

# **DNSSEC Key Management and Zone Signing**

**Olaf Kolkman**  
**<net-dns@ripe.net>**

**\$Revision: 1.14 \$**

## DNSSEC Key Management and Zone Signing

## Table of Contents

About this document .....	v
1. Introduction .....	1
Background .....	1
2. Maintaining the Keys .....	2
Creating keys .....	2
Deleting keys .....	3
Rolling keys .....	4
Rolling Keys Signing Keys .....	4
Rolling Zone Signing Keys .....	6
3. Operating the signer .....	8
Stand Alone Signer .....	8
Client/Server .....	8
A. "Cookbook" .....	10
B. Installation and Configuration .....	12
Architecture .....	12
Installation .....	13
Prerequisites .....	13
Installing Net::DNS::SEC::Maint::Key and Net::DNS::SEC::Maint::Zone .....	14
Installing the zone signer client .....	15
Setting up your UNIX environment .....	15
Configuration .....	15
Configuring the SOAP based zone signer daemon .....	17
C. Where do your private keys live .....	18
D. How to make your key store/signer application more secure .....	19
Bibliography .....	21

## List of Figures

B.1. ssh configuration .....	12
B.2. Architecture outline .....	12
D.1. Securing the signer .....	19

# About this document

This documentation is under development. Just as the code this documentation came with is a development release. Please help us catch serious bugs and let us know if you have any suggestions for improvement.

# Chapter 1. Introduction

## Background

Zone Signing is the core of DNSSEC management. During the signing of a zone DNSSIG resource records are created for all the data in the zone file. Using these signatures and related public keys security aware DNS clients can verify the validity of DNS data. In addition to creating signatures the signing process introduces NSEC RRs that can be used to validate the non-existence of data.

BIND 9.3.0 contains a tool called **dnssec-signzone**. This tool signs the zone and introduces the NSEC RRs. To use this tool users have to create key pairs, keep track of these keys and ensure proper usage.

This software suite is intended to ease key management issues. Using this tool people maintaining signed zones do not have to maintain manual logs of which keys are in use.

The intention is that zone signing is "orthogonal" to the key maintenance. The **maintkeydb** command is used to maintain the keys for a given zone while the **dnssigner** program will sort out, based on the zone, which keys to use for signing, and which public keys to insert into the zone. The persons running the **dnssigner** command is not required to have knowledge of which keys are in use, they do not even need "physical" access to the private key material.

The **maintkeydb** tool offers some assistance to the key manager with maintaining consistency during the key rollovers.

The key management procedures that are implemented are based on early experience. See also the Internet Draft 'DNSSEC key operations' draft-kolkman-dnssec-operational-practices-01.txt [<http://www.ietf.org/internet-drafts/draft-kolkman-dnssec-operational-practices-01.txt>] which describes the key rollover described herein.

## Chapter 2. Maintaining the Keys

**maintkeydb** is the tool designed to maintain keys used for DNSSEC operations. This chapter intends to provide you with a number of examples of the use of **maintkeydb** while performing certain key management tasks. ( Also see Appendix A, "Cookbook" if you think this chapter is a little too verbose. )

It is assumed that the software is installed on a machine on which the private key are stored. We will refer to as the signer box or shorter, the box. Once the box is configured you can start maintaining your keys. The program **maintkeydb** is your interface. It can be used in two different ways. With direct command line arguments or as an interactive shell.

Use **perldoc maintkeydb** to read the documentation that comes with the tool itself.

### Creating keys

```
> maintkeydb create help
Purpose: create new keys
You can choose to generate a ksk, a zsk or both keys.

If keys are first create for a zone then multiple zone signing keys
are created, thereby bootstrapping the prerequisites for automatic
rollovers using the rollovers command. The system will not create keys
if it thereby might disable automatic rollover. Use "force" to force
creation of keys.
The usage is as follows:
create ["force"] <"zsk"|"ksk"|"both"> <RSASHA1|DSA|RSA|RSAMD5> <size> <zone[[ zone]...]>
```

**maintkeydb create** will create one key signing key and two zone signing keys will be created. One ZKS set as "active" which means it is used for signing, and one set as "published" which means it is not used for signing zone data. This particular set of keys is created so that the prerequisites for a successful key rollover are met.

```
bash $ maintkeydb create both RSASHA1 1024 example.com
Created 3 keys for example.com
bash $ maintkeydb list example.com
example.com      RSASHA1 25379    ZSK published   (0d00h00m)
example.com      RSASHA1 25589    ZSK active      (0d00h00m)
example.com      RSASHA1 58140    KSK active      (0d00h00m)
```

It could well be that you would like to have the zone signing key and the key signing key of different length. In that case you will have to specify either zsk or ksk instead of both in the line above.

Instead of using **maintkeydb** the command line you can also use it as a shell. Below is an example.

```
maintkeydb -i
Command? >create
Force creation?
[skip|force] > skip
Create a KSK, a ZSK or both?
type > ksk
Algorithm?
algorithm > RSASHA1
```

```

Key Size?
keysize > 2048
Enter one or more zone names
zone(s) > example.net
Created 1 key for example.net
Command? >list
active, inactive, rollover or published keys?
state > allstates
KSK or ZSK?
type > both
Enter one or more zone names
zone(s) > all
example.com          RSASHA1 25379    ZSK published (0d00h01m)
example.com          RSASHA1 25589    ZSK active   (0d00h01m)
example.com          RSASHA1 58140    KSK active   (0d00h01m)
example.net          RSASHA1 10320    KSK active   (0d00h00m)
Command? >create zsk RSASHA1 1048 example.net
Created 2 keys for example.net

Command? >list allstates zsk
Enter one or more zone names
zone(s) > all
example.com          RSASHA1 25379    ZSK published (0d00h02m)
example.com          RSASHA1 25589    ZSK active   (0d00h02m)
example.net          RSASHA1 08906    ZSK published (0d00h00m)
example.net          RSASHA1 17639    ZSK active   (0d00h00m)
Command? >exit
bash $

