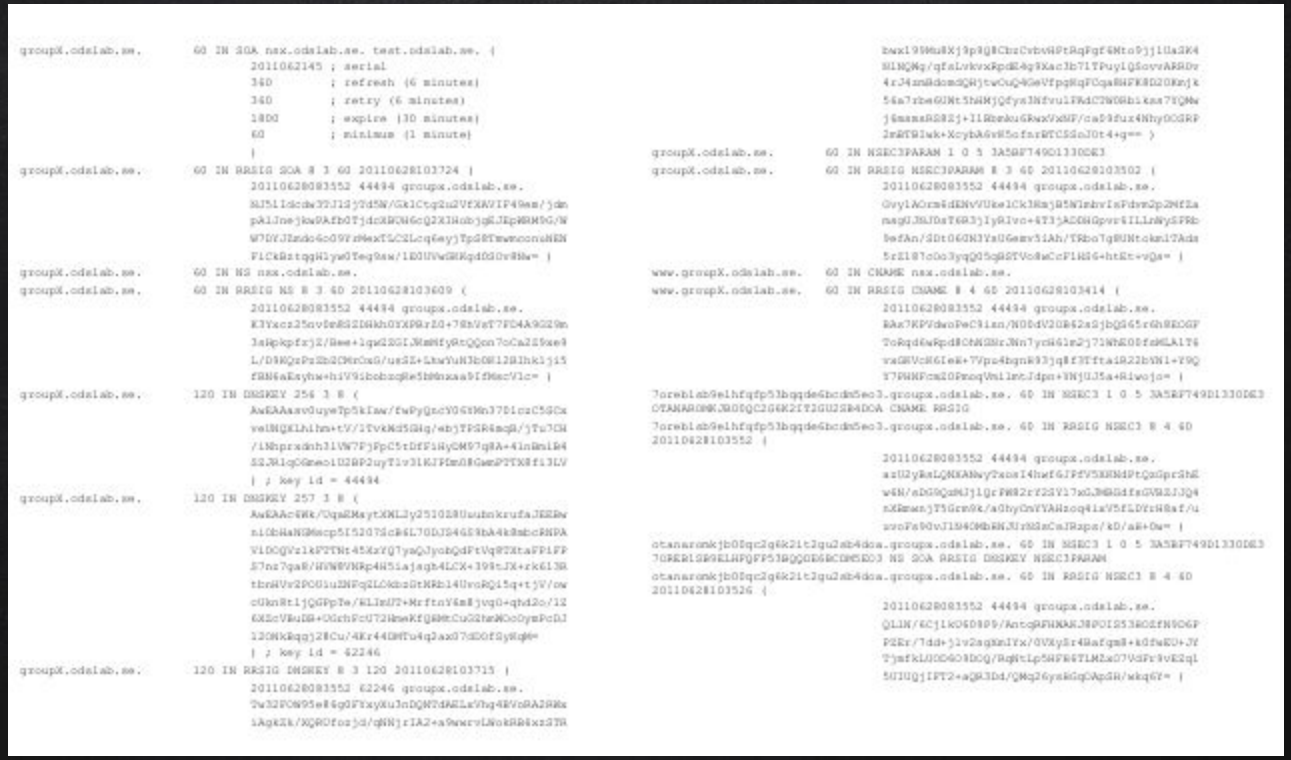
Signer software

```
$ORIGIN groupX.odslab.se.
$TTL 60
@       SOA nsX.odslab.se. test.odslab.se. (
                    2011062100 ; serial
                    360        ; refresh (6 minutes)
                    360        ; retry (6 minutes)
                    1800       ; expire (30 minutes)
                    60         ; minimum (1 minute)
                )
@       NS    nsX.odslab.se.
www     CNAME nsX.odslab.se.
```

