DNSSEC

Operational HOWTO.

*Olaf M. Kolkman**

RIPE NCC
November 12, 2002

Revision : 1.3

Contents

1 Introduction **3**
1.1 On this document 3
1.2 State of DNSSEC 4

I DNSSEC Tutorial **4**

2 Securing Zone Transfers **4**
2.1 Task at hand 4
2.2 Configuring TSIG Keys 5
2.3 Primary servers configuration of TSIG 7
2.4 Secondary servers configuration of TSIG 8
2.5 Troubleshooting TSIG configuration 8
2.6 TSIG signing of Notifies 9
2.7 Possible problems 9

3 Configuring a caching forwarder as verifier **9**
3.1 Task at hand 9
3.2 Configuring the caching forwarder 10
3.3 Troubleshooting a verifier 11

4 Setting up a locally secured zone **11**
4.1 Task at hand 11
4.2 Creating a key pair 12
4.3 Zone signing 15
4.4 Caching forwarder configuration 16

* (OKolkman@ripe.net)

4.5 Zone resigning . . . 16
4.6 Troubleshooting zone signing . . . 18
4.7 Possible problems . . . 19

5 Delegating of Signing authority; becoming globally secure. 19
5.1 Task at Hand . . . 20
5.2 Key exchange, signing and securing. . . . 21
5.3 Possible problems . . . 21

6 Rolling over keys 22

7 Emergency Rollover 24

8 How to proceed 25
8.1 Information . . . 25
8.2 Development Tools . . . 26
8.2.1 `lwres` . . . 26
8.2.2 `Net::DNS::SEC` . . . 26

II Demo Specifics 26

9 The 'DEMO' environment 26
9.1 Domain name setup . . . 26

10 example files 27
10.1 named.conf . . . 27
10.2 example zone file . . . 29

11 References, Acknowledgments and Copyright 30

III Appendix 31

A The RIPE NCC and DNSSEC 31
A.1 The RIPE NCC DISI project . . . 31
A.2 DNSSEC . . . 32

1 Introduction

1.1 On this document

This document has been produced as part of the RIPE NCC DNSSEC course.

The DNSSEC course consist of two parts. An 'Introduction to DNSSEC' and an 'real-life' demonstration.

The demonstration is presented by means of a slide show and is accompanied by this document. 'The Introduction' is presented using slides only. For this document it is assumed that the reader is familiar with the basic principles of DNSSEC, as presented in 'The Introduction'.

This document is tailored towards the course, but we have tried to write it in such a way that it can also be useful as a "HOWTO" when setting up DNSSEC in ones own environment.

In this HOWTO we will concentrate on the following tasks:

- Securing Zone Transfers. (section 2)
- Securing Zones for local use; how to secure your organization (section 4). When you learned and implemented this, you can be sure that DNS data in your organization is not tampered with. If you have a locally secured zone it is a small step to become part of a chain of trust.
- Delegating Signing authority; building a chain of trust (section 5). You will learn how to exchange keys with your parent and with your children.
- Rolling over keys. (section 6)

In every section we describe the task at hand, the tools that are needed, some troubleshooting techniques and possible problems one might encounter. We have used BIND version 9.3.0s20020722[1] during the production of this document, there will be minimal differences with respect to newer versions of BIND since the availability and functionality of tools may differ.

In the text we will refer to zone and system administrators. A zone administrator is the person that is responsible for zone data maintenance. A system administrator is responsible for maintaining the nameserver system. These 2 functions may be united in one person.

The examples given here are based on an example environment (section 9) from one of our workshops setup. In the workshop network (isolated from the Internet) we run our own root with one top level domain: `tld`. The `tld` top level domain does not exist on the Internet.

The material for the DNSSEC course is available from *www.ripe.net/disi/*. We are trying to improve the material wherever we can, so, comments or suggestions for format of the HOWTO, the slides or this text are welcomed by the author.

DNSSEC operational procedures are a moving target and tools for key management and rollover are not yet available. Therefore updates of this document may be expected in the near future. (See *www.ripe.net/disi/*.).

[1] Bleading edge technology at the time of writing.

1.2 State of DNSSEC

DNSSEC[rfc2535] is on the IETF standards track and has been implemented in the 'BIND' nameserver software (versions 9.0 to 9.2). The operational experience by early deployers, among which the RIPE NCC, has indicated that the DNSSEC protocol needs to be modified to be able to cope with operational problems that occur with delegation of authority.

Solutions for these scaling problems have been proposed and have been implemented in 'snapshot' versions of BIND 9.3. This document follows the assumes DNSSEC as in RFC2535 but with DS. Issues like NXT, OPT-IN and the exact semantics of the AD bit are still under discussion. As soon as IETF concensus on these issues is reached and supporting software is available, we'll update this HOWTO.

Part I
DNSSEC Tutorial

2 Securing Zone Transfers

2.1 Task at hand

Secure zone transfers are achieved by providing authentication of the servers and by providing data integrity during transfer.

The mechanism used to enable this is referred to as TSIG [rfc2845] and is based on a shared secret. The shared secret is used to sign the content of each DNS packet. The signature can be used for both authentication and for integrity checking of the data. In order to prevent a malicious third party to retransmit captured data (replay attack) a time stamp is included in the data. The TSIG mechanism can also be used to prevent unauthorized zone transfers; only owners of the secret key are able to do a zone transfer[2]

We will describe how primary server pri.ws.disi and secondary server sec.ws.disi need to be configured to enable TSIG for zone transfers.

To configure TSIG one has to perform the following steps:

- Synchronize clocks.
- Create and distribute a shared secret.
- At the primary server, create an access list specifying which keys are allowed transfer.
- At the secondary server, tell which keys to use when contacting which primary servers

[2]The data in the DNS is public data; disabling zone transfers does not guarantee your DNS data to become 'invisible'.

The first item is a pre-requisite for DNSSEC. If you do DNSSEC you should be in sync with the rest of the world: Use NTP. Time zones can be confusing. Use `date -u` to verify if your machine has the proper `UTC` time.

TSIG configuration is a task for system administrators.

2.2 Configuring TSIG Keys

To secure a zone transfer the primary server and the secondary server administrators have to configure a TSIG key in `named.conf`. The TSIG key consists of a secret and a hashing algorithm and are identified by domain names. We recomend that you maintain the list of secret keys in a separate file which is readable by root only and included in the `named.conf` file (e.g. by `include /var/named/shared.keys`).

The key statement has the following syntax:

```
key key_id {
    algorithm string;
    secret string;
};
```

This statement needs to be exactly the same for the two parties involved.

The *key_id* is a domain name uniquely identifying the key. If you have a large number of secret keys to maintain you should use a fully qualified domain name to avoid name clashes. We recommend you use a *key_id* that identifies both ends such as `pri-sec.ws.disi.`. For the *algorithm_string* you have no choice; Currently the only supported algorithm in the TSIG specification is 'hmac-md5'. The *secret_string* is the shared secret in a base-64 encoded string.

One can use `dnssec-keygen` to generate a truly random secret or use a pass-phrase - we describe both methods below.

Generating a TSIG secret with dnssec-keygen

`dnssec-keygen` is the tool that we use to generate keys (see figure 1, page 13 for the syntax and arguments).

We will use `dnssec-keygen` to generate a base64 encoded random number that will be used as the *secret_string*. The arguments that we have to provide `dnssec-keygen` to generate a TSIG key are:

-a	HMAC-MD5	the algorithm used
-b	*bitsize*	The length of the key generated, 128 bit is recommended in [rfc2845]
-n	HOST	The nametype, HOST is used for this purpose. The nametype is not really relevant when generating shared secrets.
name		the domain name you will be using to identify this key.
	Optional Arguments	
-r	/dev/urandom	This option is needed if /dev/random is not provided with sufficient entropy. Use /dev/urandom on FreeBSD systems (4.4-STABLE and before) or when experience 'time-outs' without this option.

The command produces two files[3] as output. The name of the files contain relevant information: K*domain_name*+*algorithm_id*+*key_id*.*extension*. The *domain_name* is the name specified as the name of the key. The *algorithm_id* identifies the algorithm used: 5 for HMAC-MD5 (1 and 3 are for RSA and DSA respectively see section **??**). The *key_id* is an identifier for the key material, it is not of relevance for symmetric keys. The *extension* is either "`key`" or "`private`", the first one is the public key and the second one is the private key.

