 27, 40, 41, 43, 81, 144, 145
    yakeyrolld, 7, 13, 38–44
configuration
  activate, 101
  algorithm, 89, 102
  allow-control, 80, 86
  allow-notify, 80, 86
  allow-query, 80, 86
  allow-transfer, 80, 86
  allow-update, 80, 86
  allow-update-forwarding, 80, 86
  also-notify, 87
  answer-formerr-packets, 80
  axfr-compress-packets, 80
  axfr-max-packet-size, 80
  axfr-max-record-by-packet, 80
  axfr-memory-threshold, 80
  axfr-retry-delay, 80
  axfr-retry-failure-delay-max, 81
  axfr-retry-failure-delay-multiplier, 80
  axfr-retry-jitter, 80
  chroot, 81
  chroot-path, 81
  cpu-count-override, 81
  create, 101
  daemon, 81
  data-path, 81
  database, 94
  delete, 101
  denial, 99
  description, 99
  dnssec, 94
  dnssec-mode, 86
  dnssec-policy, 86
  do-not-listen, 81
  domain, 86
  drop-before-load, 86
  edns0-max-size, 81
  file-name, 87
  gid, 81
  hidden-master, 81
  hidden-primary, 81
  hostname-chaos, 81
  id, 99–103
  inactive, 101
  iterations, 103
  journal-size-kb, 87
  key-roll, 100
  key-suite, 99
  key-template, 100
  keys-path, 81, 87
  ksk, 102
  listen, 81
  log-path, 81
  log-unprocessable, 81
  maintain-dnssec, 87
  masters, 87
  max-tcp-connections, 81
  max-tcp-queries, 81
  multimaster-retries, 87
  multiprimary-retries, 87
  name, 89
  no-master-updates, 87

no-primary-updates, 87
notifies, 87
notify, 87
notify-retry-count, 87
notify-retry-period, 87
notify-retry-period-increase, 87
nsid
  ascii, 98
  hex, 98
optout, 103
pid-file, 81
port, 82
primaries, 87
publish, 101
queries, 94
queries-log-type, 82
retry-count, 87
retry-period, 87
retry-period-increase, 87
rrl
  enabled, 98
  errors-per-second, 98
  exempt-clients, 98
  ipv4-prefix-length, 98
  ipv6-prefix-length, 98
  log-only, 98
  max-table-size, 98
  min-table-size, 98
  responses-per-second, 98
  slip, 98
  window, 98
rrsig-nsupdate-allowed, 87
salt, 103
salt-length, 103
secret, 89
server, 94
server-port, 82
serverid-chaos, 82
sig-validity-interval, 82, 87
sig-validity-jitter, 82, 87
sig-validity-regeneration, 82, 87
size, 102
statistics, 82
statistics-max-period, 82
stats, 94
system, 94
tcp-query-min-rate, 82
thread-affinity-base, 82
    thread-affinity-multiplier, 82
    thread-count-by-address, 82
    transfer-source, 82, 88
    true-multimaster, 88
    true-multiprimary, 88
    type, 88
    uid, 82
    version-chaos, 82
    xfr-connect-timeout, 83
    xfr-path, 83
    zone, 94
    zone-download-thread-count, 83
    zone-load-thread-count, 83
configuration file
    yadifad.conf, 27
    yadifad.conf.example, 13
    yakeyrolld.conf.example, 13

Denial of Service, 63
Distributed Denial of Service, 63
DNS, 7, 8, 39, 47–50, 61–64, 80, 82, 96, 112, 113
DNSSEC, 7, 11, 38–40, 43, 47–50, 52–54, 58, 64, 65, 72, 73, 78, 81, 86, 87, 94
dnssec-policy, 21, 50, 54
domain name
    ., 33
    .eu, 7, 8
    , 23, 53
    eu, 8, 9
    eurid.eu, 9, 48

EDNS0, 7, 81
encoding
    ascii, 61
    BASE16, 56
    sha1, 49, 55, 56

hardware
    CPU, 11, 49, 81, 82
    RAM, 86

IXFR, 7, 25, 67, 69, 71, 73, 80, 82, 83, 86, 88, 113

library
    dnscore, 13
    dnsdb, 13
    dnslg, 13

man

yadifa.8, 13
yadifa.conf.5, 13
yadifa.rc.5, 13
yadifad.8, 13
yadifad.conf.5, 13
yakeyrolld.8, 13
yakeyrolld.conf.5, 13