Unsigned zone file



```
groupX.odslab.se.       60 IN SOA nsx.odslab.se. test.odslab.se. (
                                20110621145 ; serial
                                3600        ; refresh (6 minutes)
                                3600        ; retry (6 minutes)
                                18000       ; expire (30 minutes)
                                60          ; minimum (1 minute)
                                )
groupX.odslab.se.       60 IN RRSIG SOA 8 3 60 20110628103724 (
                                20110620083552 44494 groupx.odslab.se.
                                NJ51ldcdw3PJ1djYd5N/GklCtqQnJVfXAVIP49em/jdm
                                pAlJnejkwPAfb0TjdcKWCN6cQ2XIKobjgEJEpWRM9G/W
                                W7UYJEndo6cO9YzMexT5C2Lcq4eyjTpSRTmwmconeNEN
                                F1CkBztqqH1yw0Teq9aw/1E0UVwSWKqd0S0v8Nw= )
groupX.odslab.se.       60 IN NS nsx.odslab.se.
groupX.odslab.se.       60 IN RRSIG NS 8 3 60 20110628103609 (
                                20110620083552 44494 groupx.odslab.se.
                                K3Yxcs25ov5nRS2DHkh0YXPRrZG+78bVaT7FD4A9Q29m
                                3sHpkpfxj2/Bee+lqwZ2GIJKmMfy8tQQcn7oCaZS5xe9
                                L/D8KQzPzZb2CMrCnG/usZ+LkwYuN3bOK12B1hk1ji5
                                fNN6aExyhw+hiV9ibobzqRe5MMxxaaDIf6scVlc= )
groupX.odslab.se.       120 IN DNSKEY 256 3 8 (
                                AwEAAaxv0uywTp5kIaw/fwPyQxcY0SHMn3701cz5GCx
                                velNQKlhlhm+tV/1TvkMd5GHq/ebjTPSR4mqA/jTu7CH
                                /iNhprxdnhJlVN7PjPpC5trDfFiHyCM97qRA+4lnNmlR4
                                S2JRlqCGmeoiU2BP2uyTiv3lKJPMnU8GmsP2TX8fi3LV
                                ) ; key id = 44494
groupX.odslab.se.       120 IN DNSKEY 257 3 8 (
                                AwEAAc4Wk/OqaEMaytXWL3y2510I8UuuNnkrufaJEERw
                                nlOEHaNGMscp5I52G7dcB6L70DJ84GE9hA4k8mbcBNPA
                                V1OCQVzlkP2TNt45XzYQTyaQJyobQdFtVqRTXtaFP1FP
                                S7nz7qaB/HVW8VNRp4H51ajaqh4LCX+39ftJX+rk6l3R
                                tbnWVv2PC01uZNFq2LOkbs0tKRb14UvoRQ15q+tjV/ow
                                cUknRtljQGPpYe/kLImU7+MrftnY6m8jvq0+qhd2o/12
                                6XScVBuDH+OGchFcU72HwwKfQHMtCuG2hxWOoOymPcDJ
                                12GNkBqqj2ECu/4Kr44DWTu4q2axd7dDOfSyKqM=
                                ) ; key id = 62246
groupX.odslab.se.       120 IN RRSIG DNSKEY 8 3 120 20110628103715 (
                                20110620083552 62246 groupx.odslab.se.
                                9w32PON95e8kqOfYxyKuJnDQN7dAELxVhq4kVoRA2RKx
                                iAgkIk/XQROfozjd/qKNjrIAJ+a9wxrvLWokRR8xzSTR
                                bwxl99Mu8Xj9p9QBCbzCvbvHPtRqFgf6Mto9jjlUa2K4
                                NlNQNq/qfaLvkvxKpdE4g9XacJb7lTPuyiQSovvARRDv
                                4rJ4znRdomdQHjtwCuQ4GeVfpgKqFCqaRHFK8D2GKnjk
                                56a7rbeGUNt5hNMjQfys3NfvulFKdC7W0Rb1kas7fQMw
                                j6mzmxRS8Sj+1lRbzkuGRwxVxWF/ca93fux4NhyOOGRP
                                2nWTBlwk+XcybA6vK5cfsrWTCSSoJ0t4+g== )
groupX.odslab.se.       60 IN NSEC3PARAM 1 0 5 3A58F749D1330DE3
groupX.odslab.se.       60 IN RRSIG NSEC3PARAM 8 3 60 20110628103502 (
                                20110620083552 44494 groupx.odslab.se.
                                GvylAOrm6dENvVUke1Ck3RmjR5NImhvIsFdvm2pJMfIa
                                nagGJhJDsT6R3jly8lvo+4T3jADSNGpvr6IL1nWySFRb
                                beEAn/sDt06GN3YsUGemv5iAh/YRboTqBUNtokm17Ada
                                5rZl87oOo3yqQO5qRSTVo8wCcP1HSG+htEt+vQs= )
www.groupX.odslab.se.   60 IN CNAME nsx.odslab.se.
www.groupX.odslab.se.   60 IN RRSIG CNAME 8 4 60 20110628103414 (
                                20110620083552 44494 groupx.odslab.se.
                                RAx7KPVdwoPeC9isn/N08dV2GR62s2jbQG45r6h8EOGF
                                7oRqd6wRpd8ChN3RrJNn7ycHGlm2j71WhEO8fsMLAlT6
                                vaGRVcK6IeE+7Vps4bqnk93jqBf3TftaiR22bVN1+Y9Q
                                Y7PWNFcmZOFmoqVmlkvLJdpn+NNjUJ5a+RlwojO= )
7oreblab9elhfqfp53bqqde6bcdn5eo3.groupx.odslab.se. 60 IN NSEC3 1 0 5 3A58F749D1330DE3
OTAHAR8MKJ8O0QC2G6K2IT2GU2SR4DCA CNAME RRSIG
7oreblab9elhfqfp53bqqde6bcdn5eo3.groupx.odslab.se. 60 IN RRSIG NSEC3 8 4 60
20110628103552 (
                                20110620083552 44494 groupx.odslab.se.
                                azU2yBaLQNKANwy7xosI4hwf6IPfV5XRNdPtQaGprShE
                                w4N/aDG9QsMJjlQrFW82rf2SV17mOJMBGdfsGVBZJJQ4
                                nXRmwnjT5Grn9k/aDhyOnYYANsoq4ikV5fLDYrH8af/u
                                zvoFs90vJ1M4CMbRNJrNBmCmIRzps/kD/aA+Ow= )
otanaronkjb0Sqc2q6k2it2qu2ab4doa.groupx.odslab.se. 60 IN NSEC3 1 0 5 3A58F749D1330DE3
7GRER1SR9ELHPQFF5I8QQOD6BCDMSEO3 NS SOA RRSIG DNSKEY NSEC3PARAM
otanaronkjb0Sqc2q6k2it2qu2ab4doa.groupx.odslab.se. 60 IN RRSIG NSEC3 8 4 60
20110628103526 (
                                20110620083552 44494 groupx.odslab.se.
                                QLlN/6Cj1kO6O8O9/AntqRFHNAEJ8POIS53HOZfN9OGP
                                P2Er/7dd+jlv2sgXmIYx/0VXySr4Bafqm8+kdfwEU+JY
                                7jafkLUOO4O8DCQ/BqHtLp5HF86TLMZa07VdPr9vE2ql
                                5UIUQjIPT2+aQRJDd/QMq26yskGqOApSR/wkq6Y= )
```

Signed zone file

DNSSEC brings new dimension to DNS

Past: Goofing up –> your secondaries would keep you up

DNSSEC: not doing anything means going out of existence

Past: static

DNSSEC: constant maintaining the zone.

