DNSSEC HOWTO

A Tutorial in Disguise.

Olaf Kolkman, RIPE NCC
<olaf@ripe.net>

Published September, 2004

$Revision: 1.4.4.8 $

Table of Contents

List of Figures

Introduction

" My signature is worth more than my word."
--Anonymous swindler

About This Document

This document has been produced as part of a RIPE NCC project on the deployment of DNSSEC. The document started as being an addendum to a DNSSEC course that is being presented by the RIPE NCC but soon we realized that it is probably more useful as a stand alone "HOWTO" for setting up DNSSEC in ones own environment.

In this HOWTO, which because of its bulky nature is more a "Tutorial" in disguise, we touch upon the following topics.

- Part I, "Securing DNS data", about the aspects of DNSSEC that deal with data security.
    - Creating an island of security (Chapter 1, *Configuring a recursive nameserver to verify answers* and Chapter 2, *Securing a DNS zone*) by configuring a recursive nameserver to verify the signed zones served by your organizations authoritative nameservers. When you learned and implemented this, you can be sure that DNS data in your organization is not tampered with. Once you have created an island of security it is a small step to become part of a chain of trust.
    - Delegating Signing authority; building a chain of trust (Chapter 3, *Delegating of Signing authority; becoming globally secure*). You will learn how to exchange keys with your parent and with your children.
    - Chapter 4, *Rolling Keys* is about maintaining keys and ensuring that during the rollover process clients will be able to maintain a consistent view of your DNS data.
- Part II, "Securing communication between Servers", about aspects that deal with server to server security and transaction security.
    - Chapter 5, *Securing Zone transfers* is on the use of TSIG to provide authorisation and integrity for zone transfers.

Finally Appendix B, *Hints and tips* contains a few tips and hints that may help you to troubleshoot your setup or answer your questions.

The documentation is based on the so called DNSSEC-bis specifications that where finalized by the IETF DNSEXT working group in July 2004 [I-D.ietf-dnsext-dnssec-intro, I-D.ietf-dnsext-dnssec-protocol and I-D.ietf-dnsext-dnssec-records].

At the moment this paragraph is written (August 2004) the author is aware if the following open-source and or freeware implementations of the DNSSEC-bis specifications: BIND9.3.0beta and NSD2.1.1. All our examples are based on BIND9.3.0beta.

This document is not intended as an introduction into DNS. Basic knowledge of DNS and the acronyms used is assumed. We have tried to not to use to much jargon and if we do we have tried to explain the meaning.

This document will be subject to change. Please regularly check if new versions have occurred at http://www.ripe.net/disi. Your corrections and additions are appreciated.

Part I. Securing DNS data

This part deals with securing data in zone files. We describe how you generate and manage keys, how you set up a recursive nameserver to verify signed zone data and how you sign and serve zones.

Table of Contents

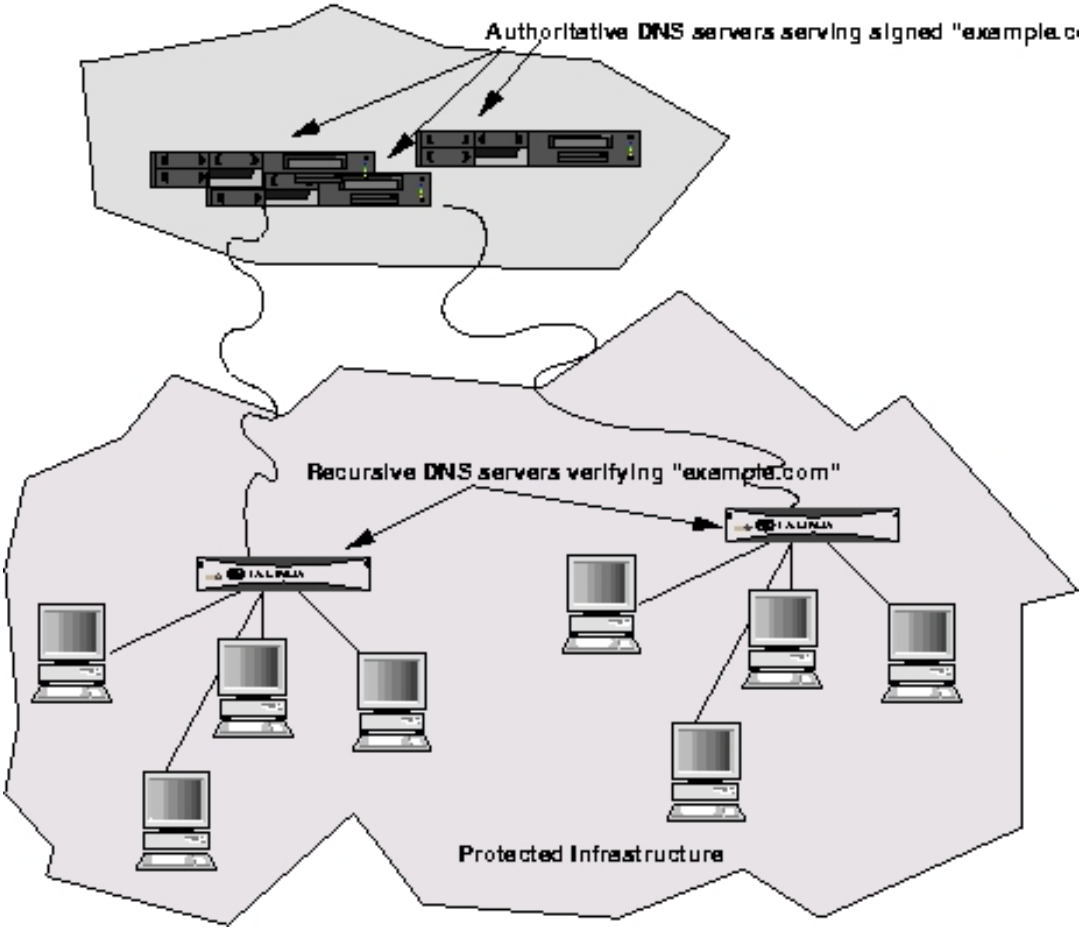
Chapter 1. Configuring a recursive nameserver to verify answers

Task at hand

We plan to configure a recursive nameserver to verify the data it receives. Users that use this recursive nameserver as their resolver will, in this way, only receive data that is either secure or unsecure. As a result spoofed data will not find it's way to the the users [1]. Having a verifying recursive nameserver protects all the users that use it as a forwarder against receiving spoofed DNS data.

In other words, with reference to Figure 1.1, “DNS environment”, we will configure the recursive DNS servers with a trusted key for "example.com" so that all the data served by the authoritative servers for "example.com" is verified before it is handed to the protected infrastructure that have the recursive servers configured as their forwarder (the nameservers that usually are assigned through DHCP or configured in `/etc/resolv.conf`).

Figure 1.1. DNS environment



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 corresponding private key. The zone acts as a secure entry point into the DNS tree and the key configured in the recursive

[1] Provided that the path between the verifying recursive nameserver and the stub resolver can be trusted. On a shared network such as an IEEE802.11 network this is not the case.

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.

We assume you have configured your nameserver to be recursive only.

We also assume that that another nameserver in your organization has been configured to run as an authoritative server for a secured zone called `example.net`. Notes on how to setup a secured zone can be found below in section Chapter 2, *Securing a DNS zone*

Warning

Configuring trust anchors means that your recursive nameserver will treat zones for which you configured trust anchors as being secured. If the zones for which you have configured trust anchors change their keys you will also have to reconfigure your trust anchors. Failure to do so will result in the data in these zones being marked as bad and therefore becoming invisible to your users.

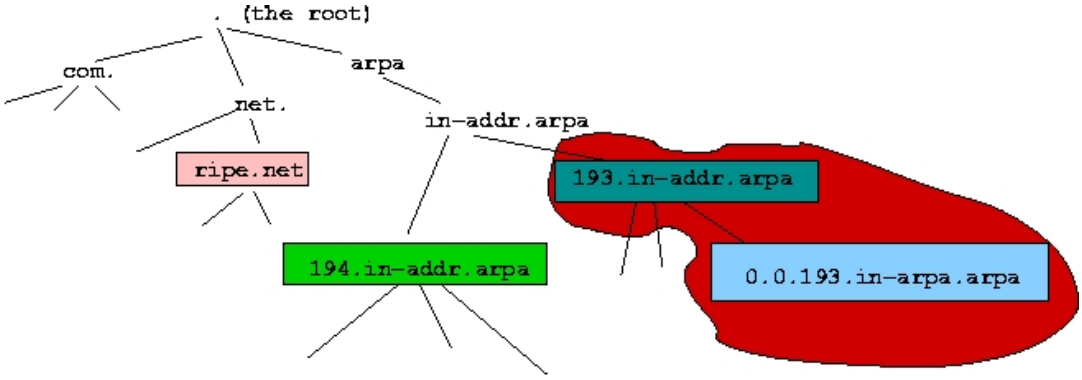
Configuring the caching forwarder

See Appendix A, *BIND installation* for information on compiling BIND with the correct switches to allow for DNSSEC. Do not forget enter the `dnssec-enable` statement in the `options` directive of your `named.conf`.

Configuring a Trust Anchor

A trust anchor is a public key that is configured as the entry point for a chain of authority. In the ideal case validating nameservers would only need one of these trust-anchors configured. But during early deployment you will probably need to configure multiple trust anchors.