The format of these files differs a bit but they contain exactly the same information; a base64 encoded random number that you are going to use as a shared secret. Do not be misdirected by the extensions "private" and "key", both the files should be kept secure[4]!

Note that the `-n HOST` and the *name* are not used for the generation of the base64 encoded random number. It is a convention to use the unique domain-name label that is used to identify the key as the *name*.

```
# dnssec-keygen -r /dev/urandom -a  HMAC-MD5 -b 128 \\
-n HOST pri-sec.ws.disi
Kpri-sec.ws.disi.+157+12274

# cat Kpri-sec.ws.disi.+157+12274.key
pri-sec.ws.disi. IN KEY 512 3 157 gQOqMJA/LGHwJa8vtD7u6w==

# cat Kpri-sec.ws.disi.+157+12274.private
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: gQOqMJA/LGHwJa8vtD7u6w==
```

[3] Two files are overkill if one only needs one random number.

[4] Since the secret material is copied to the configuration files and these files are not used in production, you may consider to delete them.

The base64 encoded random number is: `gQOqMJA/LGHwJa8vtD7u6w==`, it should be inserted in the `secret` of the `key` definition:

```
key  pri-sec.ws.disi.{
    algorithm hmac-md5;
    secret "gQOqMJA/LGHwJa8vtD7u6w==";
};
```

This key definition should be included in both primary and secondary nameserver configuration file. It is recommended to generate a secret for every different party you are involved with and you will need to maintain as many secrets as zones you are secondary for.

Other ways to generate secrets

The `dnssec-keygen` command provides you with a truly random bit sequence. It might be difficult to communicate the secret to your colleague running a secondary server on the other side of the world. In those cases you may want to choose to fall back to a pass-phrase that can be communicated over the telephone.

You can use any base64 encoder to convert the pass-phrase to a valid string in the key-definition.

```
# echo "Crypto Rules" | mmencode
Q3J5cHRvIFJ1bGVzCg==
```

or if mmencode is not available maybe this perl script can assist you.

```
#!/usr/bin/perl
use MIME::Base64;
print encode_base64("@ARGV") ;
```

Actually any string that can be base64 decoded will do.
`ThisIsAValidBase64String` can also be used as *secret_string*.

2.3 Primary servers configuration of TSIG

Both the primary and secondary server should have shared secret configured by using the `key` definition in a file included in `named.conf` (see above).

The primary server can now use the `key` in what BIND calls an *address_match_list*. These lists appear in the `allow-notify`, `allow-query`, `allow-transfer` and `allow-recursion` statements which controls access to the server.

Also see section 6.1.1 and 6.2.14.3 of the online BIND documentation

Relevant at this point is the `allow-transfer` in the `zone` definition. Using the key generated above the primary server for `ws.disi` would have the following statement in `named.conf`:

```
zone "ws.disi"  {
      type master;
      file db.ws.disi;
      \\ allow transfer only from secondary server that has
      \\ key pri-sec.ws.disi.
      allow-transfer { key pri-sec.ws.disi. ; };
      notify yes;
};
```

2.4 Secondary servers configuration of TSIG

Both the primary and secondary serve should have shared secret configured by using the `key` definition in `named.conf` (see above).

The `server` definition in `named.conf` is used to instruct the nameserver to use a specific `key` when contacting another nameserver.

```
\\ secondary for ws.disi
\\ primary server pri.ws.disi is on 10.1.1.2
server 10.1.1.2 {
        keys { pri-sec.ws.disi.;};
};
```

2.5 Troubleshooting TSIG configuration

You can check the format of your `named.conf` using the `named-checkconf` program. This program reads the configuration file using the same routines as `named` itself.

To troubleshoot your configuration you have the log file and `dig` at your disposal.

Before adding the `allow-transfer { key pri-sec.ws.disi.  ; };` you should be able to transfer the domain from any machine. `dig @pri.ws.disi ws.disi AXFR` should be successful. After key configuration the same command should give you output similar to:

```
; <<>> DiG 9.2.0rc1 <<>> @pri.ws.disi ws.disi AXFR
;; global options: printcmd
; Transfer failed.
```

You can test if the key is configured correctly in two ways.

1. You ask the zone administrator to increase the SOA serial and to have the zone reloaded on the primary server. The secondary server should pick up the changes.

   The log file of the secondary server will have entries like:

   ```
   ... general: info: zone ws.disi/IN: transfered serial 2001082801
   ... xfer-in: info: transfer of 'ws.disi/IN' from 10.1.1.2#53: end of transfer
   ```

2. You use `dig` to test the key by using `dig` with the `-k` file:

   ```
   dig @pri.ws.disi ws.disi AXFR -k Kpri-sec.ws.disi.+157+12274.key
   ```

   Alternatively you can use the `-y` switch and specify the key-name and the secret[5] with the `-y` switch;

   ```
   dig @pri.ws.disi ws.disi AXFR -y pri-sec.ws.disi.:gQOqMJA/LGHwJa8vtD7u6w==
   ```

   should do the trick.
   The log file of the primary server you tried this against will have entries similar to the following if the key did not match.

   ```
   .. security: error: client 10.1.1.6#1379: zone transfer 'ws.disi/IN' denied
   ```

2.6 TSIG signing of Notifies

TODO: add server directive for master, add allow-notify directive for slave.

2.7 Possible problems

Timing problems Machines that are involved in a TSIG signed transaction need to have their clocks synchronized within a few[6] minutes. Use 'NTP' to synchronize the machines and make sure the time zones are properly configured. A wrong time-zone configuration can lead to hard to spot problems; use `date -u'` to check what your machine thinks is the 'UTC' time.

Multiple server directives TSIG is a mechanism to protect communication on a per machine basis. Having multiple server directives for the same server or multiple keys in one server directive might lead to unexpected results... as a matter of fact it most certainly will lead to unexpected results.

3 Configuring a caching forwarder as verifier

3.1 Task at hand

We want to configure a caching server as a verifying cache. Users that use this cache as their nameserver will in this way only receive data that is either verifiable secure or verifiable insecure any spoofed data (marked 'bad') will not find it's way to the the users.

By configuring a public key for a specific zone we tell the caching forwarder that all data coming from that zone should be signed with the private key of that key pair. The zone acts as a secure entry point of the DNS tree and the key configured in the verifiable caching nameserver acts as the start for a chain of trust. In an ideal situation you have only one key configured as a secure entry point: the key of the root zone.

[5] Take care when using secrets on the command line of multi-user systems: on most Unix system command line arguments are visible on the output of `ps -auxwww` or via the `/proc/` file system

[6] BIND has 5 minutes hard coded

We assume you have a nameserver that has been configured as a caching-only nameserver and that another nameserver in your organization has been configured to run as an authoritative server for a secured zone called `sub.tld.`. Notes on how to setup a secured zone can be found below in section 4.

3.2 Configuring the caching forwarder

First you have to get hold of the public key that you want to configure. It is possible to get it from the DNS, but you will need to verify the key you obtained by an off-line mechanism. There are several possible ways that this can be done and it all depends on the policy of the zone administrator. The key could be published in the national newspaper, it can be published on a website that can be authenticated by a (trusted) 3rd party certificate. Make sure that you do not use the DNS to verify the key and you will be OK.

Besides that you have to verify the keys you obtaied out of band you have to make sure that the key you configure is the so called 'key signing key'. If the zone maintainer so decides he or she may make a destinction between keys. Some of them may be used to only sign key RR sets, others may be used to sign all the data in the zone. This distinction can only be communicated off-band since the KEY RR data provides no information on the operational use of the keys.

We assume you have obtained the key-signing key of `sub.tld.`.

Once you obtained the key you have to include the following statement in the `named.conf` of the caching forwarder.

```
trusted-keys {
 "sub.tld.''  256 3 3 ``CLGc9Re9I3pg7QAqx ...
                        VL24iYygAbSu3''

};
```

The format lookes like an entry in a zone file but note the quotes around the label and the public key material, they are not to be forgotten. There is no CLASS and TTL.