Past: Apart from SOA, individual set of records

DNSSEC: denial chain, key trust chain

Reasons for signing software

- ✘ Signing is pretty complex;
- ✘ Re-use signatures which are not about to expire;
  - Signing an entire zone is expensive
- ✘ Maintain the denial/NSEC chain

- ✘ Management of Keys;
  - Generating, backup, purging
  - Managing keys for multiple zones
- ✘ Exchanging one key for another
  - Also known as key rolling
  - Keeping into account caching

- ✘ Integration with
- ✘ Signing/ key rolling can break
  - You might “want” a point where you can monitor and validate

Types of signing software

I. Full zone signing without key management

   Quick and (dirty) unmanaged

II. Incremental signing and limited key management
   Keys need to be pre-prepared and no or limited rolling control
   But does re-use signatures

III. In-line / On-line / Edge signing

   Signing on-demand in nameserver
   Co-operating nameservers no not share signed answers
   Allows white-lies for negative answers

IV. Managed zone signing

   Key generation, rolling, and timings all under control of signer
   Permanent server process
   No manual handling of files or resource records.
   Balancing signing of multiple zones
   Ideal as hidden master (or even bump in wire)

NSEC3

Zone walking

NSEC makes it trivial to walk the entire zone and retrieve all information.

NSEC3 attacks this by not using the names

- but hashes of the names and the order of the hashes names
- hashes cannot (easily) be reversed

Zone walking

NSEC makes it trivial to walk the entire zone and retrieve all information.

NSEC3 attacks this by not using the names

but hashes of the names and the order of the hashes names

hashes cannot (easily) be reversed

mies.example.nl	83UM61VCG7A2NENBA2AFN6K4L17S6GAI
example.nl	D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB
noot.example.nl	DCUA8AFF2DKFRF6TSCN8BQOGH5JUJS6V
zus.jet.example.nl	FUJGD2ALTMG3LF71BDT1UM6Q60HBF5V3
aap.example.nl	GAL0S358A2AD56R25503JD3U6E956L3A
wim.example.nl	TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ

Zone walking

NSEC makes it trivial to walk the entire zone and retrieve all information.

NSEC3 attacks this by not using the names

but hashes of the names and the order of the hashes names

hashes cannot (easily) be reversed

mies.example.nl	83UM61VCG7A2NENBA2AFN6K4L17S6GAI
example.nl	D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB
noot.example.nl	DCUA8AFF2DKFRF6TSCN8BQOGH5JUJS6V
zus.jet.example.nl	FUJGD2ALTMG3LF71BDT1UM6Q60HBF5V3
aap.example.nl	GAL0S358A2AD56R25503JD3U6E956L3A
wim.example.nl	TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ

teun.example.nl	SCV91OGLOJU7K03UQQP7Q8V4MQD093VV

NSEC3 Resource Record

```
aaaaaaaaaaaaaaaaa.example.nl.  NSEC3 1 0 10 00 bbbbbbbbbbbbbbbbb.example.nl. A
```

This is a denial of existence Next SECure version 3 RR-type.

NSEC3 Resource Record

```
GAL0S358A2AD56R25503JD3U6E956L3A.example.nl. NSEC3 1 0 10  CAFEBABE
                                                TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ  A
```

This is a denial of existence Next SECure version 3 RR-type.

SHA-1 hashes of labels.

Any labels that hash to values between these two do not exist.

NSEC3 Resource Record

```
GAL0S358A2AD56R25503JD3U6E956L3A.example.nl. NSEC3 1 0 10  CAFEBABE
                                                    TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ  A
```

This is a denial of existence Next SECure version 3 RR-type.

SHA-1 hashes of labels.
Any labels that hash to values between these two do not exist.

As well as any other RR types other than A belonging to the label that hashes to aaaaaaaaaaaaaaaaa.example.nl. do not exist.

NSEC3 Resource Record

aaaaaaaaaaaaaaaaa.example.nl. NSEC3 1 0 10 00 bbbbbbbbbbbbbbbb.example.nl. A

- ✘ Algorithm: 1 = SHA-1
- ✘ Flags: 1 = Opt-Out
- ✘ Number of iterations: repeat hashing algorithm 10 times
- ✘ Salt: prepend "00" as value before salting

Iterations: between 1 and 100

Higher more resilience to attacks, but at compute cost on nameservers and resolvers

Attackers could pre-compute possible labels: rainbow tables

Add random salt to complicate this and change salt at some time

Zone walking

mies.example.nl	83UM61VCG7A2NENBA2AFN6K4L17S6GAI
example.nl	D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB
noot.example.nl	DCUA8AFF2DKFRF6TSCN8BQOGH5JUJS6V
zus.jet.example.nl	FUJGD2ALTMG3LF71BDT1UM6Q60HBF5V3
aap.example.nl	GAL0S358A2AD56R25503JD3U6E956L3A
wim.example.nl	TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ

Query "teun.jet.example.nl":

8BH04K7K6NIB5EB4MA4RDJOGGUIR1NGP

83UM61VCG7A2NENBA2AFN6K4L17S6GAI.example.nl. NSEC3 1 0 10 A 00
D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB

There is no label between 83UM61VCG7A2NENBA2AFN6K4L17S6GAI
and D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB

NSEC3 Resource Record

```
aaaaaaaaaaaaaaaaa.example.nl.  NSEC3 1 0 10 00 bbbbbbbbbbbbbbbbb.example.nl. A
```

This is a denial of existence Next SECure version 3 RR-type.

Flattened namespace

mies.example.nl	83UM61VCG7A2NENBA2AFN6K4L17S6GAI
example.nl	D1PPNLJMVBG8N4QSHKNMV9FUM3EA31OB
noot.example.nl	DCUA8AFF2DKFRF6TSCN8BQOGH5JUJS6V
zus.jet.example.nl	FUJGD2ALTMG3LF71BDT1UM6Q60HBF5V3
aap.example.nl	GAL0S358A2AD56R25503JD3U6E956L3A
wim.example.nl	TOGHAEMJ4UL7C6PCNIQN0KIPQ854NKIJ

Querying "teun.jet.example.nl"

Answer loses information on "jet.example.nl".

Wildcards could reside on higher levels.

Therefore NSEC3 requires upto 3 records to provide a negative answer.