Figure 1.2. Trust Anchors in the DNS tree



In Figure 1.2, “Trust Anchors in the DNS tree” we show a zone tree. In this tree the domains `ripe.net`, `194.in-addr.arpa` `193.in-addr.arpa` and `0.0.193.in-addr.arpa` are assumed to be signed. It is also assumed there is a secure delegation between `193.in-addr.arpa` and `0.0.193.in-addr.arpa`. In order to verify all these domains the validating DNS client would have to configure trust anchors for `ripe.net`, `194.in-addr.arpa` `193.in-addr.arpa`.[2].

To configure a trust-anchor you have to get hold of the public key of the zone that you want

[2]Fetching and maintaining all these trust-anchors in the absence of a signed root clearly does not scale and is one of the problems that early deployers will have to deal with. BIND9 contains a so called "look-aside" validation option that may help with this trust-anchor distribution issue. Configuring look aside validation will be a topic of future versions of this document

to use as the start of the chain of authority that is followed when your data is validated. It is possible to get these straight from from the DNS, but there are two reasons why this may not be a brilliant idea.

First you have to establish the authenticity of the key you are about to configure as your trust-anchor. How you do this depends on on what method the zone owner has made available for out of band verification of the key. You could do this by visiting the zone-owners secured web site and verify the key material there; you can give the zone-owner a call if you personally know her; you may believe the key that is published on the bill you just received from the zone owner; or, you may just believe that your OS did the verification on your behalf.

Second, and this is a little more subtle, is that you may have a choice of public keys to choose from in which case you would like to choose the the proper "Secure Entry Point" key.

In DNSSEC a difference is made key- and zone-signing keys. Key-signing keys exclusively sign the DNSKEY RR set at the apex while zone-signing keys sign all RR sets in a zone. Key-signing keys are often used as Secure Entry Points (SEP) keys. These SEP keys are the keys that are intended to be used first when building a chain of authority from a trust anchor to signed data. We assume and advice a one-to-one mapping between SEP keys and key-signing keys. Vince in practice key-signing keys have a lower rollover frequency than zone signing keys you should configure the SEP i.e. key-signing keys.

In addition to having the proper public key you should be aware of the rollover policy of the zone owner or that you have a tool that takes care of automated rollover. Failure to timely modify the trust-anchor if the corresponding SEP key is rolled will result in verification failures.

Assume you have obtained the key-signing key of `example.net.` to configure that key as a trust anchor you will have have to include the following statement in the `named.conf` of the recursive nameserver.

Figure 1.3. Trust-anchor configuration

```
trusted-keys {
"example.net." 257 3 5 "AQO8VL6u4R3BopupRb9p0Nsns2Sy3+YDlu//TG+zWRJ+ppbhTGbhxtI8
                  1vbDNeFnnGHdP6TWimibhwnqaG5D8XVoMRk5A2E/al8/8DmyHu2fqtTt
                  y0MZHHZJDKCUep+nJnQLxESdbFhHKmBZEzN9Lb3clKcnHSXDWP2qYP4S
                  cqX4uQ==";
};
```

The format is similar to the DNSKEY RR except that the "DNSKEY" label, the CLASS and the TTL are omitted and quotes are placed around the name and the public key material.

Testing

As soon as a `trusted-key` has been configured, data from that zone or its sub zones will be verified by the caching forwarder. You can test this by querying your server. If data is verified by the caching forwarder the `ad`-bit will be set by the nameserver (see the 'flags' in the following example).

```
$dig  @10.0.53.204 example.net SOA +retry=1 +dnssec +multiline

; <<>> DiG 9.3.0beta3 <<>> @10.0.53.204 example.net SOA +retry=1 +dnssec +multiline
;; global options:  printcmd
;; Got answer:
;; ->>HEADER&lt;&lt;- opcode: QUERY, status: NOERROR, id: 50414
```

```
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;example.net.          IN SOA

;; ANSWER SECTION:
example.net.          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)
                                 )
example.net.          100 IN RRSIG SOA 5 2 100 20040528103254 (
                                 20040428103254 14804 example.net.
                                 GMdIREMWV+LMuoDZvoVKyUobeEdeXTqzdV0MAUB9VSf7
                                 gT+dOBA2tMYSMG9OnVuJcbkBrzVLq56CSGgGQMEEt8/q
                                 8auCpFYiXiFri+9LitVKK+n3UvBIb6AL+/acyhUmUzbq
                                 5mF1nuWUDiuIuv/fXFGIeS9V/6P7+ufXWqedhVY= )

;; ADDITIONAL SECTION:
example.net.          100 IN DNSKEY 257 3 5 (
                                 AQO8VL6u4R3BopupRb9p0Nsns2Sy3+YDlu//TG+zWRJ+
                                 ppbhTGbhxtI81vbDNeFnnGHdP6TWimibhwnqaG5D8XVo
                                 MRk5A2E/al8/8DmyHu2fqtTty0MZHHZJDKCUep+nJnQL
                                 xESdbFhHKmBZEzN9Lb3clKcnHSXDWP2qYP4ScqX4uQ==
                                 ) ; key id = 64431
example.net.          100 IN DNSKEY 256 3 5 (
                                 AQP21jqKufd1oMcZTWfttfJJvzmMPSfSmF+s9vt+Jcw1
                                 yHYuuOoiIAyEeWNEL7ZjSRbcLsIc2/LHN4eZuOfjmedR
                                 yh4lEzNJyrUuvz/opbwCOcKNZPTY9gnBkafS02RXihWX
                                 Ae9zr5OUaygfCqxCtCvhc2LGjXpUu7j6e8b3D0VadQ==
                                 ) ; key id = 14804
example.net.          100 IN RRSIG DNSKEY 5 2 100 20040528103254 (
                                 20040428103254 14804 example.net.
                                 PNyi2K7psI5m4lX6SFbW1I2Lgm4Op3xeNxZtnNii+j2I
                                 +yHJPk2LVSwHWxSaGDI5OX6OnviFeSk0mbUdb2j7b71s
                                 xYkY+HJ0fIXFtQURandQ0ykuF1tTOyosI+rXJygJKjM5
                                 0RftBaOXALTc3c8RhZSwK/gNEii+H68mcok6B1c= )
example.net.          100 IN RRSIG DNSKEY 5 2 100 20040528103254 (
                                 20040428103254 64431 example.net.
                                 FkIYoNBonngDZOPiiDg0Prx32r52NdXRLIkHBFwQHcbC
                                 aNf9DcEsXFfLMYTfsPeU4I5w1LCzo4MsnpwAB+MD20GW
                                 0YTN8zPVzO/FVWWTbEgxzk0Zip5c/B8IFNtTgmNL9peF
                                 BcoOhl7kE8nEO653SQGpGXPHj7gw+gCtwjSfWac= )

;; Query time: 783 msec
;; SERVER: 10.0.53.204#53(10.0.53.204)
;; WHEN: Wed May 12 11:41:06 2004
;; MSG SIZE  rcvd: 890
```

It is important that you check on the proper working of the verifier. This can be done by using the BIND log facilities on the machine that is configured as the verifying recursive nameserver.

In BIND messages of a certain category can be logged to separate channels. The channels determine where the messages go and to what severity level they will need to be reported. The relevant category for DNSSEC verification is `dnssec`. In the example below the errors of the `dnssec` category are directed to the `dnssec_log` channel. In order to follow the verification process the channel has to log at least at `severity debug 3`.

```
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 log file will look similar to the output below. The attempt for positive response validation shows how the verifier 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.

```
12-May-2004 11:41:06.815 dnssec: debug 3: validating example.net SOA: starting
12-May-2004 11:41:06.816 dnssec: debug 3: validating example.net SOA:
   attempting positive response validation
12-May-2004 11:41:06.824 dnssec: debug 3: validating example.net
   DNSKEY: starting
12-May-2004 11:41:06.825 dnssec: debug 3: validating example.net
   DNSKEY: attempting positive response validation
12-May-2004 11:41:06.941 dnssec: debug 3: validating example.net
   DNSKEY: verify rdataset: success
12-May-2004 11:41:06.942 dnssec: debug 3: validating example.net
   DNSKEY: signed by trusted key; marking as secure
12-May-2004 11:41:06.942 dnssec: debug 3: validator @0x828e00:
   dns_validator_destroy
12-May-2004 11:41:06.943 dnssec: debug 3: validating example.net SOA:
   in fetch_callback_validator
12-May-2004 11:41:06.943 dnssec: debug 3: validating example.net SOA:
   keyset with trust 7
12-May-2004 11:41:06.943 dnssec: debug 3: validating example.net SOA:
   resuming validate
12-May-2004 11:41:06.945 dnssec: debug 3: validating example.net SOA:
   verify rdataset: success
12-May-2004 11:41:06.945 dnssec: debug 3: validating example.net SOA:
   marking as secure
12-May-2004 11:41:06.945 dnssec: debug 3: validator @0x823e00:
   dns_validator_destroy
```

Chapter 2. Securing a DNS zone

Task at hand

If a zone has been signed and its key has been configured in a validating recursive nameserver we usually refer to it as being an "island of security". It apparently does not have a secured parent and stands alone in the sea of unsecured other domains. Usually creating a island of security is the first step in becoming part of the secure DNS. The island of security will remain "insecure" for resolvers that have no trust-anchor configured for the domain.

If a zone owner decides to create an island of security she will sign her zones and distribute the "secure entry points" to the system administrators that want to validate her zone data. Once the island of security has been set up the island can become part of the secure tree by exchanging the "secure entry point" with the parent.

This is what we will have to do. After creation of the key-pairs used for signing and verification we want to sign the zone data for our own organization (say `example.net.`) and configure the caching forwarders on our organizations network to verify data against the public key of our organization.

Note that in the text below we assume 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 Chapter 3, *Delegating of Signing authority; becoming globally secure*.

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 Figure 2.2, “example.net example zone”.

Creating key pairs

Key Maintenance Policy

Before you are going to generate keys you will need to think a bit on your key maintenance policy. Such policy should address

- What will be the sizes of your keys.
- Will you separate the key- and zone-signing keys (See below).
- What key size will you use.
- How often will you roll the keys.
- How will system administrators that intend to use your zone as trust anchor get hold of the appropriate public key and what mechanism will you provide to enable them to verify the authenticity of your public key.
- How will you signal a key rollover, or how can you make sure that all interested parties are aware of a rollover.

Some of these issues may be trivial to address. For instance, your organisation may have established mechanisms to distribute the public keys,there may be obvious way to publish

an upcoming rollover such as the possibility of publishing the event in a corporate newspaper. Alternatively it may be possible to notify all relevant parties by a mail when a corporate X.509 hierarchy is available for e-mail verification.

A few words on key- and zone-signing keys.

It seems good practice to use zone-signing keys and key signing keys (also see Chapter 4, *Rolling Keys*). The key signing keys are usually the first keys from you zone that are used to build a chain of authority to the data that needs to be validated. Therefore these keys are often called a secure entry point key (or SEP key). These SEP keys are the ones that you should exchange with your parents or that verifying resolvers configure as their trust anchors.

Throughout this document we assume that you use separate key- and zone-signing and that the key-signing keys are exclusively used as secure entry point keys and can be identified by the SEP flag [rfc3757].

Creating the keys

Figure 2.1. dnssec-keygen arguments

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

Version: 9.3.0rc3
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 | OTHER
    name: owner of the key
Other options:
    -c <class> (default: IN)
    -e use large exponent (RSAMD5/RSASHA1 only)
    -f keyflag: KSK
    -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>
    -k : generate a TYPE=KEY key
Output:
     K<name>+<alg>+<id>.key, K<name>+<alg>+<id>.private
```