As soon as a `trusted-key` has been configured, data from that zone or its sub zones will be verified by the caching forwarder. If data is verified by the caching forwarder the `ad`-bit[7] will be set by the forwarder (see the 'flags' in the following example).

```
; <<>> DiG 9.3.0s20020722 <<>> +dnssec +retry=1 @10.0.53.204 sub.tld SOA
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36689
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

....
```

[7]The ad bit is defined in [rfc2535] section 6.1, the semantics of the AD bit are being fine tuned and may be subject to change in the near future[WG].

3.3 Troubleshooting a verifier

It is important that you check on the proper working of the verifyer. This can be done by turning on the logging for the veryfining nameserver by using a channel (dnssec_log in the example below) to which the errors of the 'dnssec' category are directed. Make sure you log at least at 'severity debug 3' that way you will be able to see the chains of trust being validated.

```
logging{
        channel dnssec_log {                  // a DNSSEC log channel
                file "log/dnssec" size 20m;
                print-time yes;                  // timestamp the entries
                print-category yes;              // add category name to entries
                print-severity yes;              // add severity level to entries
                severity debug 3;                // print debug message <= 3 t

        };

  category dnssec  { dnssec_log;  };
}
```

The output in the logfile will look similar to the output below. The attempt for positive response validation shows how the verifyer tries to prove that the RR set is trusted by following the chain of trust to the appropriate secure entry point, your trusted-key statement.

```
debug 3: client 10.0.53.204#1164: request is not signed
debug 3: client 10.0.53.204#1164: recursion available
debug 3: client 10.0.53.204#1164: query (cache) approved
debug 3: validating sub.tld SOA: starting
debug 3: validating sub.tld SOA: attempting positive response validation
debug 3: validating sub.tld KEY: starting
debug 3: validating sub.tld KEY: attempting positive response validation
debug 3: validating sub.tld KEY: verify rdataset: success
debug 3: validating sub.tld KEY: signed by trusted key; marking as secure
debug 3: validator @0x81e8800: dns_validator_destroy
debug 3: validating sub.tld SOA: in fetch_callback_validator
debug 3: validating sub.tld SOA: keyset with trust 7
debug 3: validating sub.tld SOA: resuming validate
debug 3: validating sub.tld SOA: verify rdataset: success
debug 3: validating sub.tld SOA: marking as secure
debug 3: validator @0x81e0000: dns_validator_destroy
```

4 Setting up a locally secured zone

4.1 Task at hand

The purpose of this exercise is to protect the DNS data as published by the zone administrator. Once we have configured the organization's resolvers and signed the zones DNS data, we can be sure that data is not tampered with; not while the DNS data is on the wire, not while the data lives on the master or slave servers' disks.

We will configure a caching forwarder to verify the DNS data that exists within our organization. If somebody tampered with that data the forwarder will notice this. Resolvers using this caching forwarder (it sits in `/etc/resolv.conf`) will only receive secure or verifiable unsecured data.

We want to sign the zone data for our own organization (say `sub.tld.`) and configure the caching forwarders on our organizations network to verify data against the public key of our organization. In terms of [rfc3090] this is called a locally secured zone. Note that in the text below, the assumption holds that your organization's domain names are maintained in one zone. If domain name administration is delegated to sub zones things get more complicated, see section 5.

At this moment the caching forwarder is the only architectural element for which DNSSEC verification is implemented. There are no applications that will do their own verification yet, nor are there stub-resolvers that are able of retrieving and verifying SIG resource records.

To establish this we need to create a public key pair, sign our zones and publish the data. We also need to configure the caching forwarders with the public key.

Signing the zone data is the task of the zone administrator, configuring the caching forwarder is a task of system administrators.

The examples are based on the example zone in section 10.2 on page 29.

4.2 Creating a key pair

`dnssec-keygen` is the tool that we use to generate keys (see figure 1, page 13 for the syntax and arguments).

We will use `dnssec-keygen` to generate a private-public key pair. The arguments that we have to provide `dnssec-keygen` are:

-a	RSA, RSAMD5, DH, DSA, RSASHA1	the algorithm used
-b	*bitsize*	The length of the key generated, seizes between 512 and 4096 bits depending on the algorithm.
-n	ZONE	The nametype, ZONE is used for DNSSEC.
name		the domain name of the zone you will sign with this key.
	Optional Arguments	
-r	/dev/urandom	This option is needed if /dev/random is not provided with sufficient entropy. Use this option on FreeBSD systems (4.6-Stable and before) or when experience 'time-outs' without this option.

```
Usage:
    dnssec-keygen -a alg -b bits -n type [options] name

Required options:
    -a algorithm: RSA | RSAMD5 | DH | DSA | RSASHA1 | HMAC-MD5
    -b key size, in bits:
        RSAMD5:         [512..4096]
        RSASHA1:        [512..4096]
        DH:             [128..4096]
        DSA:            [512..1024] and divisible by 64
        HMAC-MD5:       [1..512]
    -n nametype: ZONE | HOST | ENTITY | USER
    name: owner of the key
Other options:
    -c <class> (default: IN)
    -e use large exponent (RSAMD5/RSASHA1 only)
    -g <generator> use specified generator (DH only)
    -t <type>: AUTHCONF | NOAUTHCONF | NOAUTH | NOCONF (default: AUTHCONF)
    -p <protocol>: default: 3 [dnssec]
    -s <strength> strength value this key signs DNS records with (default: 0)
    -r <randomdev>: a file containing random data
    -v <verbose level>
Output:
     K<name>+<alg>+<id>.key, K<name>+<alg>+<id>.private
```

Figure 1: dnssec-keygen arguments

The output resides in two files. The name of the files contain relevant information: K*domain_name*+*algorithm_id*+*key_id*.*extension*. The *domain_name* is the name you specified on the command line, it is used by other BIND DNSSEC tools, if you use a name different from the domain name you might confuse those tools. The *algorithm_id* identifies the algorithm used: 1 for RSA, 3 for DSA, and 5 for HMAC-MD5 (TSIG only). The *key_id* is an identifier for the key material. This *key_id* is used by the SIG Resource Record. The *extension* is either "`key`" or "`private`" the first one is the public key, the second one is the private key.

We create an RSASHA1 keyset for `sub.tld`:

```
# dnssec-keygen -r/dev/urandom  -a RSASHA1 -b 1024 -n ZONE sub.tld
Ksub.tld.+005+28124
```

Lets have a look at the content of these file (we truncated the base64 key material):

```
cat Ksub.tld.+005+28124.key
sub.tld. IN KEY 256 3 5 AQO9+CFJ7j...
```

The public key (`.key` extension) is exactly as it would appear in your zone file. Note that the TTL value is not specified. The private key (`.private` extension) contains all the parameters that make an RSASHA1 private key.

```
Usage:
        dnssec-signzone [options] zonefile [keys]

Options: (default value in parenthesis)
        -c class (IN)
        -d directory
                directory to find keyset files (.)
        -s YYYYMMDDHHMMSS|+offset:
                SIG start time - absolute|offset (now)
        -e YYYYMMDDHHMMSS|+offset|"now"+offset]:
                SIG end time  - absolute|from start|from now (now + 30 days)
        -i interval:
                cycle interval - resign if < interval from end ( (end-start)/4 )
        -v debuglevel (0)
        -o origin:
                zone origin (name of zonefile)
        -f outfile:
                file the signed zone is written in (zonefile + .signed)
        -r randomdev:
                a file containing random data
        -a:     verify generated signatures
        -p:     use pseudorandom data (faster but less secure)
        -t:     print statistics
        -n ncpus (number of cpus present)
        -k key_signing_key

Signing Keys: (default: all zone keys that have private keys)
        keyfile (Kname+alg+tag)
```

Figure 2: dnssec-signzone arguments

The private key of a RSA key contains different parameters then DSA. Here is the private key (with base64 material truncated):

```
# cat Ksub.tld.+005+28124.private
Private-key-format: v1.2
Algorithm: 5 (RSASHA1)
Modulus: vfghSe400Lvrii83V4mZboA2PaEUgvhU1Oi...
PublicExponent: Aw==
PrivateExponent: fqVrhp7N4H1HsXTPj7EQ9FV5fmt...
Prime1: 9/TIUIQQTfAEbKNNlHQPWjZSKYuAAs5iunEG...
Prime2: xCHHpFj8LExLz9arlUB5W8o+eGQXAkRcAWFS...
Exponent1: pU3a4FgK3qAC8xeJDaK05s7hcQeqrImXJ...
Exponent2: gsEvwuX9ct2H3+RyY4BQ59wppZgPVtg9V...
Coefficient: laKLjRAKVqXoxmDynjYa1NvzKivzvnc...
```

This private key should be kept secure[8] *i.e.* the file permissions should be set so that the zone administrator will be able to access them when a zone needs

[8] At the RIPE NCC we are working on a dedicated signing server that has SSH based access control. Based on which key is used to login a dedicated shell is opened; Zone maintenance shell that allows signing of zones; A key maintenance shell for key maintenance. Only system administrators have privileges to access the key-material itself.

to be signed. Besides, the BIND tools will, by default[9], look for the keys in the directory where signing is performed, and that might not be the most secure place on your OS.