```

## Deleting keys

There are two methods to delete keys. You can either delete keys by providing a zone name, algorithm and key id or you can, more crudely delete all keys for a specific zone at once. Use **delete\_id** or **delete\_name** respectively. Below are two examples of deleting keys with **maintkeydb**, one in "shell" mode one from the command line.

```

bash $ maintkeydb -i
Command? >delete_id help
Purpose: Delete keys by keyid

Zone-Algorithm-KeyID uniquely defines a key. Use the list <zone> to
identify the keys. In shell mode command-line completion [tab] will show
you the available KeyIDs.
delete_id <zone> <RSASHA1|DSA|RSA|RSAMD5> <keyID[[ keyID] ...]>
Command? >delete_id
Enter a zone name
zone > example.net
Algorithm?
algorithm > RSASHA1
Enter one or more KeyIDs?
keyID > [press TAB]
08906 10320 17639
keyID > 08906
Deleting: example.net          RSASHA1 08906    ZSK published (10d21h02m)
Command? >list example.net
allstates both example.net
example.net          RSASHA1 10320    KSK active   (10d21h03m)
example.net          RSASHA1 17639    ZSK active   (10d21h03m)
Command? >exit
bash $

```

```

bash $ maintkeydb delete_name help
Purpose: Delete keys by name

To delete the KSK, the ZSK or just all keys for given algorithm and zones. Be careful
you will not be asked for confirmation.

Specfying at least one zone is mandatory.
delete_name <"zsk"|"ksk"|"both"> <RSASHA1|DSA|RSA|RSAMD5> <zone[[ zone]...>
bash $ maintkeydb delete_name both RSASHA1 example.net
Deleting: example.net          RSASHA1 10320    KSK active   (10d21h05m)

```



```
Deleting: example.net          RSASHA1 17639   ZSK active   (10d21h04m)
bash $ maintkeydb list
example.com                   RSASHA1 25379   ZSK published (10d21h07m)
example.com                   RSASHA1 25589   ZSK active   (10d21h07m)
example.com                   RSASHA1 58140   KSK active   (10d21h07m)
bash $
```

## Rolling keys

Rolling the key pairs is an operation that needs to be done on a regular basis. During a key rollover one key pair gets replaced by another. During the rollover one has to take care of the propagation of the key information through the DNS. Please refer to I-D-dnsop-dnssec-operational-practices-01 for details of the key rollover scheme that has been implemented.

There are two types of rollovers to consider. That of zone signing keys, an operation that does not need 'external' interaction and can be done relatively frequently. The other type of rollover is the rollover of keys signing keys. During that type of rollover public key information needs to be "uploaded" to DNS parents or configured in verifiers. A key signing key rollover will typically occur less frequently than a zone signing key rollover.

In general the rollover happens in two stages, during the first stage the preparations are done. The new keys et has to propagate through the Internet. How fast that happens depends on how fast the changes are applied to your zone (Through the signing operation), how fast the zone is served by secondary servers and on the TTLs on the data previously in your zone. That data may still live in distant caches (see I-D-dnsop-dnssec-operational-practices-01 for the details).

The tools do not provide hooks to test the state of the DNS (yet). You have to verify that all keys propagated to the Internet and wait for 2 TTLs before you engage in the second stage.

## Rolling Keys Signing Keys

One needs to interact with "external" parties when rolling Key Signing Keys. During the first stage of the rollover a new Key Signing Key is introduced and the "old" key is marked as being in "rollover" stage.

```
bash $ maintkeydb list example.com
example.com                   RSASHA1 25379   ZSK published (11d00h03m)
example.com                   RSASHA1 25589   ZSK active   (11d00h03m)
example.com                   RSASHA1 58140   KSK active   (11d00h03m)
bash $ maintkeydb rollover ksk-stage1 RSASHA1 example.com
bash $ maintkeydb list example.com
example.com                   RSASHA1 25379   ZSK published (11d00h04m)
example.com                   RSASHA1 25589   ZSK active   (11d00h04m)
example.com                   RSASHA1 36252   KSK active   (0d00h00m)
example.com                   RSASHA1 58140   KSK active   (11d00h04m) (R)
```

The key marked as being in rollover will be deleted during stage2 of the rollover.

What you have to do now is make sure that your parent (the .com zone in this example creates a new DS record to point to your new key signing key. To find out which keys you will have to send to your parent you use the **maintkeydb parent data** command.

```
bash $ maintkeydb parentdata key example.com
example.com. 0 IN DNSKEY 257 3 5 (
    AQPiQ9Mlv9qgsqxwrENJY/EP1vhRPedQz1X5
    8p800KKIpLUNU4dKo+9nz3yyJ2YP/V76yAZC
    SWwvdoXfWaTSB5lvO4+EzmLyCUu8Ziz1D3jK
    AG3QYMoFyfr1AajteQ5kdyKoCXawRLgVSGdc
    J7BT/xzPHK1KgaJ96601otwKM1E8ML6HboTP
    jw+Dx0YVYhU8F/Wh8BflvbI8drgnX0yxi9+V
```

```
kMx+FtvNO6i1MU2GVZp7j5CbACOhoSdK22U4
TpWUvG+IvulhZ6S6TjZFkwpq0xM0NecvE7Yn
WimUwgmth2HJTZJCzbOWRMIH3X3PX0MCMVww
+efOWnPFVlEy+V8vfunl
) ; Key ID = 36252
```

```
bash $
```