Flattened namespace

example.com	SOA …
1.1.example.com	TXT “One”
3.3.example.com	TXT “Three”

Looking up 2.2.example.com.

The closest enclosure: start chopping labels on the left until you hit a name that exist.
Which in this case is “example.com”.

The Next closest name would be the name with one less label chopped of.
Which in this case is “2.example.com”

NSEC3 wildcard denial

example.com	SOA …
1.1.example.com	TXT “One”
3.3.example.com	TXT “Three”

A resolver would be given NSEC3 of the MATCHING closest encloser

- This will be a possible place where a wildcard may be placed
- And proves “example.com” exist

NSEC3 wildcard denial

example.com	SOA …
1.1.example.com	TXT “One”
3.3.example.com	TXT “Three”

A resolver would be given NSEC3 of the MATCHING closest encloser

➤ This will be a possible place where a wildcard may be placed

➤ And proves “example.com” exist

A resolver also gets the NSEC3 that COVERS the next closest encloser

➤ This proves there is no “2.example.com”

NSEC3 wildcard denial

example.com	SOA …
1.1.example.com	TXT “One”
3.3.example.com	TXT “Three”

A resolver would be given NSEC3 of the MATCHING closest encloser

➤ This will be a possible place where a wildcard may be placed

➤ And proves “example.com” exist

A resolver also gets the NSEC3 that COVERS the next closest encloser

➤ This proves there is no “2.example.com”

Additionally a NSEC3 denying COVERING the wildcard at the depth of the closest encloser

➤ This proves there is no wildcard at this level.

NSEC3PARAM

NSEC3PARAM 1 0 10 1A2B3C4D5E6F

- ✘ 1: hash algorithm (SHA1, only)
- ✘ 0: flags
- ✘ 10: iterations
- ✘ Salt (length)

Needed by authoritative nameservers to compose NSEC3 results

Opt-Out

Useful for large zones with many delegations (NS records).
Where most delegations refer to unsigned zones (no accompanying DS record).

No (reel) need to include them in denial chain (their are not secure anyway).
Saves compute time on signing zone.

White lies

Not part of DNSSEC.

Only possible with inline/edge signing.

Signatures are creating on the fly.

Instead of creating NSEC3 record for next label that does exist, create it for label that could exist.

All DNSSEC record types

We have now almost exhausted all current DNSSEC record types.

Here's the full list:

RRSIG	Resource Record SIGnatures
DNSKEY	KEY record set
DS	Delegation of Signing
NSEC	Next SECure record; denials of existence
NSEC3	Next SECure version 3
NSEC3PARAM	Salt and Iteration parameters used in NSEC3
CDS	Child DS
CDNSKEY	Child DNSKEY

OpenDNSSEC

OpenDNSSEC

OpenDNSSEC is a zone signer that automates the process of keeping track of DNSSEC keys and the signing of zones

- ✘ Many DNSSEC Tools missing
  - key management
  - policy handling
  - hardware acceleration
- ✘ DNSSEC should be easy to deploy
- ✘ Increase DNSSEC users and add experience from previous deployments

OpenDNSSEC goals

- ✘ Open Source software with BSD license
- ✘ Simple to integrate into existing infrastructure
- ✘ Key storage and hardware acceleration using PKCS#11
- ✘ Anticipate OpenDNSSEC deployed between hidden and public master invisible as bump-in-the-wire.

Helicopter architecture

- ✘ Key and Signing Policy (KASP)
- ✘ Enforcer (handling policy, key handling)
- ✘ Signer (signing the zone based on enforcer instructions)
- ✘ HSM (hardware accelerated PKCS#11 crypto, or SoftHSM)

Key and Signing Policy (KASP)

- ✘ How to sign a zone is described by a policy
- ✘ Allows choice of key strengths, algorithm, key and signature lifetimes, NSEC/NSEC3, etc.
- ✘ One policy per zone, or some zones sharing policies

Enforcer

- ✘ Management of Keys (manages creation, disposal)
- ✘ Keeps track of timing and key rolling
- ✘ Select which keys need to sign
  - Informs signer through “signconf” how to sign the zone
- ✘ Handles changes in policy

Signer

- ✘ Automatic signing of the zones
  - can re-use signatures that are not too old
  - can spread signature expiration over time (jitter)
- ✘ Maintains NSEC/NSEC3 chain
- ✘ Updates SOA serial number

Hardware Security Module

- ✘ Performs actual crypto
- ✘ Therefor keys only life inside HSM
- ✘ Often security must, but also a good abstraction

Everything can be rebuild – except your keys

Hardware Security Module (HSM)

Hardware Security Module

- ✘ Performs actual crypto
- ✘ Keys only (need to) life inside HSM
- ✘ Often security must, but also a good abstraction
- ✘ Handles backup and replication

Everything can be rebuild – except your keys

Sideline: SoftHSM

- ✘ PKCS#11 is the industry API to HSMs;
- ✘ Private keys are assumed to be stored in HSM;
- ✘ The SoftHSM is distributed separately and independently from OpenDNSSEC (but in co-operation);
- ✘ for those who do not have HSM (most users);
- ✘ HSMs can be tricky, read documentation about capacity, backups, sessions;

SoftHSM

http://dnslab.uk/

Key Rollover

Key rollover

Rollover is the procedure to change from using one key to another

- ✘ Crypto improves over time
- ✘ (crypto works better when changing keys)
- ✘ Keys may still be stolen/compromised
- ✘ Fire drills happen for a reason

Keys should be rolled from time to time.