4.3 Zone signing

A locally secured tree must have a it's apex KEY RRs included[rfc3090] so before you sign your zone you have to add the public keys to your zone. You can do this by simply including the keys into your zone file.

Include the public keys (`.key` extension) in your zone file. Taking the example file from 10.2 we use the `$INCLUDE` directive to include the key. Of course, we do not forget to increase the serial number in the SOA. Note that in the example we include 3 keys. One RSAMD5 key (algorithm 1) one DSA (algorithm 3) and one RSASHA1 key (algorithm 5).

In the example below we will use the DSA key as the key signing key and the RSA keys as zone signing keys. In practice you will probably not use different algorithms but stick to only one (RSASHA1 is the preferred algorithm) and you will probably only have 3 keys in your zone at key-rollover time.

```
$TTL 100
$ORIGIN sub.tld.
@                       100     IN      SOA     ns.registry.TLD. (
                                                olaf.ripe.net.
                                                2002050501
                                                100
                                                200
                                                604800
                                                100
                                                )

; In this example zone we have include  3 keys.
; The DSA key +003+19854 is designated key-signing keys.
; the other two are designated zone signing keys.
$include Ksub.tld.+001+23495.key
$include Ksub.tld.+003+19854.key
$include Ksub.tld.+005+28124.key
;...$snip...
```

Once the key is included in the zone file we are ready to sign the zone using the `dnssec-signzone` tool, see figure 2 for all the arguments. We use the `-o` flag to specify the origin of the zone; normally the origin is deduce from the zone file's name.

With the '-k <KEY>' we specify which key is to be used as the Key signing key. That key will only sign the KEY RR set in the apex of the zone. The keys that come as arguments at the end of the command are used to sign all the RR data for which the zone is authoritative.

```
/usr/local/sbin/dnssec-signzone  -r /dev/urandom \
   -k Ksub.tld.+003+19854.key \
   sub.tld Ksub.tld.+001+23495.key Ksub.tld.+005+28124 \
   db.ws.disi.signed
```

[9] It is possible to fully specify the path to the keys

In the example above the DSA key with ID 19854 is the key signing key. The RSA/MD5 key with ID 23496 and the RSA/SHA1 key with ID28124 are zone signing keys.

The signed zone file is reproduced in figure 3. We've truncated all base64 representations and modified the output of dnssec-keygen a little bit to compact the presentation somewhat. (It's hard to make it really readable). Note that the apex KEY RRset is the only RRset with 3 signatures, made with the zone and key signing keys. The other RRsets are only signed with the zone signing keys.

The signing process did the following:

- It sorted the zone in 'canonical' order.
- Inserted NXT records for every label.
- Added the key-id as a comment to each KEY-record.
- Signed the KEY RR set with 3 keys; the key signing key and the zone signing keys.
- Signed the other RRs with the two zone signing keys (algorithm 1 and algorithm 5)

The signatures where created with a (default) lifetime of 30 days from the moment of signing. If signatures have expired data can not be verified and your zone will go 'bad'. Therefore you will have to resign your zone every 30 days. Zone resigning is discussed below.

4.4 Caching forwarder configuration

Now that the DNS servers publish signed data we need to configure the 'clients' to verify the data. The clients in this context are caching forwarders. To do this just configure your caching forwarder with the public key you just generated for the zone. How to do this is described in section 3 above.

4.5 Zone resigning

When the signature is almost expired or when you have added a few records to your zone there are two ways to resign your zone data. You may choose whichever option depending on the level of automation and the zone file you have to sign and frequency you have to regenerate SIG records..

- You can regenerate the signed zone from the unsigned zone file. The signer will need to sort the zone again, generate all the NXT records and generate all SIG records.