If your parent's registrar only accepts "DS" RRs you can alternatively specify `ds` instead of `key`:

```
bash $ maintkeydb parentdata ds example.com
example.com. 0 IN DS 36252 5 1 f2fb1958f15832749ad25f5e9707abcb933a8162
bash $
```

If you want to verify what appears in your zone file you can issue the **`maintkeydb showkeys`** command this will output the key set as it will appear in your zone file.

```
bash $ maintkeydb showkeys example.com

;;;;;;;; DNSKEY RRs for example.com ;;;;;;;;;
example.com. 0 IN DNSKEY 256 3 5 (
    AQOz8oz5sm5CCu4gCZBCI0URL319aAkrAGU6
    rPYqNq8e3MZ5Aae3NAPwYsc7RX60U9xLNwtj
    qdnR0Mjbi1D1lXVAAcdu5fm/B2xOhdx+8Q1a
    +SFKvM9ndQeARwSsE3NSIQ2rANFTNkoT+tkg
    6S6QSZdmpvhlESgHud91sP1M5yxsUw==
) ; Key ID = 25379 ZSK
example.com. 0 IN DNSKEY 256 3 5 (
    AQO+fa+ITw4/aY5B0L7mRaCkeFMW0YFcodun
    rYPhHnmU+2duUP/Z29mUW/MBEuaBCoXpzC7Z
    ekaTpFx5t8MZ/2MsqtjhrifbE309Zgbo2fLC
    vAFB7AAVpvsZdscTbQ/wNxS7m93q4WtyDDyh
    IyV/KGvQ7eGRfd4GQVN8dhsgBpty5Q==
) ; Key ID = 25589 ZSK
example.com. 0 IN DNSKEY 257 3 5 (
    AQPiq9Mlv9qqsqxwrENJY/EP1vhrPedQz1X5
    8p80OKKIPLUNU4dko+9nz3yyJ2YP/V76yAZC
    SWwvdoXfWaTSB5lv04+EzmLyCUu8ZizlD3jK
    AG3QYMoFyfr1AajteQ5kdyKoCXawRLgVsgdc
    J7BT/xzPHK1KgaJ9660lotwKM1E8ML6HboTP
    jw+Dx0YVYhU8F/Wh8BflvbI8drgnX0yxi9+V
    kMx+FtvNO6i1MU2GVZp7j5CbACOhoSdK22U4
    TpWUvG+IvulhZ6S6TjZFkwpq0xM0NecvE7Yn
    WimUwgmth2HJTZJCzbOWRMIH3X3PX0MCMVww
    +efOWnPFVlEy+V8vfunl
) ; Key ID = 36252 KSK (upload to parent)
example.com. 0 IN DNSKEY 257 3 5 (
    AQPZOKdG8V8mf63picc28uNeAY5nwnkMkNT
    oEbQsAlNAC4tlMxgb7gs2mcL6z05qTLUIj9F
    tmR09tpfy7RTaM2XwHx2n9opLhDJYEBd4pYD
    pGe+ktXKmLaSkjgIF9RvQ44sHpIItteqgakGr
    TlLavtj9IBl8V/Mvam55DLXfjtjV5w==
) ; Key ID = 58140 KSK (to be deprecated)

bash $
```

or, if you want to see the KSKs in your keyset only use the `ds` option, that will only print the key signing keys, as a bonus the DS RRs will be printed.

```
bash $ maintkeydb showkeys ds example.com

;;;;;;;; DNSKEY RRs for example.com ;;;;;;;;;
example.com. 0 IN DNSKEY 257 3 5 (
    AQPiq9Mlv9qqsqxwrENJY/EP1vhrPedQz1X5
    8p80OKKIPLUNU4dko+9nz3yyJ2YP/V76yAZC
    SWwvdoXfWaTSB5lv04+EzmLyCUu8ZizlD3jK
    AG3QYMoFyfr1AajteQ5kdyKoCXawRLgVsgdc
    J7BT/xzPHK1KgaJ9660lotwKM1E8ML6HboTP
    jw+Dx0YVYhU8F/Wh8BflvbI8drgnX0yxi9+V
    kMx+FtvNO6i1MU2GVZp7j5CbACOhoSdK22U4
    TpWUvG+IvulhZ6S6TjZFkwpq0xM0NecvE7Yn
    WimUwgmth2HJTZJCzbOWRMIH3X3PX0MCMVww
    +efOWnPFVlEy+V8vfunl
) ; Key ID = 36252 KSK (upload to parent)
```

```
example.com. 0 IN DNSKEY 257 3 5 (
    AQPZOKdG8V8mf63picc28uNeAY5nwknkMkNT
    oEbQsALnAC4t1Mxgb7gs2mcL6z05qTLUIj9F
    tmR09tpfy7RTaM2XwHx2n9opLhDJYEdD4pYD
    pGe+ktXKmLaSkjgIF9RvQ44sHpIiteqqakGr
    T1Lavtj9IB18V/Mvam55DLXFjtjV5w==
    ) ; Key ID = 58140 KSK (to be deprecated)

;;;;;;;; DS RRs for example.com ;;;;;;;;;
example.com. 0 IN DS 36252 5 1 f2fb1958f15832749ad25f5e9707abcb933a8162
; The next DS RR is an old one, it is to disappear
example.com. 0 IN DS 58140 5 1 877d2eeba0e626066bd8ee5e5099ab9c7e1e387d
bash $
```

Note the comments that indicate which keys are to be sent to the parent.