ZSK rollover: every 1–3 months

KSK rollover: once a year

Key rollover dangers

You must maintain a valid path

1. As long as there are signatures that rely on a key, this key should to be in the DNSKEY set

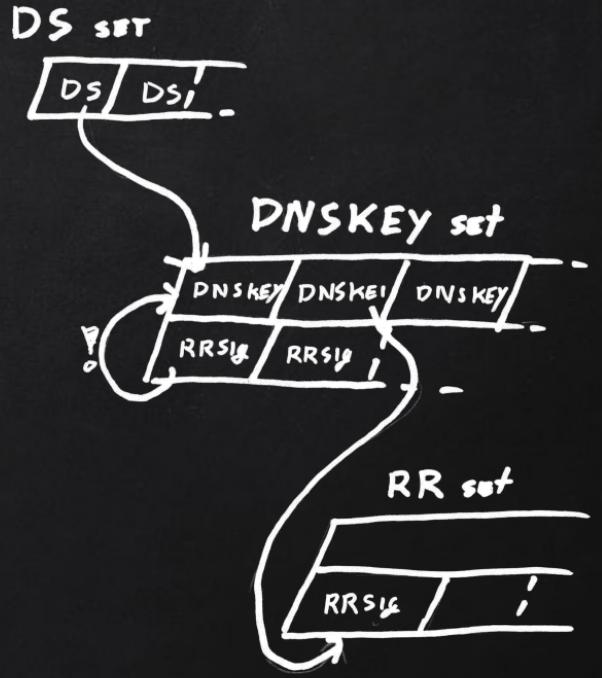
2. Signatures and DNSKEY set records are cached separately
   Worst case scenario:
   – resolver has just retrieved a signature using old key
   – resolver no longer has DNSKEY set
   – resolver fetches upstream DNSKEY set

3. For a (cached) DNSKEY set; there must be a DS in the parent zone which
   Refers to KSK signing the DNSKEY set.

Key Rolling

Key rolling: exchange one key for another in a gradual manner so that validation will not break.

The chain of trust is retrieved using a series of individual queries, which may all be cached:



In rolling keys we need to take account of delays due to caching, propagation, clock skews, etcetera.

Keys are phased in, and slowly retired.

Lifetime versus Expiration

Signatures actively expire.
The time at which they are no longer valid is embedded in the signature.

There is nothing in the DNSSEC protocol regarding their validity over time.

Therefore we talk about useful *Lifetime* of a key.

Keys do not expire.

RFC-6781 DNSSEC Operational Practices.

Common concepts

KASP

Key And Signing Policy

Signing policy: how long are signatures valid, how often updated, timing to take into account.

Key policy: Algorithm, key size, lifetime

Both involve time periods which are related, hence settings bundled into a policy.

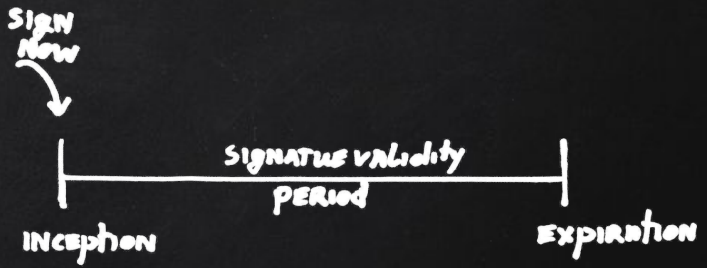
Validity period

How long signatures are valid

May be different for NSEC/NSEC3, DNSKEY and Resource Record sets.

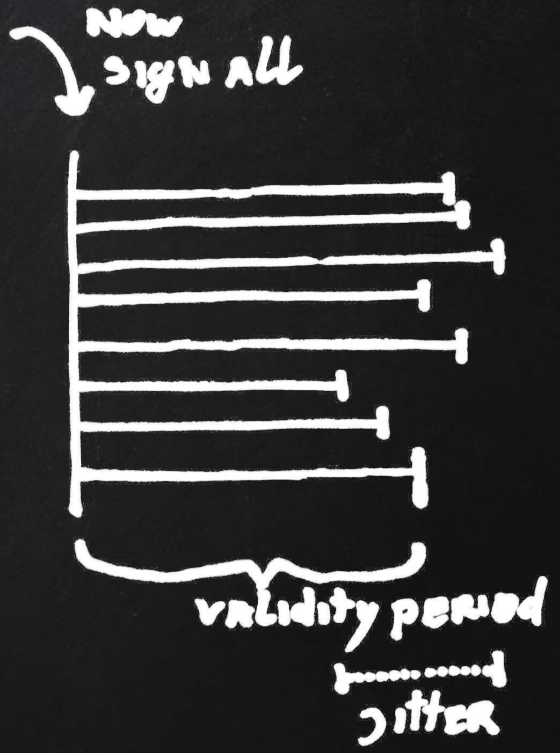
Start of validity period: inception

End of validity period: expiration



Jitter

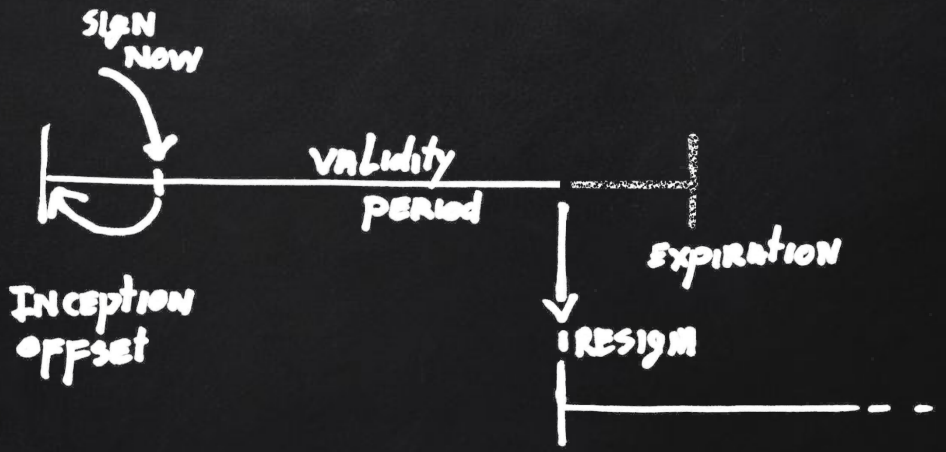
- ✘ Initially the signatures are all generated at the same time
- ✘ Signature would expire all at the same same time with the same Resign period
- ✘ Introducing random variation, Jitter, will spread signing over time