  If you generate your zone file from a back-end database this is probably the preferred mechanism.

```
; File written on Sun Aug 25 15:35:39 2002
; dnssec_signzone version 9.3.0s20020722
sub.tld.                100     IN SOA  ns.registry.TLD. olaf.ripe.net. (
                                2002050501 ; serial
                                100        ; refresh (1 minute 40 seconds)
                                200        ; retry (3 minutes 20 seconds)
                                604800     ; expire (1 week)
                                100        ; minimum (1 minute 40 seconds)
                                )
                100     SIG     SOA 1 2 100 20020924133539 20020825133539 23495 sub.tld. TOK+XczqhuVjaQI9Jj7NUGiLqoBKXH  ...
                100     SIG     SOA 5 2 100 20020924133539 20020825133539 28124 sub.tld. fFmSLmJU1mpTGniRxUEtETpruste27p ...
                100     NS      ns.sub.tld.
                100     SIG     NS 5 2 100 20020924133539 20020825133539 28124 sub.tld. p/3GONcMJ7xs9xDUeCFTWUvJwHPBN6D ...
                100     KEY     256 3 1  AQPw02b9MnR8aJplOyI1CBOu1zBGi9x ... ; key id = 23495
                100     KEY     256 3 3 ( CLGc9Re9I3pg7QAqxPebTpbokbYb5nZ ...; key id = 19854
                100     KEY     256 3 5 (AQO9+CFJ7jTQu+uKLzdXiZlugDY9oRSC ...; key id = 28124
                100     SIG     KEY 1 2 100 20020924133539 20020825133539 23495 sub.tld. TjpdtpH9g2fBHAxY4uKcAZjBp5qEIklu4Bqg
                                XPxdXDCDzQzWm4hG5yVb7Wq1Zqo43 ...
                100     SIG     KEY 3 2 100 20020924133539 20020825133539 19854 sub.tld. CCfyqvBBycBfy+7e5n/ckKS9bGnunP ...
                100     SIG     KEY 5 2 100 20020924133539 20020825133539 28124 sub.tld. RdSsIVKjX16LOCeGGBARdludKZPI94N ...
                100     NXT     b1.sub.tld. NS SOA SIG KEY NXT
                100     SIG     NXT 1 2 100 20020924133539 20020825133539 23495 sub.tld. NRHRUSRZO6WRUq5RFQzYPPRmPWmsDPT ...
                100     SIG     NXT 5 2 100 20020924133539 20020825133539 28124 sub.tld. OE2ZV/7aGk3pRU+7BFSpgAPodD2dLbQ ...
b1.sub.tld.     100     IN A    10.0.2.1
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. Hgn+P4sySKPeH3k5UXtf18Em9NxWnBd ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. sBzmRe3s4HpPozyWwboggfmH+Ssv61j ...
                100     NXT     b2.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. NAWacUhKEDNKXAI8auoLTefJ1Cef2v8 ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. jSf/vyYw0+o+iRoq0RoWIgxv7mZPnEU ...
b2.sub.tld.     100     IN A    10.0.2.2
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. T7T1BNl16UC6guBzon9f04siJ90li ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. mgmBVLHlTzt0xTQYB1hpwdBiNTUXU ...
                100     NXT     b3.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. 68Gtt2KvIy1CIlOpmyNZCAU6HOK3bT3 ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. JJjaCoX3N3a5sHteULZmxTitp7adfLn ...
b3.sub.tld.     100     IN A    10.0.2.3
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. S4WbwAC537mymQamv9IB58qF/R5l7 ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. Lff5HcyvvZomZPDQOg3BAaPvvFGmO ...
                100     NXT     b4.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. 48sFT+OWOPFNmRtiPP9tjeZDbACZ9pM ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. hx2E6rX4hF1gBt9/cHc6ewL2e7C+rWW ...
b4.sub.tld.     100     IN A    10.0.2.4
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. XB93jhZciRUsyqItmlkBfYlFzbbVf ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. XTGUKGbm9F7Fj8rJ3FpL+HgQS2f7r ...
                100     NXT     corrupt.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. sGA83aVU7TcJyUTzLs90a2OLhXh1Zu1 ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. rLVJtLH7yLvQfaiV8h/GuFfVFMb4MAU ...
corrupt.sub.tld. 100    IN A    10.0.4.1
                100     SIG     A 1 3 100 20020924140716  20020825140716 23495 sub.tld.  P6c+Fz3Xyd4S7BrtcLV9ZpBDdLZF5gd ...
                100     SIG     A 5 3 100 20020924140716  20020825140716 28124 sub.tld.  P6IZYe8LI1ttZE1LQfovtYPgrO1SBwR ...
                100     NXT     e.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924140716 20020825140716 23495 sub.tld. YXPzAQLFWB7Je13Zxltzjk3O1YFrIaLy ...
                100     SIG     NXT 5 3 100 20020924140716 20020825140716 28124 sub.tld. DF1nf536tCzcEJ7sg5wr+no75wnLhdfo ...
e.sub.tld.      100     IN A    10.0.3.1
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. t6YOdiH8Pn0LjObxG96Cc+mpDUaca ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. iF1CnqIER9DPvwKO0+YSfO1dfrVYz ...
                100     NXT     d.e.sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. eP+ZdSgLQeLpEH9dSzQHZoNzMMlLkCo ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. HwWcau8MgUbGwFHrSqW9EOJ/qtzDzdb ...
d.e.sub.tld.    100     IN A    10.0.3.2
                100     SIG     A 1 4 100 20020924133539 20020825133539 23495 sub.tld. us8sloYEkr7LCx9QUYgBYyXJA3FMA ...
                100     SIG     A 5 4 100 20020924133539 20020825133539 28124 sub.tld. KNrQRvogTiFF+W0NdYHq3cmdiGDUw ...
                100     NXT     c.d.e.sub.tld. A SIG NXT
                100     SIG     NXT 1 4 100 20020924133539 20020825133539 23495 sub.tld. JnX3x/CTNfuAD26pQ4tVZ3OsKdLhsye ...
                100     SIG     NXT 5 4 100 20020924133539 20020825133539 28124 sub.tld. UNp4mGRrzW+22WmqIYNxVGruIuJH3Cp ...
c.d.e.sub.tld.  100     IN A    10.0.3.3
                100     SIG     A 1 5 100 20020924133539 20020825133539 23495 sub.tld. LIYCqFOBPZ5GwoXV9DzczDOR6776u ...
                100     SIG     A 5 5 100 20020924133539 20020825133539 28124 sub.tld. ZMvxWA7u0gLdlbCD1YarJt6j2ZpGY ...
                100     NXT     b.c.d.e.sub.tld. A SIG NXT
                100     SIG     NXT 1 5 100 20020924133539 20020825133539 23495 sub.tld. sB3G5/5/1iF7Zo6hzVXfMGaHMVJoMOY ...
                100     SIG     NXT 5 5 100 20020924133539 20020825133539 28124 sub.tld. hv/d3WaDKnzYGt+o3uhtaiWeoNwB/2H ...
b.c.d.e.sub.tld.  100   IN A    10.0.3.4
                100     SIG     A 1 6 100 20020924133539 20020825133539 23495 sub.tld. rNvBRWSdu7JmROukTa4aru3Ih5JC3 ...
                100     SIG     A 5 6 100 20020924133539 20020825133539 28124 sub.tld. TGsJ6Anpmjc6lYwRcspHGwZUW2aJK ...
                100     NXT     ns.sub.tld. A SIG NXT
                100     SIG     NXT 1 6 100 20020924133539 20020825133539 23495 sub.tld. DtkxAFY8npe6oNhoPygG+SrC4wUuiaf ...
                100     SIG     NXT 5 6 100 20020924133539 20020825133539 28124 sub.tld. pC2xPNWmZkhjyrHGXm1pDn6qDEaCn88 ...
ns.sub.tld.     100     IN A    10.0.53.203
                100     SIG     A 1 3 100 20020924133539 20020825133539 23495 sub.tld. bk1rO9uAq3IkKmJIh9batBebPab90 ...
                100     SIG     A 5 3 100 20020924133539 20020825133539 28124 sub.tld. dT/8iVka5QhkUOYa/URZVGLrogxub ...
                100     NXT     sub.tld. A SIG NXT
                100     SIG     NXT 1 3 100 20020924133539 20020825133539 23495 sub.tld. ccUwRGjL9ESfr6StzoXr2CEEOqKyzOH ...
                100     SIG     NXT 5 3 100 20020924133539 20020825133539 28124 sub.tld. J+iYWN5piOfP8BWDvHdlaVbZefIWYEA ...
```

Figure 3: Example of signed zonefile

- You can add the new records to the already signed zone file and then run that zone file through the signer. The signer will insert the new records and associate NXTs in the already sorted zone file and will only sign the new records and the records for which the signatures are reaching the end of their validity period.

You should build tools to maintain your signed zones, `cron`, `perl` and `make` are your friends. We are planning to build a repository of DNSSEC maintenance tools at the RIPE NCC, please also make your tools publicly available.

4.6 Troubleshooting zone signing

You can check the format of your `named.conf` using the `named-checkconf` program. The `named-checkzone` program can be used to check zone files. These programs use the same routines as `named` itself.

One can use `dig` and a nameserver configured with a `trusted-key` to verify your setup. If data cannot be cryptographically verified the forwarder will return with a `SERVFAIL` status. You can test this by intentionally corrupting a resource record in the signed zone file. This is typical output of `dig` when querying for corrupted data[10]:

```
; <<>> DiG 9.3.0s20020722 <<>> +dnssec @verifyer corrupt.sub.tld
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 36778
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;corrupt.sub.tld.                IN      A

;; Query time: 14 msec
;; SERVER: 10.0.53.204#53(verifyer)
;; WHEN: Sun Aug 25 16:22:46 2002
;; MSG SIZE  rcvd: 44
```

Note that a caching forwarder will not do cryptographic verification of zones it is authoritative for. So if your caching forwarder is primary or secondary for a particular zone you will always get an answer, it is assumed that data from disk is secure.

Further troubleshooting would need to be done on a verifier. Below is an example of the log output of the verifier when we queried for corrupted data.

```
debug 3: client 10.0.53.204#1206: request is not signed
debug 3: client 10.0.53.204#1206: recursion available
debug 3: client 10.0.53.204#1206: query (cache) approved
```

[10] We corrupted the data by modifying rdata in the signed zone-file

```
debug 3: validating corrupt.sub.tld A: starting
debug 3: validating corrupt.sub.tld A: attempting positive response validation
debug 3: validating sub.tld KEY: starting
debug 3: validating sub.tld KEY: attempting positive response validation
debug 3: validating sub.tld KEY: verify rdataset: success
debug 3: validating sub.tld KEY: signed by trusted key; marking as secure
debug 3: validator @0x81e9800: dns_validator_destroy
debug 3: validating corrupt.sub.tld A: in fetch_callback_validator
debug 3: validating corrupt.sub.tld A: keyset with trust 7
debug 3: validating corrupt.sub.tld A: resuming validate
debug 3: validating corrupt.sub.tld A: verify rdataset: SIG failed to verify
debug 3: validating corrupt.sub.tld A: failed to verify rdataset
debug 3: validating corrupt.sub.tld A: verify failure: SIG failed to verify
debug 3: validating corrupt.sub.tld A: keyset with trust 7
debug 3: validating corrupt.sub.tld A: verify rdataset: SIG failed to verify
debug 3: validating corrupt.sub.tld A: failed to verify rdataset
debug 3: validating corrupt.sub.tld A: verify failure: SIG failed to verify
info: validating corrupt.sub.tld A: no valid signature found
debug 3: validator @0x81e0800: dns_validator_destroy
```

4.7 Possible problems

SOA serial If you forget to increase the serial number before resigning your zone, secondary servers may not pick up the new signatures in time. Some resolvers will be able to verify your signature while others will not.