dnssec-keygen is the tool that we use to generate keys pairs. The arguments that we have to provide **dnssec-keygen** are shown in Figure 2.1, “ 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 RRSIG 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 zone signing key pair for `example.net`:

```
# dnssec-keygen -r/dev/random  -a RSASHA1 -b 1024 -n ZONE example.net
Kexample.net.+005+25721
```

Because of the considerations in the section called “Key Maintenance Policy” you will also need to create SEP keys. To create keys with the SEP bit set by specifying the `-f KSK` flag with **dnssec-keygen**.

```
dnssec-keygen -r/dev/random  -f KSK -a RSASHA1 -b 1024 -n ZONE example.net
```

Lets have a look at the content of these file[3]:

```
cat  Kexample.net.+005+25721.key
example.net. IN DNSKEY 256 3 5   (
     AQPcvjxkh0iOC2MHE5vLmzuEuMERJ5PzQMZswbRD5oVyOmFZJTI6n5aM
     m7dKiv2//JxoxrA2dFjpDUcDafwEK4CE1oGiGe1uzYZvVIwxVc0xfZZg
     QwXSWnRIqp3owrTFUadp74cw2ZfFdPyDRm01kIqfwzVptbuLLQrw/AGM
     NBl2vQ== )
```

The public key (`.key` extension) is exactly as it would appear in your zone file. Note that the TTL value is not specified. This key has a "flag" value of 256. Since this value is even the key is not marked as a SEP key and should be used for zone signing.

The private key (`.private>` extension) contains all the parameters that make an RSASHA1 private key. The private key of a RSA key contains different parameters then DSA. Here is the private key (with base64 material truncated):

```
# cat Kexample.net.+005+25721.private
Private-key-format: v1.2
Algorithm: 5 (RSASHA1)
Modulus: 3L48ZIdIjgtjBxOby5s7hLjBE...
PublicExponent: Aw==
PrivateExponent: kyl9mFowXrJCBLe9M...
Prime1: 87j0EvXaiMPPC4g34CD1obNidJ...
Prime2: 59zwwY67tUGCdqWr3QtSM3NasW...
Exponent1: ontNYfk8Wy00slrP6sCjwSJ...
Exponent2: mpNLK7R9I4EBpG5yk1zhd6I...
Coefficient: lkLl54jAMUVDGRTqy/HcH...
```

This private key should be kept secure [4] i.e. the file permissions should be set so that the zone administrator will be able to access them when a zone needs to be signed. Besides, the BIND tools will, by default, [5] look for the keys in the directory where signing is performed, and that might not be the most secure place on your OS.

Zone signing

[3] We slightly edited the output for readability. We printed the base64 material over several lines and introduced the brackets so that this is a legal multiline representation of a RR.

[4] 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.

[5] It is possible to fully specify the path to the keys.

Once you created key pairs you should include them in your zone file. Refer to the example in Figure 2.2, “example.net example zone” where we use the `$include` directive to include the keys. Of course, we do not forget to increase the serial number in the SOA record before signing.

In the example below we will use the DSA key as the key signing key and the RSA keys as zone signing keys.

Figure 2.2. example.net example zone

```
$TTL 100
$ORIGIN example.net.
@                       100     IN      SOA     ns.example.net. (
                                                olaf.ripe.net.
                                                2004071901
                                                100
                                                200
                                                604800
                                                100
                                                )


                NS ns.example.net
                NS ns2.example.net
ns          A         193.0.2.1
ns2          A 193.0.2.2


; include the public keys.
$include Kexample.net.+005+25721.key   ; ZSK inserted 20040719
$include Kexample.net.+005+37062.key   ; KSK inserted 20040719
```

Once the key is included in the zone file we are ready to sign the zone using the **dnssec-signzone** tool (see Figure 2.4, “ dnssec-signzone arguments” for all the arguments). We use the `-o` flag to specify the origin of the zone; by default the origin is deduced from the zone file's name.

With the '`-k key_name`' we specify which key is to be used as the Key signing key. That key will only sign the DNSKEY 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. If you do not specify the keys BIND will use those keys for which the public keys are included in the zone and use the SEP flag to distinguish between key signing and zone signing keys.

In practice you would not want to rely on the default, since in key rollover scenarios you will have a public key in your zone file but you would not want to use that for zone signing (in order to avoid double signatures and therefore longer signature generation times and more resource consumption on your nameserver). Below is the command issued to sign a zone with the 37062 key as key signing key and the 25721 keys as zone signing key.

```
/usr/local/sbin/dnssec-signzone  \
   -r /dev/random \
   -o example.net  \
   -k Kexample.net.+005+37062 \
   db.example.net \
   Kexample.net.+005+25721.key
```

The signed zone file is reproduced in figure Figure 2.3, “Example of a signed zone”. Note that the apex DNSKEY RRset is the only RRset with 2 signatures, made with the zone and key signing keys. The other RRsets are only signed with the zone signing keys.