You will now have to wait until your parent has published the "new" DS RR and for the "old" DS RR to expire from all the caches, that live somewhere on the Internet and are not under your control. It will take at least the TTL value of the "old" DS RR as published by your parent for that to happen. You may want to play safe and wait for the signature over the "old" DS to be expired before pulling the DNSKEY it points to. Pulling the DNSKEY is done by "stage2" of the rollover.

```
bash $ maintkeydb rollover ksk-stage2 RSASHA1 example.com
bash $ maintkeydb list example.com
example.com      RSASHA1 25379      ZSK published (11d18h16m)
example.com      RSASHA1 25589      ZSK active   (11d18h16m)
example.com      RSASHA1 36252      KSK active   (0d17h13m)
```

You can verify that the key previously marked to be in "rollover" has now been removed.

## Rolling Zone Signing Keys

The prerequisite for a zone signing key rollover is that there are two keys present, one is set to active and is used for signing, the other is only published i.e. available in the DNS, but is not used for signing. If you have used the "create" function with the default settings the two keys should have been created.

Again you have to take into account that it takes a while before data published in the DNS has reached all the clients. So do not roll to fast. The timing mostly depends on your TTL settings.

We perform a stage one rollover using the interactive mode:

```
bash $ maintkeydb list example.com
example.com      RSASHA1 25379      ZSK published (11d20h03m)
example.com      RSASHA1 25589      ZSK active   (11d20h03m)
example.com      RSASHA1 36252      KSK active   (0d20h00m)
```

```
Command? >rollover
Enter rollover stage?
rollover stage > zsk-stage1
Algorithm?
algorithm > RSASHA1
Enter one or more zone names
zone(s) > example.com
Command? >list
active, inactive, rollover or published keys?
state > allstates
KSK or ZSK?
type > zsk
Enter one or more zone names
zone(s) > example.com
```

```
example.com      RSASHA1 25379   ZSK active      (0d00h00m)
example.com      RSASHA1 25589   ZSK published   (0d00h00m) (R)
Command? > exit
bash $
```

You can tell that the key with key ID 25589, the key that was previously active is set to "published" and has its rollover attribute set (the "(R)" behind the at the end). There is a newly created key with ID 25379 that is set to active. The times are both "reset" to 0 as these indicate the time since the last state change and both keys had a state change.

In the stage2 key rollover the published key with the rollover key will be deleted and a new key will be published that is ready for introduction as a signing key in the future. We demonstrate the stage two rollover in the command line mode.

```
bash $ maintkeydb rollover zsk-stage1 RSASHA1 example.com
There is a key marked as being rolled
You probably want to run zsk-stage2 for example.com (RSASHA1)
bash $ echo $?
6
bash $
```

Oops.. typo... you see the tool provides a warning and returns a non-zero return code.

```
bash $ maintkeydb rollover zsk-stage2 RSASHA1 example.com
bash $ echo $?
0
bash $ maintkeydb list example.com
example.com      RSASHA1 25379   ZSK active      (0d00h07m)
example.com      RSASHA1 36252   KSK active      (0d21h01m)
example.com      RSASHA1 61760   ZSK published   (0d00h00m)
```

# Chapter 3. Operating the signer

## Stand Alone Signer

The `dnssigner` is the application that uses the key store to sign zones. The intention is that the "user" is not aware of which keys are currently being marked as "active", "passive" or in "rollover" but just signs the zone. During the signing operation the appropriate set of public keys will be added and the zone will be signed with the appropriate private keys.

The **dnssigner** command takes the following form:

```
dnssigner -h
dnssigner -V

General Flags
-h print this help message and exit
-V print version information and exit
-v increase verbosity

Client
zonefile      name of the zonefile.

-o <origin>    origin of the zone. If not supplied the name of the zone
               will be used as origin.

-t            print statistics of the signing process to stderr.

-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)
```

The arguments are similar BIND's **dnssec-signzone** except that key information is not needed.

## Client/Server

The same functionality can be provided through a "SOAP" based zone signer server-client application. The client provides the zones and arguments while the server does all the work, all communication is over a SOAP channel. Refer to the section called "Configuring the SOAP based zone signer daemon" for how to configure the daemon.

The client has exactly the same arguments as **dnssigner** but needs the address and port number of the server.

```
Usage:
dnssigner_client -H <host> -P <port> [-o <ORIGIN>] [-s <STARTDATE>] [-e <ENDDATE>] <ZONEFILE>
dnssigner_client -H <host> -P <port> -o <ORIGIN> (Zone file is fed through STDIN)
dnssigner_client -?
```

`dnssigner_client` takes (either from STDIN or from given file name) an unsigned DNS zone file, passes it through DNSSEC Signer Appliance and puts the signed zone file to STDOUT.

Options:

```
-? or -h      Help. This message.

-H host       Host on which the dnssigner_daemon process runs

-P port       Port on which the dnssigner_daemon process runs

-o ORIGIN     Origin. if file is supplied it is optional and
               file name is taken as the origin.

-s STARTDATE  Start date
```

-e ENDDATE	End date
ZONEFILE	DNS zone file

# Appendix A. "Cookbook"

Here we describe the steps to take when maintaining a zone.

Create keys

```
bash $ maintkeydb create KSK RSASHA1 2048 example.net
Created 1 key for example.net
bash $ maintkeydb create ZSK RSASHA1 1024 example.net
Created 2 keys for example.net
bash $
```

Use the signer to sign your zone and publish the signed zone in the DNS.

```
bash $ dnssigner example.com
Output written to :example.com.signed
```

After some time (say a few months) roll your zone signing keys.

```
bash $ maintkeydb rollover zsk-stage1 RSASHA1 example.net
```

Use the signer to sign your zone and publish the signed zone in the DNS. Wait until the change has been picked up by all your secondary servers and then wait at least the the maximum TTL value over all the records in your zone, then proceed with stage2.

```
bash $ maintkeydb rollover zsk-stage2 RSASHA1 example.net
```

Use the signer to sign your zone and publish the signed zone in the DNS.