'Zone signing key' rollover If a zone administrator makes a distinction between zone and key signing keys then the rollover of a zone signing key will not involve any action of the administrators of the verifiers. If a key signing key is to be changed care should be taken that all resolvers in the organization have been supplied with a new `trusted-key`.

If the zone is only locally secured (i.e. is not part of a chain of trust) then the rollover of a key signing key is relatively simple. Remember that to verify data there has to be at least one signature that can be verified with the `trusted-keys` in resolvers; During a limited time you use two key signing keys to sign your zone: the old and new key. During that time you start reconfiguring the resolvers in your organization with new `trusted-keys`. Once all resolvers have the new key configured in their `trusted-key` statement, the zones should be signed with the new key only.

Rollovers are described in more detail in section 6

5 Delegating of Signing authority; becoming globally secure.

This section is subject to change as the tools needed for this are being modified/developed.

5.1 Task at Hand

We have covered how to deploy DNSSEC in a single zone. We now want to build a chain of trust so that once a client has securely obtained a public key high in the DNS hierarchy, it can follow the chain to verify data in your or your children's zone.

To be able to delegate authority the parent has to sign data that securely indicates which child key is to be used as the next step in the chain of trust. [Gud] describes how this can be established by having the parent generate a signature over the DS record that is generated from the childs KEY.

Below we will describe how to setup a zone that is globally secured based on the parental signature over the DS record pointing to the child's key signing key.

In the example we use `tld` as parent and `sub.tld` as child. We assume that the parent zone is already locally secure as described in the previous section. This means that the parent has no DS RR for `sub.tld.` and that resolvers that follow the chain of trust via `tld` will treat the `sub.tld.` zone as verifiable insecure. The `sub.tld.` zone assumed not to be secure, much of the procedure will be as described section 4, but, since keysets are used, some details are different.

Apex data is data that has the same name as the origin of a zone. Some resource records in the apex can appear at both the parent and the child. The only data that is allowed at the parent and child side of the apex are NS, NXT, KEY and SIG. In [rfc1034, rfc1035] it is stated that the child is authoritative for data in the apex of a zone.

The DS RR is the only RR record that may only be published at the parent's apex and for which the parent is authoritative.

Therefore the key set signing keys, generated by the child, need to be exchanged with the parent to get the DS records generated from. Although you could build tools to have the child generate DS records so that the parent can include them in their zone file, the current tools use so called keysets.

Note that there is a command called `dnssec-makekeyset` available in most BIND 9 versions. The command is used to generate a keyset. However, the use of keyset has changed with the introduction of DS and the `dnssec-makekeyset` command has been removed from the BIND 9.3s20020722 distribution. Although the `dnssec-makekeyset` may reappear in later versions we describe the procedure as if the tool is not available.

The sequence of events will be:

1. The child gets it's key signing key to the parent.

2. The parent creates a keyset file in a special directory.

3. The `dnssec-signzone` command finds the keysets and automatically generates and signs DS records.

We will describe the steps in more detail below.

5.2 Key exchange, signing and securing.

We assume the child is secured and works locally. So we have created zone signing and key signing keys as described in section 4.2 and signed our zones as described in 4.3.

The parent will now need to obtain our key signing key.

The easiest way to upload the key is tho have the child cut and paste the key into an email and sending it to the parent.

In an operational environment it is extremely important that the authenticity and the integrity of the KEY is established[11]. The zone administrator[12] of the parent will need to verify that the key came from the zone administrator of the child zone. If possible this should be confirmed by an out-of-DNS mechanism. The parent could use it's customers database to verify if key was actually sent by the zone administrator. If a wrong key is signed the child zone will be vulnerable for attacks; signing the wrong key breaks DNSSEC.

The parent will need to create a keyset file. This is done by putting the key material in a file called `keyset-<child-domainname>`.

The parent stores the keysets in a directory that is to be specified with the `-d` flag of `dnssec-signzone`. The signzone tool will automatically generate the appropriate DS records if a `keyset-<child-domainname>.` file is found containing one or more KEY RRs for the `<child-domain>`. Note that although the keyset generated by the child contains signatures the SIG RRs do not need to be available in the `keyset-<child-domain>.` file at the parent, the sign tool will not do signature verification.

Below is an example on how you could ivoke the command:

```
dnssec-signzone -r /dev/urandom -d /registry/tld-zone/child-keys
 -o tld -f tld.signed db.tld
```

5.3 Possible problems

Public Key Algorithm To be globally secure one needs to use at least one key of an algorithm that is mandatory to implement. Mandatory to implement are RSA/SHA1 and DSA keys. We recomend the use of RSA/SHA1 keys only[13].

Parent indicating child security It is important that the KEY with that is sent to the parent is in use as key signing key (or as zone and key signing key if there is no distinction made) before the parent includes a signed DS RR for that key.

If the parent includes a DS RR while the child has not yet signed with the key then the child will go 'bad'; By not having a DS RR for the child the parent indicates the child to be in-secure.

[11] Tools to for verification of selfsigned keys are not yet available. There is work done in this area.

[12] The person who is responsible for publishing the zone data

[13] This document is not consistent with respect to that advise. We will update the document in a later version

As a parent you should always verify that the child publishes signed KEY before including a DS RR.

6 Rolling over keys

Roll over is the process where a zone decides to change their key. Since keys have a limited lifetime — If only because of Moore's law – they will need to be changed occasionally. Care needs to be taken that existing chains of trust are not broken during the change of key. In general this means that during a limited period two keys, the old and the new key are being used.

If the rollover is planned we refer to it as scheduled rollover. If the rollover is the result of a (suspected) compromise or loss of private key it is called a unscheduled or emergency key rollover.

If a zone rolls over a key it will need to contact it's it's parents. The children will not notice a rollover.

The middle-zone '`sub.tld`' wants to change its key signing key. The old key has tag=1, the new key has tag=2. The `sub.tld` zone uses key 10 as zone signing key. These are the steps that need to be taken. The parent's key has tag=123 and the child key has tag=789. These two keys do not change during the process.

step 0 The initial situation is the following:

The middle-zone has key '1' published signed by itself and the parent. The child has it's key signed by key '1' and publishes the signature over it's key made by the middle zone.

```
; published by the parent

sub.tld         DS <tag=1>
sub.tld         SIG (DS) <signkey=tld tag=123>


; Published in middle zone:
sub.tld.          KEY       <tag=1>      ; key signing key
sub.tld.          KEY       <tag=10>     ; zone signing key
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=1>
sub.tld.          SIG(KEY)  <signkey=sub.tld tag=10>

child.sub.tld     DS    <tag=789>
child.sub.tld     SIG(DS)   <signkey=sub.tld tag=10>


; Published in child zone:
child.sub.tld..     KEY    <tag=789>
child.sub.tld.    SIG(KEY)  <signkey=child.sub.tld tag=789>
```

The child should query one of the authoritative servers for it's DS RR and store the TTL it commes in handy in step 3.

step 1 The middle-zone now starts the rollover by uploading key 2 to it's parent and by publishing all keys in the DNS.

```
; published by the parent

sub.tld         DS <tag=1>
sub.tld         SIG (DS) <signkey=tld tag=123>


;  published by the middle zone
sub.tld.          KEY       <tag=1>
sub.tld.          KEY       <tag=2>
sub.tld.          KEY       <tag=10>
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=1>
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=2>
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=10>


child.sub.tld     DS    <tag=789>
child.sub.tld     SIG(DS)   <signkey=sub.tld tag=10>
```

step 2 Key 2 has been received by the parent. The parent now knows which key to use as secure entrypoint. They will be able to verify the new key against the existing chain of trust by using the KEY RR set from the DNS and know that key 2 is uploaded by the child because it is signed with key 2 alone, something that only the child can do.