Inception Offset and Refresh period

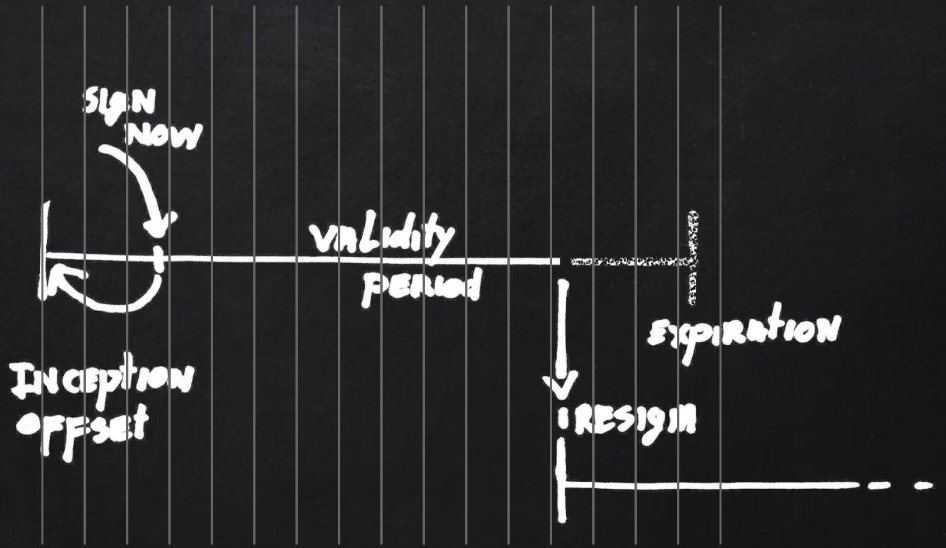
Inception offset: safety margin to allow for clock skew, etcetera

Refresh period: period before actual signature expiration to take into account



(Resign period)

Inspect zone file for changes and create new signatures
for updates and signatures that fall into their resign period.



Key states

Keys are introduced and can be replaced.
During their lifetime they will be in different states:

1. Generate
2. Publish
3. Ready (waiting for DS seen)
4. Active
5. Retiring
6. Revoked
7. Purge

Problem is that some of these are actions, others processes or states.

(Distribution states)

The resource records sets DS, DNSKEY, RRSIG(DNSKEY), RRSIG(RR) contain or are generated using a number of keys.

DS, DNSKEY and RRSIG are retrieved and cached separately.

The key state does not tell anything about the distribution of these RRs.

The signatures produced by an Active ZSK may not all have been propagated, so a Retiring key must be kept in place.

(Distribution states)

Therefore the rolling algorithm should take into account how the world sees these RRs:

1. Hidden:
   No-one can see this record (yet)
2. Rumoured:
   Some may have seen this RR, others might have a predecessor of this RR in its cache.
3. Omnipresent:
   Everyone who has this RR or wants to retrieve it will get this up-to-date RR.
4. Unretentive:
   The signed zone at the source no longer contains the outdated RR, but it may still be in caches or at secondaries.

ods-enforcer key list -d

Publish and Retire safety

Most key rolling mechanism require keys to be published first.

Publishing means adding them to the DNSKEY set without using them to generate signatures with (i.e. not Active).

Some time is required to get the DNSKEY set from being rumoured to being omnipresent.

Publish, Publishing, Published

The minimal required time it o.a. TTL of DNSKEY.

Additional time to take into account for external events (like failures).

SOA serial policy

In order to keep zone valid, signatures must be updated.

This means zone must be updated.

Secondaries will only pick up updated zone if SOA serial increments.

In absence of input updates the signing software should increment the SOA serial number.

SOA serial Number often follows certain semantics:

1. Counter
2. Datecounter (YYYYMMDDcc)
3. Unix timestamp (seconds since epoch)
4. Keep (take input SOA serial, requires regular updates)

This means that output serial will mismatch input serial (but still needs updating).

Responsibility

Signer responsibilities

- ✘ Update SOA serial number
- ✘ Maintain NSEC/NSEC3 chain, Opt-out
- ✘ Produce signatures for updates
- ✘ Produce signatures for expiring signatures
- ✘ Balance producing signatures over time and zones
- ✘ (Produce white lies)
- ✘ Accept updates from upstream
- ✘ Inform downstream of updates
- ✘ Make sure using right keys.
- ✘ Protect input zone against errors for DNSSEC (TTL, DNSSEC records)

Key manager/enforcer responsibilities

- ✘ Maintain key pool and/or create keys on demand
- ✘ Manage key backup
- ✘ Proceed key states through its lifetime according to timings
- ✘ Purge old keys from HSM
- ✘ Inform signer on key updates, and signing method
- ✘ In case of NSEC3 maintain salt, possibly re-salting.
- ✘ Assist updating DS record
- ✘ Maintain zone list
- ✘ Notify rollovers / provide insight key roll progress

OpenDNSSEC Configuration

Configuration of OpenDNSSEC

Very configurable

- conf.xml -- used for overall configuration of the system
- kasp.xml -- defines the various policies for signing zones
- addns.xml -- defines In- and Output adapter parameters (e.g. TSIG).

Time definition in OpenDNSSEC

"P1M3DT1H30M10S" ISO8601

P[n]Y[n]M[n]DT[n]H[n]M[n]S

- ✘ Configuration of OpenDNSSEC about durations (periods) not absolute times.
- ✘ No clue about Gregorian Calendar (P1Y == P365D)

Pitfall: P5M PT5M

A world of difference (actually 13391700s).