Once ever so often (say once or twice per year) roll your key-signing keys

```
bash $ maintkeydb rollover ksk-stage1 RSASHA1 example.net
```

Use the signer to sign your zone and publish the signed zone in the DNS. Remember the TTL on the DS record currently at your parent. (**dig example.net DS**) and upload the new key that you obtain from the database with:

```
example.net. 0 IN DNSKEY 257 3 5 (
    AQOv4Vvdv2K6zYhuc20+Kd0r9DbwEZamqig8
    hthWSd02UF9MjWs2KRyYYGmMPfQktIwe6hyD
    gxcvWKEvMKn1swJWRT/jWhU6VA4vTW8a8h60
    E5p8und0vp3+67kz2cuZpzEaZ1j4boj42kmX
    SSHmCsS2BcmcwWPsEvEg3ikQFFT1VrCGIUl8
    pUlJJLE+rczNND+9ab3eg4BzB1DTbRzHwkBj
    +giX3KezJ92SVjK0k8BDj/QlkyaaXuJcJG
    CjbJynIDL85ywdi66YYpGrELDuvyiDQ++os3
    FfPslYdIfZ6RDQMULhrXPb/wJOWopIfxSR4/
    Eqz5djrlcChfehplYTRx
    ) ; Key ID = 2526
example.net. 0 IN DS 2526 5 1 5aa9bff246e645776ab9cc3de130978df82e6090
```

Wait until your parent has published the new DS in all its authoritative servers and then at least another TTL of the previous DS (you noted that above). Only then perform stage2 of

the KSK rollover:

```
bash $ maintkeydb rollover ksk-stage2 RSASHA1 example.net
```

Use the signer to sign your zone and publish the signed zone in the DNS and the rollover is done.

```
bash $ dnssigner example.com  
Output written to :example.com.signed
```

Note that the command issued is exactly the same, even after the rollover of the keys. The whole issue of key maintenance has been separated from the signing of the zone.



# Appendix B. Installation and Configuration Architecture

See Figure B.2, "Architecture outline".

We provide the a perl library(`Net::DNS::SEC::Maint::Key`) that implements a "key-store". And a perl library(`Net::DNS::SEC::Maint::Zone`) that implements the interactions between zonefiles and the keystore. The **maintkeydb** application implements the user interface using the first library while **dnssigner** is the user interface for the second. These software components can be integrated in the provisioning chain but it is possible, and preferred, to use these components to build a key store/signer application server.

The libraries can be used to build additional key management applications.

If the application server is properly set up getting access to the private keys will be non-trivial for users that do not have physical access to the machine.

Using ssh magic as in Figure B.1, "ssh configuration", access to the key store/signer application can be provided through a dedicated interactive shell. That shell can be used to perform key pair creation, rollovers and other key management tasks. Users of the shell do not have access to the private key material.