The parent generates a new DS RR for pointing to key 2 of the middle zone.

```
; published by the parent

sub.tld.                DS <tag=2>
sub.tld.                SIG (DS) <signkey=tld tag=123>


; Published by middle zone
sub.tld                 KEY        <tag=1>
sub.tld                 KEY        <tag=2>
sub.tld                 SIG(KEY)   <signkey=sub.tld, tag=2>
sub.tld                 SIG(KEY)   <signkey=parent, tag=123>


child.sub.tld     DS    <tag=789>
child.sub.tld     SIG(DS)   <signkey=sub.tld tag=10>
```

If there are any resolvers with `trusted-keys` set to 1 than they should be replaced by key 2.

step 3 After a given time the middle zone needs to remove the old key. It does that by creating a key RR set with only the new key:

```
;  published by the middle zone
sub.tld.          KEY       <tag=2>
sub.tld.          KEY       <tag=10>
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=2>
sub.tld.          SIG(KEY)  <signkey=sub.tld, tag=10>

child.sub.tld     DS    <tag=789>
child.sub.tld     SIG(DS)   <signkey=sub.tld tag=10>
```

Note that the new parental DS RR will need to be propagated from the parents master to the parents slave servers and besides the old DS RR will need to have expired from the world's caches. At this point the TTL from the old DS RR commes in handy. Once you have seen the new DS picked up by all the authoritative servers of the parent you will have to wait for the value of the TTL before you throw away your old key.

These 3 steps involve interactions that are policy dependend. They can, to a large extend, be automated.

If one is able to distinguish key signing keys from zone signing keys then the process can be fully automated.[Kol].

7 Emergency Rollover

If your private key has been compromised or lost and your old keys are about to expire you have to be prepared for trouble.

It is important to plan in advance what to do if a emergency key rollover needs to take place. We advise you to create a procedure which you publish both on and off-line. This is a non-complete list of what such a procedure should contain.

Emergency procedures cannot be automated.

Responsible persons Document who has which responsibility in case of compromise also document how they can be contacted.

Parental Key exchange Your parent will need to know that you are compromised and will need to stop generating signatures over your old key. The parent will need to establish your identity in a similar way as during an initial key exchange, the parent should avoid that a non-authoritative 3rd parties signal the parent that a child's key should not be used. Document how this is done and what is needed to be able to do this.

Child Keys You will have to resign the DS records of your children. Make sure that you have all the keysets (or DS RRs) available.

Resolver Key exchange Resolvers that have your key configured as trusted-key need to be reconfigured. Do you have knowledge of which resolvers need to be reconfigured? Do you need to contact them via an off-band medium (shttp or news-paper).

New key generation How will you go about generating a new key. Who is to generate the new key.

Security assessment How did this happen? Will the new key be safe? Who will audit?

Key publishing Will you continue to publish the compromised key while the parental DS is pointing to it? If not you may 'go bad' and you and your children will drop from the earth. While the parental signature over the DS pointing to your key is valid you are in vulnerable to attacks by owners of your compromised key even while you do not publish it.

8 How to proceed

8.1 Information

BIND Book

The 'DNS and BIND' book by Paul Albitz & Cricket Liu[AL01] is a good starting point for any DNS work. As of the 4th edition there is a chapter on DNSSEC.

BIND online documentation

The BIND documentation should be at your disposal. It is in the distribution under the `bind-9-?-??/doc/arm/` directory.

DISI webpages

On the RIPE NCC Disi webpages one can information related to this course and to DNSSEC. See *www.ripe.net/disi* for details.

NLnet Labs DNSSEC resources

At NLnet Labs a thorough list with DNSSEC resources is maintained. This is a good starting point if you are looking for information. *http://www.nlnetlabs.nl/dnssec/index.en.html* .

The folk at NLnet Labs maintain an experimental secured zone `nl.nl`.

IETF working groups;

DNSSEC is developed within the IETF. Two working groups are actively involved in these development.

- The DNS Extensions (DNSEXT) workin group which charter can be found at *http://www.ietf.org/html.charters/dnsext-charter.html* . It's mailinglist is namedroppers@ops.ietf.org. .
- The Domain Name Server Operations (DNSOP) working group which charter can be found at *http://www.ietf.org/html.charters/dnsop-charter.html* . It's mailinglist is dnsop@cafax.se.

- DNSSEC mailinglist
  The majordomo mailinglist 'dnssec@cafax.se' has grown to be the mailinglist where people exchange DNSSEC specific information.

RFCs;

If you want to gain thorough understanding you have to resort to the RFCs. See chapter A4.1 of the BIND on-line documentation for the RFC that relate to DNS and DNSSEC.

8.2 Development Tools

8.2.1 `lwres`

BIND 9 comes with the light weight resolver library. It *"provides resolution services to local clients using a combination of a lightweight resolver library and a resolver daemon process running on the local host. These communicate using a simple UDP-based protocol, the "lightweight resolver protocol" that is distinct from and simpler than the full DNS protocol."*

For more detail see the BIND9 documentation, chapter 5.

8.2.2 `Net::DNS::SEC`

If you need to create your tools you might want to consider using the PERL `Net::DNS::SEC` module. The module contains classes for DS, KEY, SIG, NXT, methods for creation and verification of SIGs and DS records and some examples to get you started. See *www.ripe.net/disi/*. or *CPAN*. for the latest version of the module.

In figure4 we present an example of the use of Net::DNS::SEC. About 10 lines of code are needed to fetch a key from a nameserver, verify the signature and generate a DS set.

Part II
Demo Specifics

9 The 'DEMO' environment

Most of the examples from this text are drawn from the setup used in a workshop. Some of the details are presented below.

9.1 Domain name setup

The demo LAN has it's own root server on 10.0.53.201 ; The hints file reflects that situation (section **??**).

Another machine on the LAN (10.0.53.202) acts as a Top Level Domain (TLD) server. The name of the TLD is '`dtld`'. The 3rd nameserver on 10.0.53.203 acts as server of `sub.tld`.

```
###
# (... cut ...)

$res = Net::DNS::Resolver->new;
$res->dnssec(1);
$res->nameservers($nameserver) if defined $nameserver;
$packet = $res->query($domain, 'KEY', 'IN');
die "No results for query $domain KEY" if ! defined $packet;

$keyset=Net::DNS::Keyset->new($packet) ;
if ( ! $keyset ){
    print $Net::DNS::Keyset::keyset_err;
    return 0;
}


# Print DS records to STD out
#
my @ds=$keyset->extract_ds;
foreach my $ds ( @ds ) {
    $ds->print;
}
# (... cut ...)
```

Figure 4: Example code showing the use of Net::DNS::SEC

A 4th nameserver (10.0.53.204) will acts as caching forwarder. This is a machine that can be used to test against.

10 example files

10.1 named.conf

```
//
// Special setup

// tld configured on: 10.0.53.202
//

options {
        directory "/divers2/DNSSEC-Demo/tld/";
        pid-file "/divers2/DNSSEC-Demo/tld/named.pid";
        listen-on {10.0.53.202;};
        recursion no;
};


$include /var/named/dns.secret.keys

controls {
       inet 10.0.53.202 port 953
      allow { 127.0.0.1;}  keys { "rndc-key"; };
};
```

```
zone "tld" {
        type master;
        file "zones/tld.signed";
};

zone "." {
        type hint;
        file "zones/root.hints";
};

logging {
        channel syslog_channel {
                syslog daemon;        // end to syslog's daemon
                severity debug 6;
                print-severity yes;
                 print-category yes;
        };
        channel query_channel {
                file "log/querylog" size  5m ;
                print-time yes;
        };
        channel notify_channel {
                file "log/notify+update.log" size 5m;

                severity debug 6;
                print-time yes;
        };

        channel everything_else {
                file "log/runlog" size 5m;
                print-time yes;
                severity debug 6;
                print-severity yes;
                print-category yes;
        };


        channel dnssec_log {                  // a DNSSEC log channel
                file "log/dnssec" size 20m;
                print-time yes;                  // timestamp the entries
                print-category yes;              // add category name to entries
                print-severity yes;              // add severity level to entries
                severity debug 6;                // print debug message <= 3 t

        };
        category dnssec  { dnssec_log;  };
        category security { dnssec_log;  };
        category queries { query_channel; };
        category update  { update_channel; syslog_channel; };
        category notify  { notify_channel; syslog_channel; };
        category default { everything_else; };
};
```