Figure 2.3. Example of a signed zone

```
; File written on Mon Jul 19 13:26:01 2004
; dnssec_signzone version 9.3.0beta3
example.net.         100 IN SOA ns.example.net. olaf.ripe.net. (
                                2004071901 ; serial
                                100        ; refresh (1 minute 40 seconds)
                                200        ; retry (3 minutes 20 seconds)
                                604800     ; expire (1 week)
                                100        ; minimum (1 minute 40 seconds)
                                )
                         100 RRSIG SOA 5 2 100 20040818102601 (
                                20040719102601 25721 example.net.
                                cN74KZQeipMr+yMPwdTs355k95GOhHSq9GHR
                                KCwHZoqPwYNjluuehRffPR6jiZaKVOrnGALh
                                R2D6ghney3bqdanFhzENo/fhaN0Vr8MsBart
                                fpuZrVwiCtXIHekjZSmE7mVOju43DPbpQ7rA
                                rGq3XB7oYnlD9jbcnjj39pUdkEs= )
                         100 NS ns.example.net.example.net.
                         100 NS ns2.example.net.example.net.
                         100 RRSIG NS 5 2 100 20040818102601 (
                                20040719102601 25721 example.net.
                                Y/OEpLuAU7zZSzhDqwoXyzBWnLcD0ibq+ju1
                                vQAYAcocSJed0EmtOBdbom3Icw+tIVYotHC0
                                D34ilf5FIJDldy3aQBgWXA+eT7o2NujYtZy7
                                +JHIWW1feyDRRZJshocQ47rS63P1DqR1Pb+J
                                u6cfI6J/0dwm07rrKlzD7fFmMyI= )
                         100 NSEC ns.example.net. NS SOA RRSIG NSEC DNSKEY
                         100 RRSIG NSEC 5 2 100 20040818102601 (
                                20040719102601 25721 example.net.
                                ItbZrp4/ICR5IzBVMUTmTrcdhvsxUORdSL+u
                                jAdaWjk3iBsM0SF9EH8M3UeVHjvNCJcDHiJr
                                ZfEHFpRUhqE/uuV6JMkdaq3oPU8BKnLFRgxu
                                BhgrczSe9lPIx+YOBNoEpRZdaIZee1aKND5C
                                By+/pCZ0tXjBeQZ3rxZT11OAEA8= )
                         100 DNSKEY 256 3 5 (
                                AQPcvjxkh0iOC2MHE5vLmzuEuMERJ5PzQMZs
                                wbRD5oVyOmFZJTI6n5aMm7dKiv2//JxoxrA2
                                dFjpDUcDafwEK4CE1oGiGe1uzYZvVIwxVc0x
                                fZZgQwXSWnRIqp3owrTFUadp74cw2ZfFdPyD
                                Rm01kIqfwzVptbuLLQrw/AGMNBl2vQ==
                                ) ; key id = 25721
                         100 DNSKEY 257 3 5 (
                                AQPOkuCvnQPxTBXdd903yIPZlvAJ5nsFt09R
                                naIJME0K2l6ebuFKRf/9Npb+1PQ/aMzey8HX
                                3WI5BJ0jqajpvOmh3J6EtflIetoSvf8yd9ls
                                yw8oxFLrA4IhpG1x3Pn1A4rrPfJhNTED7ZO7
                                iQUGjcIar3Vnt/PqVF1mN6qRWNWhsQ==
                                ) ; key id = 37062
                         100 RRSIG DNSKEY 5 2 100 20040818102601 (
                                20040719102601 25721 example.net.
                                cK0aLkPtUlVVrieOO7Wp1msPOJEvk/C6MKiY
                                Qx1QdEtUtDXs9IlSOqTuYjqJwceoQYwdXfEu
                                72NJKgbYwtjhdbAF1AmkuCbRdP9f0Wdnnoem
                                P2mR5Fl1Trdz3OCK8y1UYIDoUCa8+w7QYEZu
                                grZlRe9RGIKnFLAv4zBZQJE1ZV8= )
                         100 RRSIG DNSKEY 5 2 100 20040818102601 (
                                20040719102601 37062 example.net.
                                gQyCtOIzDB6LMKsMQ4Hu0+vkP7OdxyO4HuDW
                                VbXlkyZXFQbt7U2Foy+oq24M8LJTowZ3Kssm
                                +8cxnii7fGiiwn3MUlvzsQx+CrNRP54DMDKS
                                sZ04X4BjHEziO8yTob7+4l5BN4RsMtlT3DkL
                                R28dDzetmtTqA5XVVvWtWdNIfWo= )
ns.example.net.          100 IN A 193.0.2.1
                         100 RRSIG A 5 3 100 20040818102601 (
                                20040719102601 25721 example.net.
                                E4zCBxCEhCXW7cPrNJgKwjze7PQX9p89gxk6
                                ZHLE82STajzjPwEhpJ/C245/uB7pViubcEPQ
                                qbZwU+v/plsD3g9Px4uydkTpT9p19SDTSAf2
                                b6qNEq7N4d/oXvUS5FUDKSMTkIpSz9rNC1+t
                                AMo+tfSdmPd1IUPkdZSotkiJhZE= )
                         100 NSEC ns2.example.net. A RRSIG NSEC
                         100 RRSIG NSEC 5 3 100 20040818102601 (
                                20040719102601 25721 example.net.
                                GIIhvpUkXV/5Nni6OCwh7Q/mz691BhE9uKeQ
                                8ckAS3B0AfAlXZQLIJvEjs9ovXqosdF0Q2dj
                                wOWCUdlmdfpMeelY+OPwjBvlZW9pppIlskQe
                                //H25VAqciA1dlpxiRtv6sL2RVwBR0ipLz6T
                                7gzfb+TSOC2f+mHaU8RBRY2MUIg= )
ns2.example.net. 100 IN A 193.0.2.2
                         100 RRSIG A 5 3 100 20040818102601 (
```

```
                                20040719102601 25721 example.net.
                                tXKfDO3yjYaHOHEUUuYXtQzyIv8bsmK6tuRE
                                jarIoUZPuPQlfuYnSdpPEbsxoey997AiWHAz
                                rocfpxFT1eqeblEmn8T8F/bGF4uU8s8K2wDk
                                RM5LKVPk0bXLqd4pIleJjRJ2dO2IMnav9VEQ
                                m8G/iQG0KMIz4jp3IjJGjkmWVQ0= )
                        100 NSEC example.net. A RRSIG NSEC
                        100 RRSIG NSEC 5 3 100 20040818102601 (
                                20040719102601 25721 example.net.
                                dYkpY/cVxRsyKsJfbKyNyDd+CCG2LVYFjG8Z
                                rgRgEFlqHcPNdQ+J7TCSKQ+T6CQM0tRLDfMo
                                Z5rEf7LVjw+7G4qvdfnldZRkbrHCfPZDlO+l
                                KshMPJjXh+MlmazD5gOyGuimZ2M2JA4a8qbT
                                qaJhtxoKjLo387Zksb2AT36NuQQ= )
```

The signing process did the following:

- It sorted the zone in 'canonical' order.
- Inserted NSEC records for every label.
- Added the key-id as a comment to each DNSKEY-record.
- Signed the DNSKEY RR set with 2 keys; the key signing key and the zone signing key.
- Signed the other RRs with the one key key, the zone-signing key.

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

The signed zone is stored in `db.example.net.signed`, make sure you have configured **named** to use this file to serve the zones from.

Figure 2.4. dnssec-signzone arguments

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

Version: 9.3.0rc3
Options: (default value in parenthesis)
        -c class (IN)
        -d directory
                directory to find keyset files (.)
        -g: generate DS records from keyset files
        -s YYYYMMDDHHMMSS|+offset:
                RRSIG start time - absolute|offset (now - 1 hour)
        -e YYYYMMDDHHMMSS|+offset|"now"+offset]:
                RRSIG 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
        -l lookasidezone
        -z: ignore KSK flag in DNSKEYs
Signing Keys: (default: all zone keys that have private keys)
        keyfile (Kname+alg+tag)
```

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 recursive nameservers. Just configure your recursive nameserver with the public SEP key you just generated for the zone. How to do this is described in the section called “Configuring the caching forwarder” above, also see Figure 1.3, “Trust-anchor configuration”.

Zone Resigning

When the signatures in your zone are almost expired or when you have added a few records to your zone you will have to resign your zone. There are two ways to resign your zone data. You may choose whichever option depending on your level of automation, the size of your zone and the frequency with wich you have to generate SIG RRs.

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

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

- You can add the new records to the already signed zone file and then run that zone file through the signer. The (BIND) signer will insert the new records and associate NSECs 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 (also see ???. Please also make your tools publicly available.

Troubleshooting Signed Zones

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 to parse the configuration and zone files as **named** itself but only test on syntax.

One can use **dig** and a nameserver configured with a `trusted-key` to verify ones 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[6]:

```
; <<>> DiG 9.3.0s20020722 <<>> +dnssec @verifier corrupt.example.net
;; 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.example.net.               IN      A

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

Note that a caching forwarder will not do cryptographic verification of the zones it is author-

[6] We corrupted the data by modifying rdata in the signed zone-file to generate this example

itative for. So if your caching forwarder is a primary or secondary server for a particular zone you will always get an answer since it is assumed that data from disk is secure.