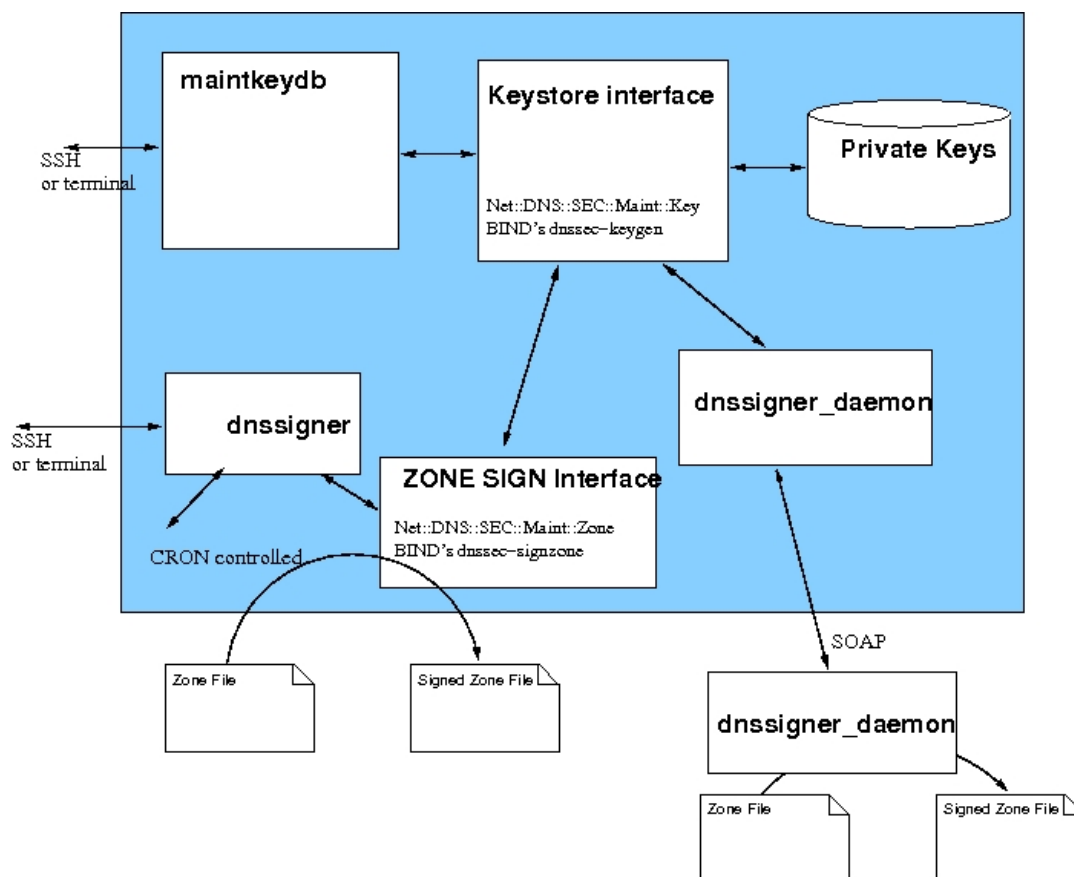
## Figure B.1. ssh configuration

Example content of `~/.ssh/authorized_keys2`  
consult the ssh documentation for details.

```
command="/usr/local/bin/maintkeydb -i",no-port-forwarding,\
no-X11-forwarding,no-agent-forwarding
ssh-rsa AAAAB3NzaClyc2EAAAABIwAAA...zrPyXc
```

Zones can be signed through a SOAP based client/server application. The zone signer uses the information stored in the key store to sort out which zones use which keys.

## Figure B.2. Architecture outline



The key store/signer application server can be build using a out of the box components. A commodity PC with Linux and FreeBSD installed can be used. Some effort should be put in securing the box. We'll give some suggestions later.

## Installation

These are the instruction for setting up the system. If you are setting up a key store/signer application server these instructions are relevant. See below for how to set up a zone signing client.

We assume that all installations are done in `/usr/local` your mileage may vary if you try to install the software elsewhere.<sup>1</sup>

## Prerequisites

First you have to make sure you have installed BIND 9.3.0 or more recent. It is important that during the installation you have configured the package with the `--with-openssl` configuration otherwise DNSSEC functionality will not be available. For example:

```
cd bind9source-dir/
./configure --with-openssl=/usr/local --prefix=/usr/local
make
```

On the key store/signer application server you will need to install a recent perl5 and you will have to use CPAN to install a number of perl dependencies.

<sup>1</sup>Do not hesitate to contact the developer if you run into problems

We have made a "bundle" available together that will allow for "easy" installations of the dependencies. Download the bundle's tar ball. And perform the following commands.

If you would like to install the perl dependencies manually than you can get a listing of them by unpacking the bundle and issuing the command **perldoc KeystoreSignerPre**.

There are at least two non-perl libraries that you have to have installed for all these modules to be installed successfully. You will need openssl that is used by the perl crypto libraries. Since there is some XML library dependency you will have to have the expat libraries installed on your system. These are available on sourceforge [expat.sourceforge.net].

```
> tar -xvzf Bundle-Private-KeystoreSignerPre-0.001_1.tar.gz
> cd Bundle-Private-KeystoreSignerPre-0.001_1
> perl Makefile.PL
> sudo make install
> sudo perl -MCPAN -e 'install Bundle::Private::KeystoreSignerPre'
```

Just enter the defaults for any questions asked during the installation process.

During the installation process you may see warnings like:

```
The most recent version "2.07" of the module "File::Copy"
comes with the current version of perl (5.8.4).
(...)

Bundle summary: The following items in bundle
Bundle::Private::KeystoreSignerPre had installation problems:
  File::Basename File::Copy File::Basename
```

it is safe to ignore these if you have a recent version of perl.<sup>2</sup>

## Installing **Net::DNS::SEC::Maint::Key** and **Net::DNS::SEC::Maint::Zone**

Once you have the prerequisite bundles installed you can start installing **Net::DNS::SEC::Maint::Key** and **Net::DNS::SEC::Maint::Zone**. You will need to install the packages in the above order. What follows is an example install session. Except for the version numbers, which may be different, you can just cut and paste these commands.

```
> tar -xvzf Net-DNS-SEC-Maint-Key-0.010_1.tar.gz
> cd Net-DNS-SEC-Maint-Key-0.010_1
> perl Makefile.PL PREFIX=/usr/local
> make
> make test
> sudo make install
```

```
> tar -xvzf Net-DNS-SEC-Maint-Zone-0.010_2.tar.gz
> cd Net-DNS-SEC-Maint-Zone-0.010_2
> perl Makefile.PL PREFIX=/usr/local
> make
> make test
> sudo make install
```

You should not get complaints about missing dependencies when you run **perl Make-**

---

<sup>2</sup>Other errors than these are an indication that things go wrong. During the tests of this bundle we constantly ran into a problem with one of the more esoteric dependencies. During the install of the libwww-perl package a few of the `t/robottests`. After digging around for some time we found that `/etc/hosts` contained the wrong IP address mapping for our hostname.

**file.PL**. During **make test** a number of tests are run. They will surely fail if **dnssec-keygen** and or **openssl** are not in your path.

## Installing the zone signer client

Since the zone signer client has fewer dependencies the zone signer client script can be installed as a separate package. This package only depends on **IO::Handle**, **File::Basename**, **Getopt::Std** and **SOAP::Transport::HTTP**.

If you are uncertain the following command should get you all set:

```
perl -MCPAN -e 'install qw(IO::Handle File::Basename SOAP::Transport::HTTP);'
```

Note that **Getopt::Std** is excluded. It comes with recent perl versions.