CONF.XML

Configuration contains

RepositoryList (which HSMs)

Common

Enforcer

Signer

```
<Configuration>
    <RepositoryList>
      …
    </RepositoryList>
    <Common> … </Common>
    <Enforcer> … </Enforcer>
    <Signer> … </Signer>
</Configuration>
```

Repository List

Defines where private keys live

- ✘ You need at least one but can have more (separate ZSK/KSK)
- ✘ HSM interface available
- ✘ Each private key repository is listed as an <repository> element
- ✘ SoftHSM if you do not have the ~~money~~ hardware

```
<RepositoryList>
  <Repository name="SoftHSM">
     <Module>/usr/lib/libsofthsm2.so</Module>
     <TokenLabel>OpenDNSSEC</TokenLabel>
     <PIN>1234</PIN>
  <Repository>
```

Having multiple Repositories

Off-line KSK when not needed

Place signing of keyset (using KSK)

And regular zone signing (ZSK) in different locations

- Long validity period signatures keyset
- Zone needs updating -> short period remainder

Configration conf.xml highlights

Enforcer/Datastore

Enforcer/ManualKeyGeneration

Enforcer/AutomaticKeyGenerationPeriod

Enforcer/RolloverNotificationPeriod

Enforcer/DelegationSignerSubmitCommand

Signer/Threads

Signer/NotifyCommand

KASP

Values in default policies are sane starting values

Signatures/Resign
Signatures/Refresh
Signatures/Validity/Default
Signatures/Validity/Denial
Signatures/Jitter
Signatures/InceptionOffset
Signatures/MaxZoneTTL

Denial/NSEC3/OptOut
Denial/NSEC3/Resalt
Denial/NSEC3/Hash

Keys/TTL
Keys/RetireSafety
Keys/PublishSafety
Keys/Purge
Keys/KSK/Lifetime

Zone/PropagationDelay
Zone/SOA/TTL
Zone/SOA/Minimum
Zone/SOA/Serial

Zone/PropagationDelay
Zone/SOA/TTL
Zone/SOA/Minimum
Zone/SOA/Serial

Parent/PropagationDelay
Parent/DS/TTL
Parent/SOA/TTL
Parent/SOA/Minimum

Using OpenDNSSEC

http://labs.opennetlabs.nl/

Key Rollover

Rolling methods

Different types of roll:

- ✘ Double signing (larger responses)
- ✘ Pre-publishing key (more interaction)

ZSK: preferred method pre-publication

KSK: preferred method double signature

Rolling means changing DS record that is not under direct control

Rollover times depends on TTL and method

Rolling involves

- ✘ Publishing new key in DNSKEY
- ✘ Removing old key in DNSKEY
- ✘ KSK: publishing DS for new key
- ✘ KSK: removing DS for old key
- ✘ Signing appropriate RRs with new or both keys
  - KSK: only DNSKEY
  - ZSK: entire zone, lot of signatures
- ✘ Planning generation, purging of keys

When to do what depends on KASP:

- When are keys supposed to replaced
- When can you expect your changes to be propagated

Key Rollover methods

Know the state of your keys and signatures

1. Hidden
2. Rumoured
3. Omnipresent
4. Retentive

1.x compatible vs. 2.0 key list

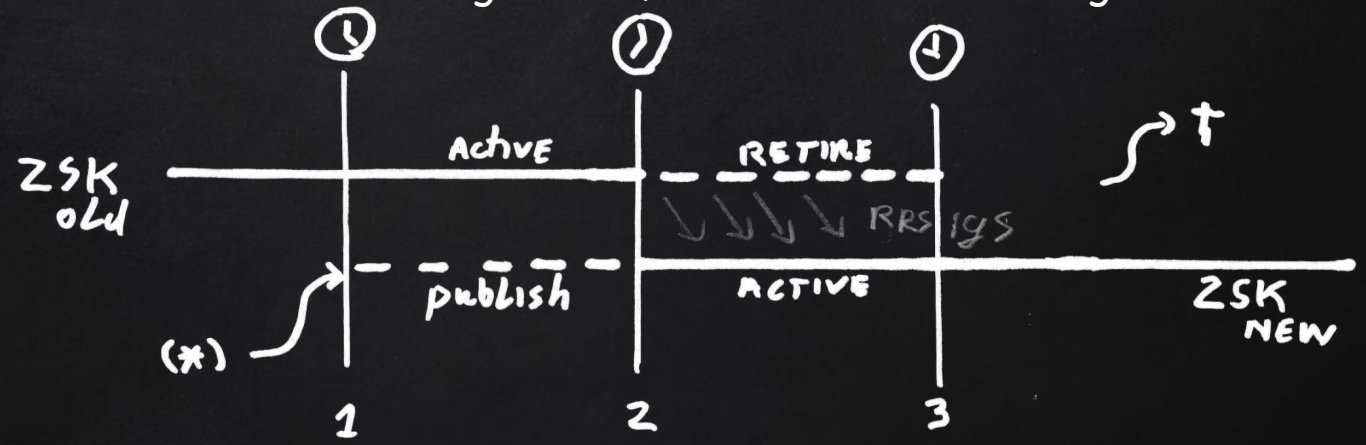
```
$ ods-enforcer key list
Zone:        Keytype: State:  Date of next transition:
example.com  KSK      publish 2016-04-15 00:22:18
example.com  ZSK      ready   2016-04-15 00:22:18


$ ods-enforcer key list -d
Zone:        Key role: DS:    DNSKEY:  RRSIGDNSKEY: RRSIG:      Pub: Act:
example.com  KSK       hidden rumoured rumoured     NA           1    1
example.com  ZSK       NA     rumoured NA           omnipresent 1    1
```

Pre-publication

First make sure the key is known first.