Included in this file is the dns.secret.keys. This file is '`chown root /var/named/dns.secre`

```
; chmod 600 /var/named/dns.secret.keys'

// Example
// /var/named/dns.secret.keys
// Contains TSIG configuration
//

key  rndc-key.{
// echo 'Passhrase for TSIG' | mmencode -b
    algorithm hmac-md5;
    secret UGFzc2hyYXNlIGZvciBUU0lHCg==;
};
```

10.2 example zone file

```
.

;
; Demonstration sub.tld.
; not to be published on a server connected to the internet
;
$TTL 100
$ORIGIN sub.tld.
@                       100     IN      SOA     ns.registry.TLD. (
                                                olaf.ripe.net.
                                                2002050501
                                                100
                                                200
                                                604800
                                                100
                                                )


$include Ksub.tld.+001+23495.key
$include Ksub.tld.+003+19854.key
$include Ksub.tld.+005+28124.key



                        NS              ns.sub.tld.
ns.sub.tld.             A               10.0.53.203


; Query for a.b.c.d.e.sub.tld. to see which  NXT records are generated
; for the denial of existence of wildcards.

e                       A               10.0.3.1
d.e                     A               10.0.3.2
c.d.e                   A               10.0.3.3
b.c.d.e                 A               10.0.3.4

b1                      A               10.0.2.1
b2                      A               10.0.2.2
b3                      A               10.0.2.3
b4                      A               10.0.2.4
```

```
;
;  The data of 'corrupt' will be modified
corrupt                 A               10.0.4.1
```

11 References, Acknowledgments and Copyright

References

[AL01] Paul Albitz and Cricket Liu. *DNS and BIND, 4th Edition.* O'Reilly, 4 edition, April 2001.

[Bel95] Steven M. Bellovin. *Using the Domain Name System for System Break-ins.* In *Proceedings of the fifth USENIX UNIX Security Symposium: June 5–7, 1995, Salt Lake City, Utah, USA*, editor: USENIX Association, pages 199–208. USENIX, June 1995.

[Gud] Olafur Gudmundsson. *Delegation Signer record in parent. <draft-ietf-dnsext-delegation-signer-08.txt>*, June 2002. `ftp://ftp.ietf.org/internet-drafts/draft-ietf-dnsop-resolver-rollover-08.txt`, DNSOP WG Internet draft, drafts are subject to change and have a limited lifetime.

[Kol] Olaf Kolkman. *KEY RR Key Signing (KS) Flag<draft-ietf-dnsext-keyrr-key-signing-flag-00.txt>*, September 2002. `ftp://ftp.ietf.org/internet-drafts/draft-ietf-dnsext-keyrr-key-signing-flag-00.txt`, DNSOP WG Internet draft, drafts are subject to change and have a limited lifetime.

[Nem00] Evi Nemeth. *Securing the DNS. ;login:*, pages 21–31, November 2000.

[rfc1034] P. V. Mockapetris. *RFC 1034: Domain names — concepts and facilities.* IETF, November 1987. `ftp://ftp.ietf.org/rfc/rfc1034.txt`.

[rfc1035] P. V. Mockapetris. *RFC 1035: Domain names — implementation and specification.* IETF, November 1987. `ftp://ftp.ietf.org/rfc/rfc1035.txt`.

[rfc2535] D. Eastlake. *RFC 2535: Domain Name System Security Extentions.* IETF, March 1999. `ftp://ftp.ietf.org/rfc/rfc2535.txt`.

[rfc2845] P. Vixie, O. Gudmundsson, D. Eastlake, and B. Wellington. *RFC 2845: Secret Key Transaction Authentication for DNS (TSIG).* IETF, May 2000. `ftp://ftp.ietf.org/rfc/rfc2841.txt`.

[rfc3090] Edward Lewis. *RFC 3090: DNS Security Extensions Clarification on Zone Status.* IETF, March 2001. `ftp://ftp.ietf.org/rfc/rfc3090.txt`.

[Sch00] Bruce Schneier. *Secrets & Lies.* John Wiley& Sons, Inc, 2000. isbn 0471253111.

[Vix95] Paul A. Vixie. *DNS and BIND Security Issues.* In *Proceedings of the fifth USENIX UNIX Security Symposium: June 5–7, 1995, Salt Lake City, Utah, USA*, editor: USENIX Association, pages 209–216. USENIX, June 1995.

[Wbstr] *Merriam-Webster Collegiate Dictionary, Tenth Edition.* Merriam-Webster Inc. Webpages, 2001. `http://www.m-w.com/`.

[WG] Brian Wellington and Olafur Gudmundsson. *Redefinition of DNS AD bit <draft-ietf-dnsext-ad-is-secure-03.txt>*, July 2001. `ftp://ftp.ietf.org/internet-drafts/`, DNSOP WG Internet draft, drafts are subject to change and have a limited lifetime.

Note: URLs may be subject to change.

Acknowledgments

There are numerous people who helped compiling these notes, either by helping me to understand DNSSEC, by giving feedback on earlier courses. Special thanks go to Jakob Shlyter, Daniel Karrenberg and Daniel Diaz for the feedback they provided on this on the drafts. Miek Gieben and his colleagues of NLnet Labs and Roy Arends have been of great help with developing the tutorial. The workshop organized bu USC/ISI on operational testing of DS in Washington DC has provided a substantial amount of material for this HOWTO.

A 'login;' article by Evi Nemith[Nem00], the text in the BIND book[AL01] and the various presentations by Edward Lewis have been very helpful to gain understanding and to compare these notes against.

Document History

Version 1.1 of this document was compiled for a DNSSEC tutorial in Prague, October 8, 2000. The document were a set of notes to be used in a workshop setup.

Version 1.2 was not published.

Version 1.3 is the first modification of the document. First experiences with DS have been incooperated and the document has been rewritten to be useful as a more generic HOWTO and introduction to DNSSECoperations.

Part III
Appendix

A The RIPE NCC and DNSSEC

A.1 The RIPE NCC **DISI** project

As the Internet permeates more an more aspects of daily life, the need for security[14] has increased dramatically. Technologies providing security features

[14] [Wbstr] describes 'security' as *the condition or quality of being secure* and describes 'secure' as *To make safe; to relieve from apprehensions of, or exposure to, danger; to guard; to*

are being standardized and implemented. Many of these are end-to-end, being deployed as part of end-systems and applications. Some things, however, need to be deployed or supported in the Internet infrastructure itself. These need actions or at least awareness among ISPs and the RIPE community at-large.

The RIPE NCC is working to raise awareness and support deployment of security technologies in the Internet Infrastructure. This new effort is called "Deployment of Internet Security in the Infrastructure": DISI.

DISI strives to continue the successful deployment and awareness efforts of the RIPE NCC such as our support for the Implementation of CIDR. DISI will provide information about relevant technologies by means of white papers, seminars and workshops. We will also deploy the technologies in our own environment, collect deployment experiences by others and provide easy access to this knowledge in order to make deployment easier for everyone. As far as possible we will also keep track of the level of deployment in the RIPE community and report on that regularly.

DISI is definitely not an incident response coordination effort, nor is it targeted at general computer systems security, nor at the deployment of the end-to-end technologies mentioned above. Its scope is limited to security technologies that need to be deployed or supported by the Internet infrastructure.

The first activities of the DISI eefforts will be "Deployment of Domain Name Security on the reverse tree" and supporting a RIPE "Security WG".

A.2 DNSSEC

At the time the Domain Name System (DNS) — the distributed database that is used to translate domain names in IP addresses and vice verse [rfc1034] — was designed little thought was given to the authentication of the Resource Records (RR). This opens up the possibility for man in the middle attacks [Bel95, Vix95].

The DNS Security Extension (DNSSEC) was proposed by the IETF [rfc2535] to cope with these kind of vulnerabilities. It uses public key technology to sign RRs and ensure the integrity and authenticity of DNS data.

At the RIPE NCC we focus on securing the DNS infrastructure in the RIPE area. We approach this by organizing workshops like this and by deploying DNSSEC over a part of the in-addr tree — the DNS tree that maps IP-addresses into names.

protect In the context of this document we use the term 'being secure' when it is impossible to use resources or get to information without proper authorization.
A good general text on security in a networked environment is 'Secrets and Lies' [Sch00]).