Further troubleshooting needs to be done on a a server configured as a validating recursive nameserver. 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
debug 3: validating corrupt.example.net A: starting
debug 3: validating corrupt.example.net A: attempting positive response validation
debug 3: validating example.net DNSKEY: starting
debug 3: validating example.net DNSKEY: attempting positive response validation
debug 3: validating example.net DNSKEY: verify rdataset: success
debug 3: validating example.net DNSKEY: signed by trusted key; marking as secure
debug 3: validator @0x81e9800: dns_validator_destroy
debug 3: validating corrupt.example.net A: in fetch_callback_validator
debug 3: validating corrupt.example.net A: keyset with trust 7
debug 3: validating corrupt.example.net A: resuming validate
debug 3: validating corrupt.example.net A: verify rdataset: RRSIG failed to verify
debug 3: validating corrupt.example.net A: failed to verify rdataset
debug 3: validating corrupt.example.net A: verify failure: RRSIG failed to verify
debug 3: validating corrupt.example.net A: keyset with trust 7
debug 3: validating corrupt.example.net A: verify rdataset: RRSIG failed to verify
debug 3: validating corrupt.example.net A: failed to verify rdataset
debug 3: validating corrupt.example.net A: verify failure: RRSIG failed to verify
info: validating corrupt.example.net A: no valid signature found
debug 3: validator @0x81e0800: dns_validator_destroy
```

(This output was generated by using the `dnssec` category to a logging channel with `severity debug 3;` configured.)

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. This may cause served by some of the authoritative servers to time out so 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-key**s. 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 sectionChapter 4, *Rolling Keys*.

Slave server problems

Slave servers will need to run code that is DNSSEC en-

abled if one of the authoritative servers for a zone is not DNSSEC aware problems may arise for the DNS client that happen to try to fetch data from those DNSSEC oblivious servers.

Related is that the load on all your nameservers will increase. Zone files, Memory and bandwidth consumption will grow. Factors 2-5 are not uncommon see Appendix B, *Hints and tips* for some numbers.

Chapter 3. Delegating of Signing authority; becoming globally secure

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

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. I-D.ietf-dnsext-dnssec-intro describes how this can be established by having the parent generate a signature over the DS record that is generated from the childs DNSKEY.

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 Chapter 2, *Securing a DNS zone*, but, since keysets are used, some details are different.

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 signing keys, or more precisely the SEP keys, generated by the child, need to be exchanged with the parent to get the DS records generated from. To ease the process BIND introduces keysets.

A keyset is a small file, with the same syntax as a zone file, that contains one or more key signing keys of the child. Keyset files are automatically generated when signing a zone with **dnssec-zonesign**.

The sequence of events will be:

i. The child uploads it's key signing key to the parent.

   The details of this procedure depend on the parents key exchange policy normally the contend of the `keyset` file will need to be uploaded.

ii. The parent stores the keyset file in the directory with the parent zone. For each delegation there will be one keyset file.

iii. The **dnssec-signzone** command finds the keysets and automatically generates and signs DS records.

Alternatively the parent generates DS records and puts them in the zone file directly.

We will describe the steps in more detail below.

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 the section called “Creating key pairs” and signed our zones as described in the section called “Zone signing”.

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

The easiest way to upload the key is to 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 DNSKEY is established. The zone administrator[7] 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` is found containing one or more DNSKEY RRs for the `child-domain`. Note that although the keyset generated by the child contains signatures the RRSIG 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 invoke the command:

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

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 recommend the use of RSA/SHA1 keys only.
Parent indicating child security	It is important that the DNSKEY 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. As a parent you should always verify that the child publishes signed DNSKEY before including a DS RR.

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

Chapter 4. Rolling Keys

A rollover 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 rollover.

The rollover is defined by the moment keys generated with the "new" private key are first introduced into the zone. The key pair may have been generated well in advance and the public key may have been made public well in advance too.

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.

There are two types of scheduled key rollovers. The rollovers of key-signing keys and the rollovers of zone-signing keys. Key-singing keys are the first keys from your zone used by a verifier when building a chain of authority hence these keys are also referred to as secure entry point keys or SEP keys.

Although the DNSSEC protocol does not make a distinction between zone- and key-signing keys we strongly advice to make this distinction as it provides a clear separation between the keys that can be rolled without external interaction (the zone-signing keys) and the keys that need external interaction (the key-signing keys). You should use the `-f KSK` flag with **dnssec-keygen** when creating key-signing keys so that you can always make a distinction between key- and zone-signing keys by looking at the so-called flag field in the DNSKEY resource record. Its flag-field will be odd (257 mostly) when you deal with a key-signing , or SEP, key.

DNS traversal

Whenever data in a zone file is replace by other data it will need to propagate through the DNS before DNS clients actually see the new data. In a non-DNSSEC environment this may hardly ever be noticed but when operating DNSSEC allowing data to traverse through the DNS is critical.

DNS data with its associated signatures and the public key with which this data is verified travel through the DNS independently. This also implies that the public keys and the signatures are independently cached and therefore expire from caches at different times. As a consequence it can happen that an RRSIG is verified with a DNSKEY from a cache and that the RRSIG and DNSKEY come from different versions of the zone i.e. the public key relates to a key that is older than the signature. The reverse, where the signatures are older than the public keys that are used for verification can off course also happen.

As a zone administrator you have to be aware of this behavior and take into account that your signatures will need to verify with any future or previous version of your keyset. I-D.ietf-dnsop-dnssec-operational-practices describes the details which differ for zone-signing and keys-signing key rollovers. There are two approaches for this. The "pre-publish" and the "double signature" rollover.

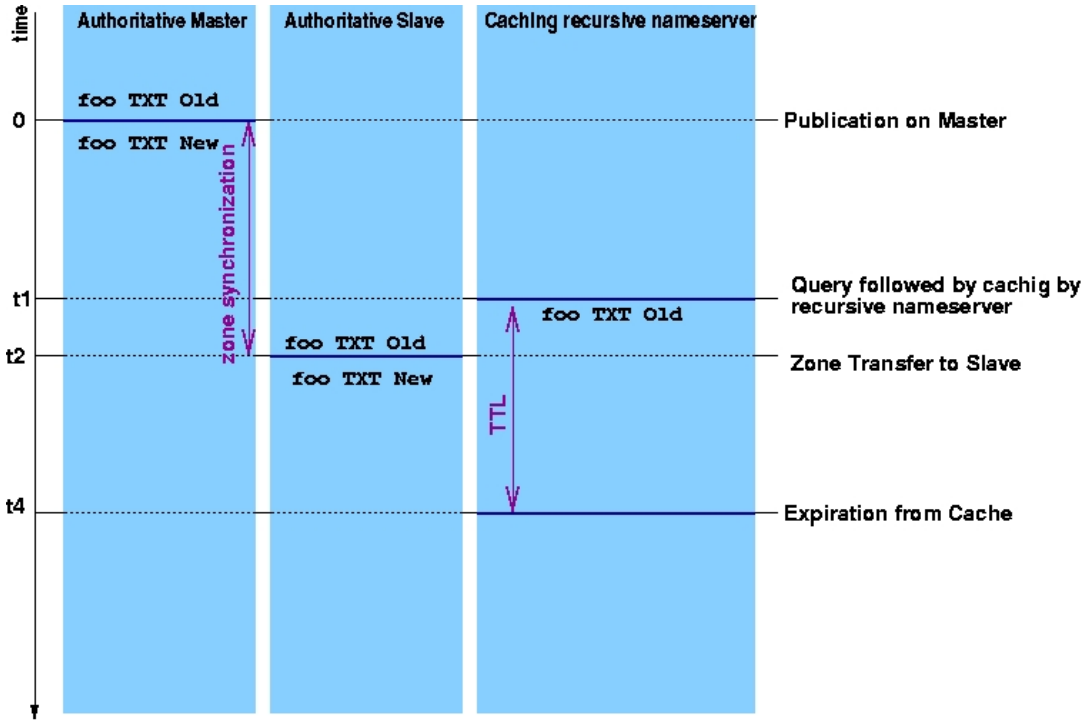
First let us take a closer look on how data traverses through the DNS. See Figure 4.1, “DNS Data Propagation” for reference.

At `t0` new data replaces data that was living in a previous version of the zone file. The data is published on the authoritative master or primary server. It will take some time, which we refer to as zone synchronization time, before the new version of the zone is picked up by all authoritative servers. In the worst case scenario a change a slave server will not be able to reach the master server and the zone will expire. So the maximum value of the zone synchronization time will be the value of the SOA expiration parameter.

Assume that at some time (t1) between publication of the new zone on the master server(t0) and the time the new zone is picked up by a slave server (t2) a query for the data is done by a recursive caching nameserver. Than that recursive server will return the *old* data to any of its clients for the time that is set by the TTL value on the *old* RRset. Only after t4 the recursive server will go back and query for new data and pick up the new records.

Note that the t4 does not only depend on t1+TTL but will be upper bound by the signature expiration time of the signature over the old RRset.

Figure 4.1. DNS Data Propagation



"Pre Publish" and "Double Signature" rollovers

During a pre publish rollover the public key is introduced in the DNSKEY RRset well before RRSIGs are made with the private part of this key. The "new" public keys are than available in caches when the RRSIGs over the data show up on the authoritative nameservers and caching nameservers can used cached DNSKEY RRs to verify the new data.

During a double signature rollover the new key pair is introduced and signatures are generated with both the new and the old key while both public keys are published in the DNS. After the period it takes to have this data propagate through the DNS the old key is remove and one only signs and publishes the new key.

Tools

To properly maintain state you will need an operational notebook. As for each of your zone there will be multiple KSKs and ZSKs and these keys all have a 'state' the situation may become very confusing. Below we give an overview of the operations using an "operational notebook". At the RIPE NCC we have developed a tool that takes the role of the "operation-

al notebook" and that links to the signing operations. This tool will be available shortly[8].

Zone-Signing key rollovers

During a Zone-Signing key (ZSK) rollover we use a "pre-publish" scheme.

Preparation (production phase)

We use the trivial `example.com` zone (Figure 4.2, “Trivial example.com”) as an example. The zone is stored in `db.example.com`.

Figure 4.2. Trivial example.com

```
$TTL 100
$ORIGIN example.com.
@                       100     IN      SOA     ns.example.com. (
                                                olaf.ripe.net.
                                                2004071801
                                                100
                                                200
                                                604800
                                                100
                                                )


                NS ns.example.com
                NS ns2.example.com
ns          A         193.0.2.1
ns2          A 193.0.2.2
```

Assuming that we first start to publish `example.com` we generate two ZSK keys and one KSK key.

```
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE example.com
Kexample.com.+005+63935
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE example.com
Kexample.com.+005+64700
dnssec-keygen -f KSK -a RSASHA1 -b 1024 -n ZONE example.com
Kexample.com.+005+54915
```

In our operational notebook we make a note that key 63935 will be used as the *active* and key 64700 as the *passive* ZSK. Both keys will be available through the keyset but only the active key is used for signing.

After we generated the keys we include them in the zone by adding the following include statements to `db.example.com`

```
;; ZSKs
$include Kexample.com.+005+63935.key
$include Kexample.com.+005+64700.key

;; KSKs
$include Kexample.com.+005+54915.key
```

Then we proceed by signing the zone. Since we do not want to use **dnssec-signzone**'s default behavior, which is to use all available keys for signing we have to fully specify which keys to use on the command line. since we will have to do this often the operational notebook will come in handy.

[8] July 22,2004: Contact olaf@ripe.net if you want a beta version of the tool

```
dnssec-signzone -k  Kexample.com.+005+54915.key -o example.com db.example.com Kexample.com.+005+639
```

Note that we supplied the KSK as an argument with the `-k` switch and only the active key ZSK as a signing key.

The Rollover (phase1)

Note down the signature expiration over the DNSKEY RR as is it is available in the DNS now. This value you can use as an upper limit for the duration of this phase. It is the value of `t4` in Figure 4.1, “DNS Data Propagation”. In the DNSKEY RR set below the signature expiration time is August 21, 2004 around 11:35 UTC. If *all*[9] the TTLs in your zone are not higher than say 600 than you would not have to wait that long. You would have to wait until you see the new zone published in all authoritative servers and an additional 10 minutes. The signature expiration is really an upper bound.