NSEC3
Opt-Out, 43

operating system
AUR, 19
Debian, 18, 19
Gentoo, 20
Ubuntu, 19
BSD, 7
FreeBSD, 20
GNU, 14
GNU/Linux, 7, 12
macOS, 7, 20
OpenBSD, 20
Solaris, 20

rcode
NOTAUTH, 68
SERVFAIL, 68
resource record, 9, 25, 38, 39, 41–43, 47–49, 51–53, 56, 61, 107
resource record set, 47, 51, 65
resource record type
A, 49
CNAME, 26
DNSKEY, 26, 39, 42, 48, 49, 51, 52, 65, 66
DS, 26, 48, 49, 52, 65, 103
MX, 49
NS, 8, 26, 49
NSEC, 7, 26, 43, 52, 53, 55, 56, 94, 99, 102
NSEC3, 7, 43, 49, 52–56, 58, 94, 99, 102, 103
NSEC3PARAM, 53, 55
RRSIG, 26, 38, 39, 42, 43, 47, 48, 51, 52, 56, 65, 66, 87
SOA, 9, 26, 67, 69, 71–73
response rate limiting, 63
rfc, 7, 25
1034, 104
1035, 104, 108
1183, 108
1348, 108
1637, 108
1706, 108
1712, 108
1876, 108
2163, 108
2168, 108
2230, 109
2308, 108
2535, 108
2536, 108
2537, 108
2539, 108
2782, 108
2874, 109
2915, 108
2931, 108
3008, 108
3110, 108
3123, 109
3225, 109
3226, 109
3403, 108
3596, 108
3658, 109
3755, 108, 109
4025, 109
4034, 49, 108, 109
4255, 109
4398, 109
4431, 109
4701, 109
5155, 49, 109
5205, 109
5702, 49
5864, 108
6563, 109
6672, 109
6698, 109
6742, 109
6891, 109
7043, 109
7208, 109
7344, 109
7477, 109
dns notify, 68–70, 72, 73, 87
dns update, 25, 29, 35, 65, 72–74
FQDN, 23, 78, 96
IP, 8, 9, 24, 61, 62

KSK, 26, 38, 42, 43, 49–52, 54, 55, 57, 58, 65, 66, 99, 102
NSID, 61, 97
opt-out, 53
SEP, 51, 52
ssl, 12
TCP, 26, 64, 71, 81, 82, 96, 112, 113
TLD, 48
TSIG, 22, 26, 68, 96, 139
UDP, 63, 71, 72, 96, 112
ZSK, 38, 39, 42, 49–52, 54, 55, 57, 58, 65, 66, 99, 102
rrl
seeresponse rate limiting, 63, 64, 98, 112, 114

section
acl, 21, 75, 78, 90
channels, 21, 22, 75
denial, 22, 53, 55, 56, 76, 99, 100, 103
dnssec-policy, 21, 22, 53–56, 58, 75
key, 21, 22, 75
key-roll, 22, 53, 59, 60, 76, 100
key-suite, 21, 53–55, 57, 75, 99, 100
key-template, 22, 53, 58, 76, 100
keyword
activate, 59
algorithm, 58
allow-control, 27
create, 59
data-path, 87
delete, 59
denial, 54
dnssec-policy, 54
errors-per-second, 64
generate, 60
id, 54, 55, 57, 59, 60, 99–103
inactive, 59
include, 22
ipv4-prefix-length, 64
ipv6-prefix-length, 64
iterations, 55, 56
ixfr-from-differences, 73
key-roll, 57
key-suite, 54, 55, 58
key-template, 57
master, 88
max-table-size, 64
min-table-size, 64
multiprimary-retries, 68
optout, 55, 56
primaries, 67, 74
primary, 88
publish, 59, 60
responses-per-second, 64
salt, 55–57, 103
salt-length, 55–57, 103
salt-length's, 57
secondary, 88
section-key-roll, 101
section-key-template, 102
size, 58
slave, 88
slip, 64
true-multiprimary, 69, 70, 72–74
window, 64
xfr-retry-delay, 70
xfr-retry-failure-delay-max, 70
xfr-retry-failure-delay-multiplier, 70
xfr-retry-jitter, 70
loggers, 21, 22, 75
main, 21, 22, 27, 70, 75, 87, 88
nsid, 21, 22, 75
rrl, 21, 22, 75
zone, 21, 22, 53, 54, 74, 75
software
cron, 38
EPEL, 17, 18
GCC, 14
LibreSSL, 7, 12
OpenSSL, 7, 12

tld
.eu, 48
root (.), 8, 48, 49
TTL, 26, 41, 42