The installation package can be created from the **Net-DNS-SEC-Maint-Zone** distribution by running the **create-client-dist.sh** command.

```
> tar -xvzf Net-DNS-SEC-Maint-Zone-0.010_2.tar.gz
> cd Net-DNS-SEC-Maint-Zone-0.010_2
> ./create-client-dist.sh
```

This will create a tar ball named **Net-DNS-SEC-Maint-ZoneSigner-0.00\_01.tar.gz**(version number probably differs). Copy this file to the appropriate machine and install.

```
> tar -xvzf Net-DNS-SEC-Maint-ZoneSigner-0.00_01.tar.gz
> cd Net-DNS-SEC-Maint-ZoneSigner-0.00_01
> perl Makefile.PL PREFIX=/usr/local
> make
> make test
> sudo make install
```

## Setting up your UNIX environment

In order for the tools to work you will have to set up a couple of directories in which the private key material will be kept.

All users of the key store will need to be member of a specific group we use the group **dnssecmt** as the example throughout this document. Make sure you edited **/etc/group** to include the uid's you want to allow access to private key material.

Create the needed directories and set the appropriate permissions by issuing the following commands.

```
mkdir /usr/local/var/dnssec_maint/
mkdir /usr/local/var/dnssec_maint/DNS_KEY_DB
mkdir /usr/local/var/dnssec_maint/log
mkdir /usr/local/var/dnssec_maint/tmp
chmod -R o-rwx /usr/local/var/dnssec_maint
chgrp -R dnssecmt /usr/local/var/dnssec_maint
chmod -R g+rwX /usr/local/var/dnssec_maint
```

## Configuration

Once you have installed the software you have to configure your key store/signer. Both the key store and the signer depend on the same configuration settings.

The Net::DNS::SEC::Maint::Key package came with `dnssecmaint-config`. You can use this program to install a configuration file. At a later stage you can use this program to modify your configuration.

`dnssecmaint-config` is called without arguments. It will ask for a few configuration settings. In most cases the defaults make sense. What follows is an example session. The program must be run with write permissions for the directory where you want to store the configuration file (default location for this file will be `/usr/local/etc/dnssecmaint.conf`)

```
This is a program to write Net::DNS::SEC::Maint configuration files.
It is typically used at install time or to create alternative configurations.
Type 'exit' to leave the program.
----
conf file specifies where the configuration file can be found
conf file is set to /usr/local/etc/dnssecmaint.conf
Enter value for conf file>/usr/local/etc/dnssecmaint.conf
----
dns_key_db Path to the directory in which the key database is kept
dns_key_db is set to /usr/local/var/dnssec_maint/DNS_Key_DB
Enter value for dns_key_db>/usr/local/var/dnssec_maint/DNS_Key_DB
----
dnssec_keygen full path to BIND's dnssec-keygen command with optional arguments
This value is currently set using the DNSSECMAINT_DNSSEC_KEYGEN
dnssec_keygen is set to /usr/local/sbin/dnssec-keygen -r /dev/urandom
Enter value for dnssec_keygen>/usr/local/sbin/dnssec-keygen -r /dev/urandom
----
dnssec_signzone full path to BIND's dnssec-signzone command with optional arguments
This value is currently set using the DNSSECMAINT_DNSSEC_SIGNZONE
dnssec_signzone is set to /usr/local/sbin/dnssec-signzone -r /dev/urandom
Enter value for dnssec_signzone>/usr/local/sbin/dnssec-signzone -r /dev/urandom
----
dsa_keysizekey Default size for DSA Key Signing Keys
dsa_keysizekey is set to 1024
Enter value for dsa_keysizekey>1024
----
dsa_keysizezone Default size for DSA Zone Signing Keys
dsa_keysizezone is set to 512
Enter value for dsa_keysizezone>512
----
logdir specifies the directory under logfiles are stored
logdir is set to /usr/local/var/dnssec_maint/log
Enter value for logdir>/usr/local/var/dnssec_maint/log
----
maintgroup Name of group that has R/W access to the dnssecmt
maintgroup is set to dnssecmt
Enter value for maintgroup>dnssecmt
----
rsa_keysizekey Default size for RSA Key Signing Keys
rsa_keysizekey is set to 2048
Enter value for rsa_keysizekey>2048
----
rsa_keysizezone Default size for RSA Zone Signing Keys
rsa_keysizezone is set to 768
Enter value for rsa_keysizezone>768
----
tmpdir Path to the directory in which temporary files are stored
tmpdir is set to /tmp/
Enter value for tmpdir>/tmp/
Save configuration file to:/usr/local/etc/dnssecmaint.conf? (yes|no)>yes
To use this configuration file you have to set DNSSECMAINT_CONFFILE=/usr/local/e
tc/dnssecmaint.conf
```

The last line is particularly important. You will have to set the `DNSSECMAINT_CONFFILE` to point to the relevant configuration file. You are best off if you do this for all users of the system.

On a related note. Most configuration parameters can be overwritten by environment variables. This is essentially what the `dnssecmaint-config` does internally. At startup it tries to

establish the path to BIND's `dnssec-keygen` program and then sets `DNSSEC-MAINT_DNSSEC_KEYGEN`. When the `dnssecmaint-config` asks for the path for `dnssec-keygen` you see a warning that the default presented is read from the `DNSSEC-MAINT_DNSSEC_KEYGEN` environment variable.

Finally a *warning*. The system defaults to the use of `/dev/urandom` as the random number generator. The reason for doing so is that on a server without mouse and/or keyboard the amount of entropy gathered will not be enough to keep `/dev/random` going. `/dev/urandom` are pseudo random and not the best choice for key generation. Also see `truly_random`

## Configuring the SOAP based zone signer daemon

We provide `dnssigner` for signing zones while having direct access to the filesystem on which the private keys live. This is often not the model under which the system is operated. Therefore we also provide an `dnssigner_daemon` and `dnssigner_client` application that communicate to each other over a "SOAP" based connection (see Figure B.2, "Architecture outline" and the section called "Installing the zone signer client").