- Generate / allocate key
- Place it in DNSKEY set
- Wait
- Active new key / deactivate old key
- Start producing new signatures and wait until all old are gone
- Remove old ke from DNSKEY set

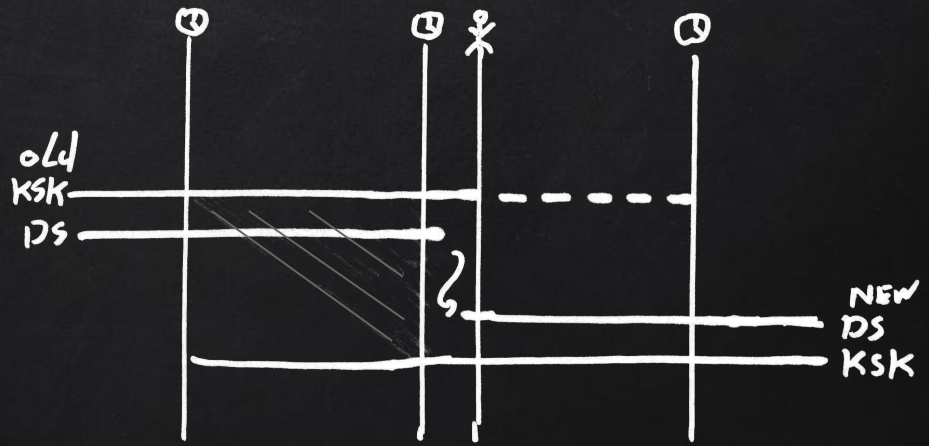
Period when some RR sets have old signatures, and others have new signatures



Double signature roll

Make sure signatures for old and new key remain known until no longer needed

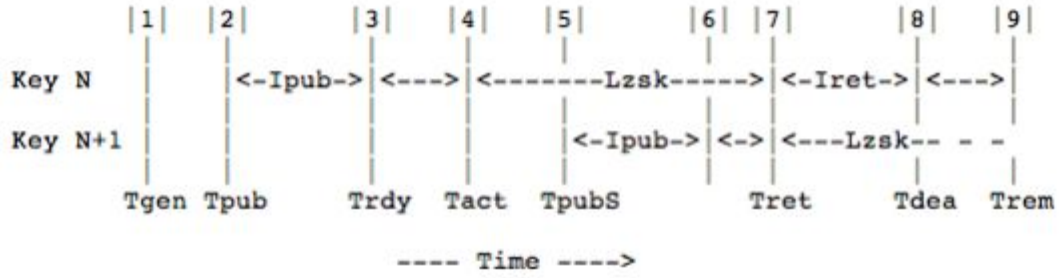
- Generate / allocate key
- Place it in DNSKEY set
- Start signing DNSKEY set with old and new keys
- Wait
- Remove old and insert new DS and ensure it <u>can</u> be seen
- Wait
- Remove old KSK



ZSK Method	KSK Method	Description
Pre-Publication	N/A	Publish DNSKEY before the RRSIG
Double-Signature	**Double-Signature**	Publish DNSKEY and RRSIG at the same time. For a KSK, this happens before the DS is published
Double-RRSIG	N/A	Publish RRSIG before the DNSKEY
N/A	Double-DS	Publish DS before DNSKEY
N/A	Double-RRset	Publish DNSKEY and DS in parallel.

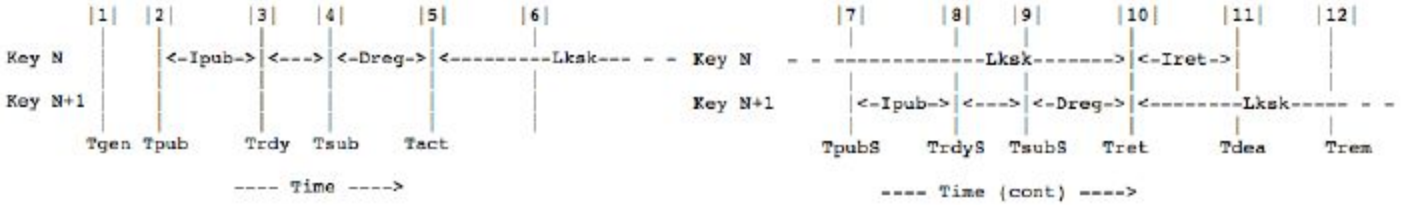
Rollover methods

Pre-Publication ZSK rollover



- First key: Ipub = Dprp + min(TTLsoa, SOAmin)
- Future keys: Ipub = Dprp + TTLkey
- TpubS <= Tact + Lzsk - Ipub
- Iret = Dsgn + Dprp + TTLsig

Double-Signature KSK rollover



- Ipub = Dprp + TTLkey
- TpubS <= Tact + Lksk - Dreg - Ipub
- Iret = DprpP + TTLds

Miscellaneous

Monitoring

OpenDNSSEC, NSD, Bind, all are stable, but integration will break:

- ✘ Signer up and running
- ✘ Signature expiration nearing unexpectedly
- ✘ Zone updates get through

Prepare for when things to wrong

- ✘ Backup not just keys, also var/lib/opendnssec and kasp.conf

CDS and CDNSKEY

RFC8078: Managing DS records from the parent via CDS/CDNSKEY

1) A child zone publishes the DS records it wants to have published in the parent zone as CDS records in its domain.
2) Parents actively monitor for CDS (or CDNSKEY) records in the child delegations.
3) They adopt changes (with some safeguards).
4) Child zones monitor whether their changes have been accepted and then make the changes to the DS records and remove the CDS records.

Requires on-line monitoring of zones

dnssec-dsfromkey -C example.dnslab.uk
  To generate CDS entries
dnssec-cds
  To adopt entries

Rollover specials

- ✘ Emergency roll-over (rollover when in rollover procedure)
- ✘ Modifying timing and Key parameters in live environment
- ✘ Algorithm rollover
  - KSK and ZSK roll combined in one
  - Requires signatures to be published before DNSKEY

Signing ceremony

Keeping KSK in air gapped, bunker environment

Prepare sequence of signed DNSKEY sets for a specified period of time.

https://github.com/NLnetLabs/dnssec-ceremony-doc/

Migrating

- ✘ Exporting keys and importing them
- ✘ Publish DNSKEY in old sign environment
  - Double RRset / Double DS
- ✘ Go insecure