```
 100     DNSKEY  256 3 5 (
                 AQPQyhg865V4zkFZN+FICLAZPWWaAf5I43pW
                 UcuOiejT92AVu0eHOkbH5YiHV97r+QjAdZ7K
                 W7W+bvbqKBR5P4QMVNm8zCs5Trb9OcOY0+bb
                 LYZG3aG69wUfF1pjvmFV5zUSRHCLMEzXb5NS
                 XdazgdhhuM07L2e2EfJGp5qijtRwpQ==
                 ) ; key id = 63935
100     DNSKEY  256 3 5 (
                 AQPWrMsW0sGSTD7iE9ou+s7886WeSLIq/l/J
                 CgqwAn7jlECGAAN6cSHV5jWvovcWFthapWdG
                 DpC1uL48AcWtVWkRABGjU8Q16CAy0EcZ+24V
                 4cul+VluBt1YjuNfUlye+k5V+lmkjXBQ3Qdf
                 E8/owjsdx9mTkeQC4qiFjUxWXTl4DQ==
                 ) ; key id = 64700
100     DNSKEY  257 3 5 (
                 AQPhZQ29Xg60NLgR+qdJENZpklU+WQF0abmp
                 Ni3CeOYyR+bd01Q/2WDI6BbWCLdIb9YflRaj
                 hmyb+AmzmjNzhw8VjcY9Sr2zIcG50ctuZ8Og
                 t7fcGrCbEM9fIDIKdDRlf+SY8OnGEMi6sI4m
                 bZ4zoh+nWfNrTxQR5hHv074uSAvZyQ==
                 ) ; key id = 54915
100     RRSIG   DNSKEY 5 2 100 20040821114554 (
                 20040722114554 54915 example.com.
                 gcnf3rf+D6izv9A//16u+Jx/LDVinLtcpkWR
                 yxDV5goS2SnoLfyEryqbSAyKbh4redyQCjSW
                 /HZXFBoPYrAy8fqaY1AfjVP+q9zJPvysUOp+
                 2T6mm8/9pcZoGXw1wPjPUAz+AF0oJnoaWo7t
                 764xvZc47kAI1pT0RTizV2BofcU= )
100     RRSIG   DNSKEY 5 2 100 20040821114554 (
                 20040722114554 63935 example.com.
                 T7gRcEZkxEl5iGJdCzSu47Og9ydMO5Uggvcz
                 A9jETiTUrBttyYua7qDZOjNrzT4GVZ6s/UBw
                 tbGCqyMU/sVvaulP4h8oerX44bw5eP/mluLY
                 T9rwm2jBI1rZSPDdGDp8lJ2vvrxASYSF2Fxg
```

At the moment of the rollover you have to make your current passive key (64700) active and your current active key (63935) passive also make a note that this key is to be removed from the keyset in the next phase of the rollover.

Increase the SOA serial number and resign the zone using the new active key.

```
dnssec-signzone -k  Kexample.com.+005+54915.key -o example.com db.example.com \
Kexample.com.+005+64700
```

Publish this zone in the DNS and make sure it is published long enough to propagate through the DNS.

Cleanup (phase2)

After the data has propagated though the DNS it you have to replace the passive ZSK (63935) by a new passive key.

Start with generating the passive ZSK.

[9] which includes the last parameter in the SOA which is used for negative caching

```
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE example.com
Kexample.com.+005+01844
```

Add the new passive key (01844) and remove the old passive key (63935) into the zone file.

```
;; ZSKs
$include Kexample.com.+005+64700.key
$include Kexample.com.+005+01844.key


;; KSKs
$include Kexample.com.+005+54915.key
```

Increase the S0A serial and resign using the same active key as in phase 1.

```
dnssec-signzone -k  Kexample.com.+005+54915.key -o example.com db.example.com \
Kexample.com.+005+64700
```

After you published your zone you are back into the "production" phase. You should not proceed into a new rollover until the current DNSKEY RRset had a change to propagate through the system

You can now delete the 63935 key. We suggest you move the key pair to a separate directory or make a backup.

Modifying zone data during a rollover

You can at any time modify zone data other than the data in the keyset. As long as you use the suitable active ZSK for signing.

Key-Signing key rollovers

During a Zone-Signing key (ZSK) rollover we use a "double signature" scheme.

Preparation (production phase)

We again use the trivial `example.com` zone (Figure 4.2, “Trivial example.com”) as an example. The zone is stored in `db.example.com`. It contains a active and a passive ZSK (63935 and 64700 respectively) and a KSK (54915). The include statements are the same as the section called “Preparation (production phase)”:

```
;; ZSKs
$include Kexample.com.+005+63935.key
$include Kexample.com.+005+64700.key

;; KSKs
$include Kexample.com.+005+54915.key
```

and the command to sign the zone is the same too.

```
dnssec-signzone -k  Kexample.com.+005+54915.key -o example.com db.example.com Kexample.com.+005+639
```

The rollover (phase 1)

We start the rollover by generating a new KSK

```
dnssec-keygen -f KSK -a RSASHA1 -b 1024 -n ZONE example.com
Kexample.com.+005+06456
```

Include the new KSK into the zone file:

```
;; ZSKs
$include Kexample.com.+005+63935.key
$include Kexample.com.+005+64700.key


;; KSKs
$include Kexample.com.+005+54915.key
$include Kexample.com.+005+06456.key
```

Sign the zone with both KSKs and the active ZSK.

```
dnssec-signzone -k  Kexample.com.+005+54915.key -k  Kexample.com.+005+06456.key \
  -o example.com db.example.com \
  Kexample.com.+005+64700
db.example.com.signed
```

You have now introduce the new key.

Since you are rolling a KSK you will have to upload this key to your parent or have to configure it into your trust anchors (the section called “Configuring the caching forwarder”. The public key you will have to upload configure is the new one with key-id 06456.

If your parent has a `DS` RR pointing to your old key it will take time before that `DS` RR has expired from caches. The upper limit on the `t4` parameter is the signature expiration time over the `DS` RR pointing to the old KSK (54915).

dnssec-signzone provides two files that will help you during this process. `dsset-example.com.` and `keyset-example.com.`. The "dsset" file contains the DS RRs that relate to the KSKs in your zone file and the "keyset" file contains the KSKs published in your Zone file. Remember that since you are replacing keys only one of these entries (06456) will need be sent/appear at your parent.

The cleanup (phase 2)

Once you are satisfied that all trust anchors are updated and the parental `DS` RR has traveled through the DNS you can remove the old key from the set of includes:

```
;; ZSKs
$include Kexample.com.+005+63935.key
$include Kexample.com.+005+64700.key

;; KSKs
$include Kexample.com.+005+06456.key
```

Sign the zone with the new KSK and the active ZSK.

```
dnssec-signzone  -k  Kexample.com.+005+06456.key \
  -o example.com db.example.com \
   Kexample.com.+005+64700
db.example.com.signed
```

From this moment on you are in the production phase again.

Multiple KSKs

This algorithm also applies if you have multiple KSKs.The steps are: generate and include the new KSK in the zone; sign the zone with all KSKs; wait for propagation; remove one of the KSKs and sign with all left over KSKs.

Part II. Securing communication between Servers

This part considers transaction security issues. It is focused on securing the transactions between authoritative servers but the same techniques can be used to secure dynamic updates.

Table of Contents

Chapter 5. Securing Zone transfers

Introduction

The communication between hosts can be secured (read authenticated and encrypted) using a scheme based on symmetric cryptography. By sharing a key the administrators of two servers can be sure that DNS data is only being exchanged between those two boxes and that the data has not been tampered with in transit.

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[10]. We will describe how primary server `ns.foo.example` and secondary server `ns.example.com` 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, the TSIG key.
- 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

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.

Generating a TSIG key

There are multiple alternatives on how to create a shared secret.

Generating a TSIG secret with dnssec-keygen

dnssec-keygen is the tool that we use to generate a base64 encoded random number that will be used as the `secret`. The arguments that we have to provide **dnssec-keygen** to generate a TSIG key are (also see Figure 2.1, " dnssec-keygen arguments"):

```
dnssec-signzone -a hmac-md5 -b 256 -n HOST ns.foo.example-ns.example.com.
```

The command produces two files[11] 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`

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

[11]It is a feature of the **dnssec-keygen** program that it always creates two files, even when it is generating symmetric keys.

identifies the algorithm used: 5 for HMAC-MD5 (1 and 3 are for RSA and DSA respectively see Appendix A.1. in I-D.ietf-dnsext-dnssec-records). 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. Since the secret material is copied to the configuration files and these files are not used in production, you should actually consider to delete them.

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/random -a  HMAC-MD5 -b 128  -n HOST \
  ns.foo.example-ns.example.com.
Kns.foo.example-ns.example.com.+157+12274

# cat Kns.foo.example-ns.example.com.+157+12274.key
ns.foo.example-ns.example.com. IN DNSKEY 512 3 157 gQOqMJA/LGHwJa8vtD7u6w==

# cat Kns.foo.example-ns.example.com.+157+12274.private
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: gQOqMJA/LGHwJa8vtD7u6w==
```

The base64 encoded random number is: gQOqMJA/LGHwJa8vtD7u6w== it should be specified as the secret in the key statement:

```
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.

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 recommend 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 looks like:

```
key ns.foo.example-ns.example.com. {
    algorithm hmac-md5
    secret "gQOqMJA/LGHwJa8vtD7u6w==";
};
```

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 uniquely identifies both ends of the communication such as **`pri-sec.example.com.`** or **`ns.foo.example-ns.example.com.`**. For the `algorithm` use the only supported algorithm in the TSIG specification: `hmac-md5`. The `secret` is followed by 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.

Primary servers configuration of TSIG

Both the primary and secondary server should have shared secret configured by using the `key` statement 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` statement. 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;
};
```

Secondary servers configuration of TSIG

Both the primary and secondary serve should have shared secret configured by using the `key` statement 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.;};
};
```

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 re-loaded 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* flag. **`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 [12] 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
   ```

TSIG signing of Notifies

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

Possible problems

[12] 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** or via the `/proc` file system

Timing problems

Machines that are involved in a TSIG signed transaction need to have their clocks synchronized within a few[13] 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.