You should start the daemon at system initialization. Start with two parameters the IP address and the port the daemon should start on.

**`dnssigner_daemon -h ipaddress -p portnumber`**

Whenever you run **`dnssigner_daemon`** you will have to use the same IP address (or hostname) and portnumber.

# Appendix C. Where do your private keys live

The system has been designed to be used as a frontend to BIND's `dnssec-keygen` and `dnssec-signzone`. Any person with shell access and appropriate permissions will have access to the private key material. The `maintkeydb` tool will obfuscate the private key material and if `maintkeydb` is used as a "user shell" than users will not be able to see the private key material.

The key material is stored in the directory configured in the configuration file under `dns_key_db` this directory defaults to `/usr/local/var/dnssec_maint/DNS_Key_DB`.

For each zone for which keys are maintained there is a sub directory with the name of that zone. In these zones there is on directory called `Expired_keys`. This is where keys are moved to when deleted. So in case of accidental deletion somebody with physical access can still get to the private key material.

In addition to the `Expired_keys` directory the zone specific directories contain files called `K<zonzme>.+<algid>+<keytag>.(adm|attr|key|private)`. The files with the extension `key` and `private` contain the public and the private key as generated by `dnssec-keygen` the file with the extension `attr` contains "attribute" information needed to operated the key store, while the file with extension `adm` contains some administration and audit information.

You should replicate the database directory on a regular basis. Either by using a mirrored disk or by making regular backups on tape, floppy or optical media. Note that the backup media contain private key material and must thus be protected against disclosure or theft.

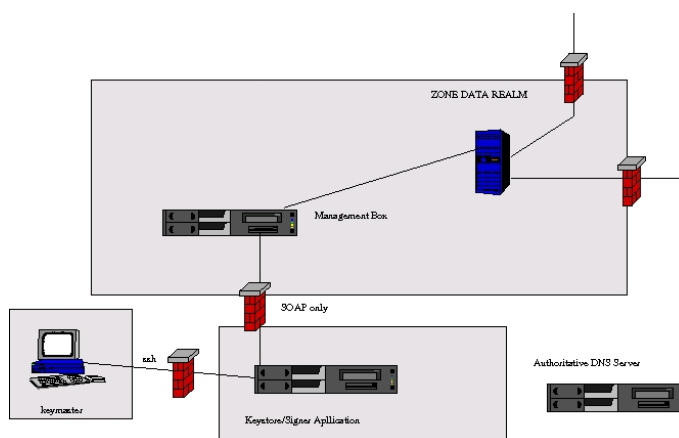
One of the methods to protect the private key material is to store it on an encrypting file system (for example `CFS` [<http://www.crypto.com/software/>]). When using a encrypting filesystem backups or replications can be made from the encrypted private keys and the private keys are better protected against physical theft.

# Appendix D. How to make your key store/signer application more secure

The only thing we provide is the software to create the key store and the dnssigner that interacts with it. It is your own responsibility to create an application server that suits your security needs. Below we provide some hints on what sort of solutions you can apply to make your server more secure. The assumption is that dnssigning has to be done in an operational environment and on a regular basis. "Sneakernet" is not an option.

Random Number Generator	The system defaults into using <code>/dev/random</code> this choice was made to prevent the system from blocking while waiting for entropy to be gathered from a not-present keyboard. We suggest to use hardware random number generators such as the ones available on USB devices. See <code>truly_random</code> for details.
Encrypt Private Keys	Use an encrypted file system to store the private keys (see Appendix C, <i>Where do your private keys live</i> )
Root access	The root user has access to the private key material  Only allow root access from "the console".
Network security	Make sure there are several firewalls between the application and the Internet.  Connect the key store/signer application server to a "management machine" through a cross cable.  Use IPtables to only allow an SSH connection and a connection over the SOAP port from the management machine.  Only allow the "keymaintainer" to log in via ssh, make the <code>maintkeydb</code> program the default shell for that user.

**Figure D.1. Securing the signer**





# Acknowledgements

Paul Wouters, Miek Gieben and his colleagues at NLnet Labs for testing early beta's of this work and for giving feedback. Timothy Mc Ginnis and Emma Bretheric for reviewing the documentation.

# Bibliography

## World Wide Web

[truly\_random] Rick van Rein. Copyright © 2002 OpenFortress. *How To Generate Truly Random Bits*.  
<http://openfortress.org/cryptodoc/random/> Link verified: April 2005 .

## IETF documents

[dnssec-operational-practices] Olaf M. Kolkman and Miek Gieben. Copyright © 2005 ISOC. *DNSSEC Operational Practices*. March 2005.  
<ftp://ftp.ripe.net/internet-drafts/draft-ietf-dnsop-dnssec-operational-practices-03.txt> Link verified: April 2005 .

[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>]

[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>]

[rfc4033] *DNS Security Introduction and Requirements* . R. Arends . R. Austein . M. Larson . D. Massey . S. Rose . March 2005 . <http://www.ietf.org/rfc/rfc4033.txt> [<http://www.ietf.org/rfc/rfc4033.txt>]

[rfc4034] *Domain names - implementation and specification* . R. Arends . R. Austein . M. Larson . D. Massey . S. Rose . March 2005 . <http://www.ietf.org/rfc/rfc4034.txt> [<http://www.ietf.org/rfc/rfc4034.txt>]

[rfc4035] *Protocol Modifications for the DNS Security Extensions* . R. Arends . R. Austein . M. Larson . D. Massey . S. Rose . March 2005 . <http://www.ietf.org/rfc/rfc4035.txt> [<http://www.ietf.org/rfc/rfc4035.txt>]