[13] BIND has 5 minutes hard coded

Appendix A. BIND installation

DNSSEC is available as of bind 9.3.0. The latest versions of bind can be find on ISC's ftp server. DNSSEC support is only compiled if the openssl library is configured during compilation.

configure with the `--with-openssl` flag.

If you want to have the "sigchase" capability (see the section called “Using dig for troubleshooting” compiled into **dig** you will have to set the `CFLAGS` variable to the `-DDIG_SIGCHASE=1`

Check the output of config to confirm that `openssl` was indeed found. For example:

```
cd /usr/local/src
tar -xzf bind-9.3.0rc3
cd bind-9.3.0.rc3
./configure --prefix=/usr/local --with-openssl=/sw/ CFLAGS="-DDIG_SIGCHASE=1"


...
checking whether byte ordering is bigendian... yes
checking for OpenSSL library... using openssl from /sw//lib and /sw//include
checking whether linking with OpenSSL works... yes
...
```

Please note that BIND 9.3.0 does not have DNSSEC enabled by default. Therefore you have to use the `dnssec-enable` directive in the `options` section of `named.conf`.

```
    options {
    // turn on dnssec awareness
    dnssec-enable yes;

    };
```

Appendix B. Hints and tips

Your contributions are welcomed

Using dig for troubleshooting

dig has a few switches that come in useful when troubleshooting DNSSEC setups.

`+multiline` Structures the output of **dig** so that it is easily readable as a bonus the `keyid` will be printed as a comment behind `DNSKEY` RRs.

`+cd` The `+cd` option sets the "checking disabled" bit on the query. You would typically use this when your verifying recursive nameserver would answer a `SERVFAIL` and you would like to establish if this is due to DNSSEC marking this data as "bad".

`+dnssec` The `+dnssec` option makes the server that is being queried include the DNSSEC related data. Use this in combination with the `+cd` to establish if data from a zone is signed at all or, for instance, if you want to figure out if the validity interval's on the signatures are correct.

`+trace` Traces the delegation chain. This option may be helpful if you trying to figure out where the delegation points are.

`+sigchase` Traces the the signature chain. You will also need to have a `./trusted-keys.keys` or `/etc/trusted-keys.keys` available that contains trusted key entries.

[Editors note: the sigchase option gave an abort trap at the moment we compiled these notes (bind9.3.0rc3). We will expand this item later]

Estimating zone size increase

When planning for having zones signed you have have to consider that zone signing will increase your zone file size and the amount of memory used in the authoritative nameservers. We have performed some measurements where we took a number of zone files, signed them and loaded them on a nameserver.

We started with the 1.8 thousand zones that the RIPE NCC servers on their authoritative servers. For a number of these zones the RIPE NCC is the primary server but for the largest part these are zones that the RIPE ncc is secondary for. The zones can roughly be distinguished into two classes; "end-node" zones and "delegation" zones. In end node-zones the data at most names in the zone is authoritative (the contain e.g. A, AAAA or PTR for most of the names). Delegation zones contain mostly delegations (NS) records, typically these are top level domains and "/16 reverse delegation" domains.

We signed the zone files with a 1024 bit RSASHA1 zone signing key. During the signing `NSEC` RRs with corresponding `RRSIG`s are added and all RRsets in the zone are signed. Since a delegation NS RR is not an authoritative piece of data no signature is created. So typically for a "end-node" zone one `NSEC` and two `RRSIG`s are introduced into the zone while for a delegation type zone only one of each type of security record is introduced. In Figure B.1, “Zone size vs number of owner names” we plotted the zone file size increase as a function of the number of `NSEC` records. The number of `NSEC` records correlates one to one with the number of domain names in a zone. In the figure you can clearly see a bimodal distribution. One for the "end-node" type of zones and one for the delegation type of zone.

We fitted two linear relations to this data and found that for a delegation type of zone the size increase is 350 bytes per owner name while for an "end-node" zone the increase is 672 bytes per owner name.

In Figure B.2, “Core size vs zone file size” we plotted the relation between increase in core size versus the zone file size increase due to signing. The relation is linear and the slope is roughly 0.73. In other words the core size increase is roughly 200 and 500 bytes for delegation type and end-node type zones respectively.

You can use these parameters for rough size calculations, your mileage may vary depending on the size and the algorithm of the key you use, the version of bind[14] and the content of the zone.

Figure B.1. Zone size vs number of owner names

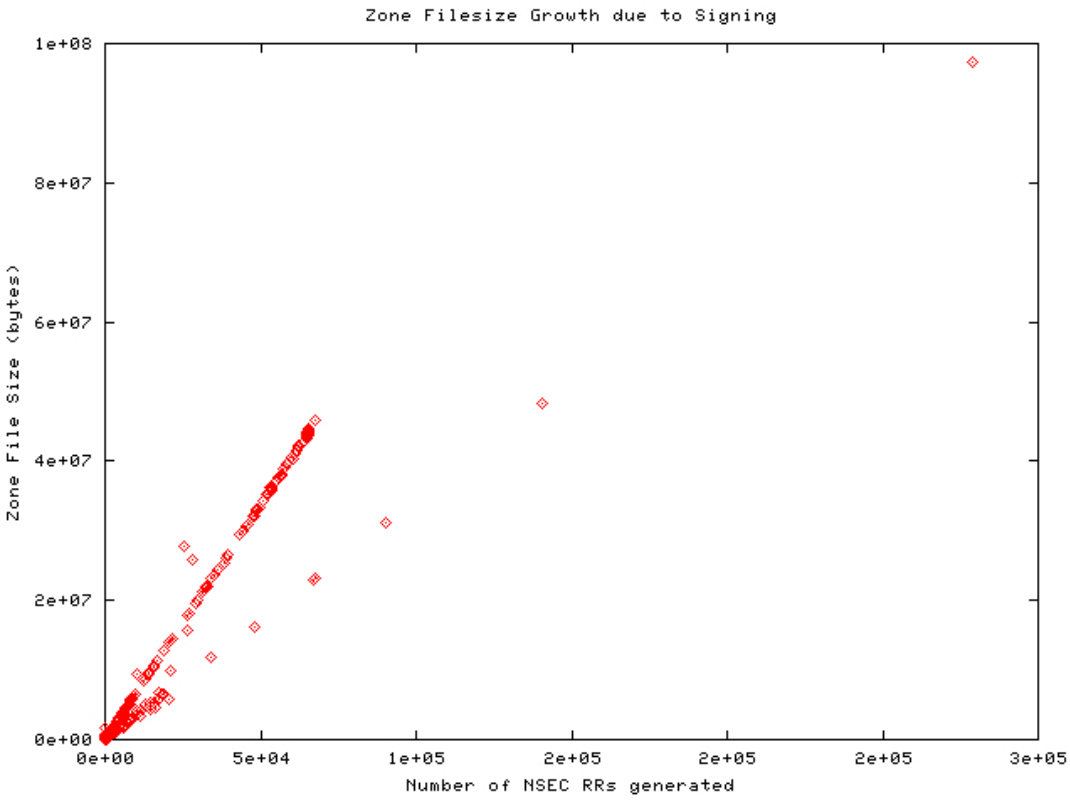
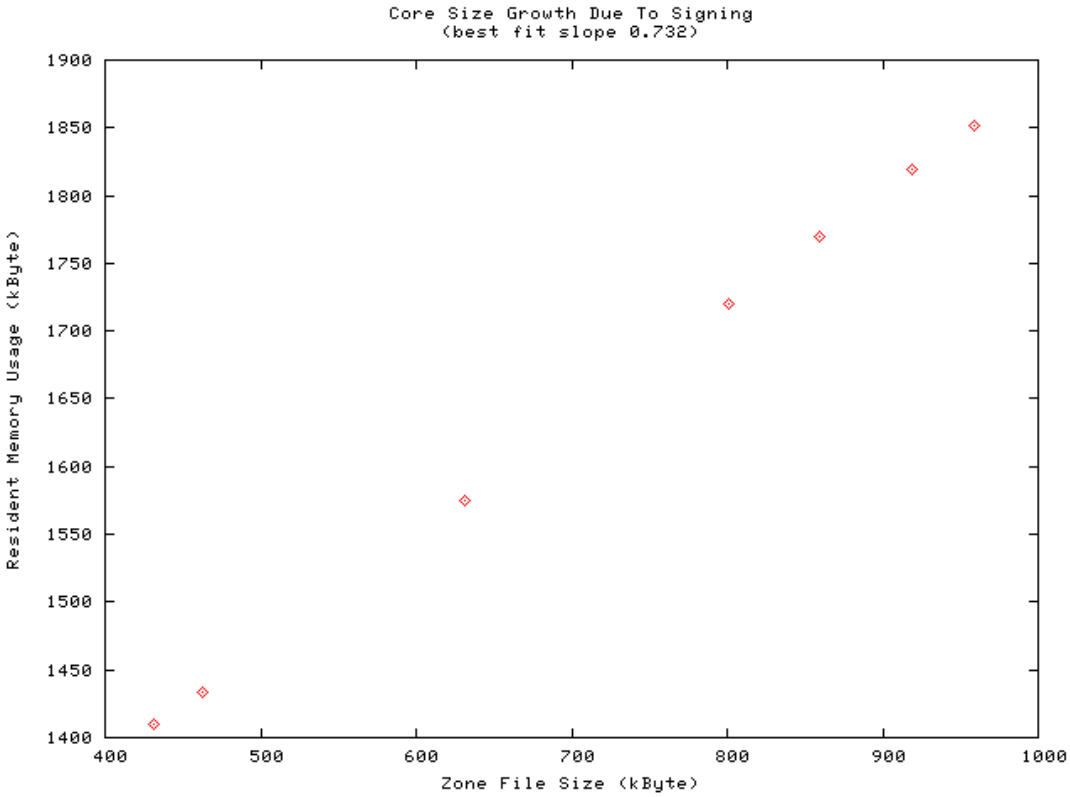


Figure B.2. Core size vs zone file size

[14] these measurements were done with a snapshot version of bind9.3



Generating Random Numbers

The generation of keys and, for the DSA algorithm, the generation of signatures requires random numbers. You should take care that the random number generator generates 'real' randomness. The quality of random numbers generated in software is debatable, and that even applies to some degree to /dev/random devices, even these extract randomness from hardware response times. You should ensure that your operating system produces a flow of good random numbers. For a machine that does not have any external sources of randomness this may be tricky to achieve and cause your key generator or signer to block and wait for entropy (randomness).

One relatively simple tool to test randomness of data streams is `ent` [http://www.fourmilab.ch/random/] from www.fourmilab.ch. A "good" measurement result from `ent` should not be interpreted as a sign that one's random number generator is perfect. There could be systematic effects that are hard to find using this particular tool[15].

Relatively cheap sources of random data are USB crypto tokens. For more information about these tokens and random number generation see http://openfortress.org/cryptodoc/random/ and references therein.

Perl's `Net::DNS::SEC` library

A shameless plug.

If you want to build tools to maintain your DNSSEC zones you may want to have a look at the `Net::DNS::SEC` library that is available on CPAN (http://search.cpan.org [http://search.cpan.org/search?query=NEt%3A%3ADNS%3A%3ASEC]). Using this extention to the `Net::DNS` library it is fairly easy to write scripts such as the one below that veri-

[15]For instance the predictable behavior of hardware during the boot of an OS may cause bit streams generated shortly after two boot sequences to be correlated

fies that the signature over a SOA will not expire within the next 24 hours.

Figure B.3. `Net::DNS::SEC` Example Script

```
#!/usr/local/bin/perl -T -Wall
#

# checkexpire.pl

# Example script that queries an authoritative server for a SOA
# record and verifies that the signatures over the record are still
# valid and will not expire in the next 24 hours.

# This anotated and somewhat verbose script is written for
# demonstration purposes only hence some possible error conditions
# are not tested for.

use strict;
use Net::DNS::SEC;
use Time::Local;

# The domain and its master server.
my $domain="secret-wg.org";
my $authoritative_server="ns.secret-wg.org";


# Setting up the resolver (use perldoc Net::DNS::Resolver for the
# documentation of this class and its methods)
my $res = Net::DNS::Resolver->new();

# Query the default resolver to find out what the address is of the
# authoritative server.
my $answerpacket_auth_server= $res->query($authoritative_server,"A");


# Digest the packet see perldoc Net::DNS::Packet and perldoc
# Net::DNS::RR::A We ignore error checking. The first RR in the answer
# section is assumed to be the A RR for ns.secret-wg.org.

my $auth_address=($answerpacket_auth_server->answer)[0]->address;

# Set up the resolver object so it queries the authoritative server.
$res->nameserver( $auth_address );

# Set up the resolver so that it talks DNSSEC
$res->dnssec(1);

# Send the query for the soa to the authoritative nameserver.
my $packet=$res->send($domain,"SOA");


# Digest the answer section, realizing there may be more than one
# RRSIG (per definition there is always one SOA RR.

my $soa;
my @soasig;
foreach my $rr ( $packet->answer ){
    if ($rr->type eq "SOA"){
        $soa=$rr;
        next;
    }
    if ($rr->type eq "RRSIG"){
        push @soasig,$rr;
        next;
    }
}


die "NO SOA RR found" unless $soa;
die "NO RRSIGs over the SOA found" unless @soasig;
print @soasig ." signatures found\n";

# Fetch the keys that belong to this zone (DNSKEYs live, like the SOA
# at the apex.)

my @keyrr;
$packet=$res->send($domain,"DNSKEY");
foreach my $rr ( $packet->answer ){
    if ($rr->type eq "DNSKEY"){
        push @keyrr,$rr;
        next;
    }
}
```

```
die "NO DNSKEYS found for $domain" unless @keyrr;



# Now loop over each signature, fetch the public part of the key with
# which the signature was made, validate the signature and do the date
# comparisson.

# See perldoc Net::DNS::RR::RRSIG for the methods to access the RRSIGs
# internals

SIGLOOP: foreach my $sig ( @soasig ){
    print "Checking signature made with key ".$sig->keytag ."\n";
    # verify the signature.
    # first select the key with the proper keytag from the key set.
    my $keyfound=0;
  KEYLOOP: foreach my $key (@keyrr){
      next KEYLOOP if ($key->keytag != $sig->keytag);
      $keyfound=$key;
      last KEYLOOP;
  }
    print "WARNING: NO public key found to validate:\n " .
        $sig->string."\n" unless $keyfound;

    # Do the actual validation.
    if (! $sig->verify([ $soa ],$keyfound)){
        # The signature did not validate. Say why.
        print "WARN: Signature made with " .$sig->keytag . " failed to verify:\n".
            $sig->vrfyerrstr;
    }else{

        # The signature validated.
        # Lets verify if we have more than 24 hours before expiration.

        $sig->sigexpiration =~ /(\d{4})(\d{2})(\d{2})(\d{2})(\d{2})(\d{2})/;
        my $expiration=timegm ($6, $5, $4, $3, $2-1, $1-1900);
        my $hourstogo=($expiration-time())/3600;
        print "WARNING: Signature made with ".$sig->tag. "will expire within ".
            $hourstogo . " hours\n" if $hourstogo <24;

    }


}


####
# $Id$
####
```

Bibliography

[rfc1034] *Domain names - concepts and facilities*. , P. Mockapetris. , 1 November 1987. http://www.ietf.org/rfc/rfc1034.txt [http://www.ietf.org/rfc/rfc1034.txt]ftp://ftp.isi.edu/in-notes/rfc1034.txt

[rfc1035] *Domain names - implementation and specification*. , P. Mockapetris. , 1 November 1987. http://www.ietf.org/rfc/rfc1035.txt [http://www.ietf.org/rfc/rfc1035.txt]ftp://ftp.isi.edu/in-notes/rfc1035.txt

[rfc2535] *Domain Name System Security Extensions*. , Donald E. Eastlake 3rd. , March 1999. http://www.ietf.org/rfc/rfc2535.txt [http://www.ietf.org/rfc/rfc2535.txt]ftp://ftp.isi.edu/in-notes/rfc2535.txt

[rfc2845] *Secret Key Transaction Authentication for DNS (TSIG)*. , P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington. , May 2000. http://www.ietf.org/rfc/rfc2845.txt [http://www.ietf.org/rfc/rfc2845.txt]ftp://ftp.isi.edu/in-notes/rfc2845.txt

[rfc3757] *Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag*. , O. Kolkman, J. Schlyter, E. Lewis. , May 2004. http://www.ietf.org/rfc/rfc3757.txt [http://www.ietf.org/rfc/rfc3757.txt]ftp://ftp.isi.edu/in-notes/rfc3757.txt

[I-D.ietf-dnsext-dnssec-records] *Resource Records for the DNS Security Extensions*. , Roy Arends. , 19 July 2004. Work in progress. http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-records-09.txt

[I-D.ietf-dnsext-dnssec-intro] *DNS Security Introduction and Requirements*. , Roy Arends, Rob Austein, Dan Massey, Matt Larson, Scott Rose. , 19 July 2004. Work in progress. http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-intro-11.txt

[I-D.ietf-dnsext-dnssec-protocol] *Protocol Modifications for the DNS Security Extensions*. , Roy Arends. , 19 July 2004. Work in progress. http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-protocol-07.txt

[I-D.ietf-dnsop-dnssec-operational-practices] *DNSSEC Operational Practices*. , Olaf Kolkman. , 14 May 2004. Work in progress. http://www.ietf.org/internet-drafts/draft-ietf-dnsop-dnssec-operational-practices-01.txt

Acknowledgments

There are numerous people who helped compiling these notes, either by helping me to understand DNSSEC or by giving feedback on earlier versions of this document.

Special thanks go to Roy Arends, Daniel Diaz, Miek Gieben, Daniel Karrenberg, Ed Lewis, Rick van Rein, and Jakob Shlyter who provided feedback on this or earlier versions of this document

A workshop organized bu USC/ISI on operational testing of DS in Washington DC has provided a substantial amount of material for version 1.3 of the document.

A 'login;' article by Evi Nemith, the text in the BIND book and the various presentations by Edward Lewis have been very helpful to gain understanding to compile version 1.1.

Document History

Revisions 1.5	Public release.
Revisions 1.4.4.1 - 1.4.4.*	Several snapshots that where not publically available. The document source was ported from TeX to docbook source and was updated to reflect experiences with key-management. Large chunks were rewritten and we added a number of figures. This version of the documentation is based on the DNSSEC bis specification and bind9.3.0beta implementation.
Version 1.3, Oct 2002	Yhe 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 \dnssec operations.
Version 1.2	was not published.
Version 1.1	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.