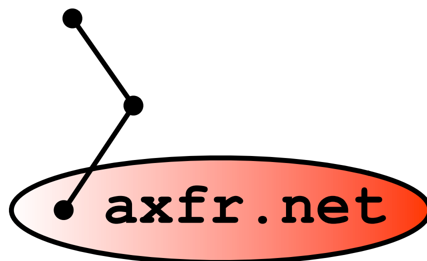


DNSSEC Tutorial*

© 2002—2016 Firma Johan Ihrén

Version v5.98 (**axfr.net**, v1.16)

Johan Ihrén
dns-training@axfr.net

September 19, 2016

Abstract

The lecture notes for the DNSSEC Special Course contains all the information from the slide presentation but formatted in a slightly more compact format and extended with an index at the end.

* main.tex, revision: 1.16 on September 19, 2016 by johani

Contents

1 Fundamentals **4**
1.1 DNS Wire Protocol 4

2 DNS Implementations **7**
2.1 NSD and Unbound 9

3 Lab Environment **10**

4 Recursive Server Configuration **12**
4.1 The `root.hints` File 12
4.2 Unbound Configuration 12
4.3 Nameserver Logging 17

5 `dig` **18**

6 Authoritative Server Configuration **19**
6.1 Zone File Format 19
6.2 NSD Configuration 23
6.3 Query Logging 27

7 DNS Theory #3 **29**
7.1 Zone Transfer 29
7.2 Slave Server Configuration 30

8 DNS Security **31**
8.1 DNS Vulnerabilities Overview 31

9 EDNS(0): Extended DNS **32**
9.1 Background 32

10 DNSSEC **34**

10.1 “Cache Pollution” 34
10.2 Data Protection 38
10.3 Creating a Signed Zone 48
10.4 Key Management and Key Rollovers 58
10.5 DNSSEC Debugging 66
10.6 DNSSEC Operations 71
10.7 Autenticating Negative Responses 72
10.8 DNSSEC Policy Issues 80

11 DNSSEC for Applications **80**

11.1 DANE 82

12 DNSSEC Tools **86**

12.1 OpenDNSSEC 88
12.2 Updating Trust Anchors 92

Index **98**

Publication of Your Configs

- Over the next few days you will be doing quite a lot of configuration work. After the course, all your configurations will be published at the following URL:

  `http://www.axfr.net/student-data/dnssec-lacnic-sep2016/`

- The URL is not linked to from the **`www.axfr.net`** homepage, so the only (reasonable) way of finding it is by using the URL above directly.
- There is no authentication and each labgroup's configs will be made available as a plain "**`tar`**" archive.

1 Fundamentals

1.1 DNS Wire Protocol

"RRset"

- An RRset ("Resource Record set") is the set of records that share the same owner, the same class and the same type
  - i.e. the RRset is the set of records that all match the same query
- Because of the DNS coherency requirement it is never allowed to split an RRset during transport
  - i.e. in the answer **all** records that match the query must be sent
- When we get to DNSSEC later on the RRset concept will be central to the design

The DNS Wire Protocol

- The DNS protocol comprises a message pair: an outgoing message (usually a query) and the corresponding response
  - Port 53, UDP, or TCP, are always used to receive messages

- The most common message is the Query, which is used for the entire name look-up activity
- There are other message types, notably
  - Notifies: covered later in this course
  - Dynamic Updates: covered in the Advanced Course

The Structure of a DNS Message

- DNS messages always consists of five parts
  - **Header**, **Query**, **Answer**, **Authority** and **Additional**
  - i.e. both a "Query" and the response to the Query have the same structure
- in fact, also DNS messages other than Queries and Responses to Queries are structured into five parts
  - but then the parts are interpreted differently

Header
Query
Answer
Authority
Additional

Wire Protocol: Header (12 octets)

- Present in all types of DNS messages
  - Queries, responses, and also other types (such as "Notifies" and "Updates" covered elsewhere)
- Contents:

`id:`	a “unique” identifier (16 bits)
`flags:`	recursion desired, recursion available, etc.
`opcode:`	Query, Status, Notify, Update, . . .
`rcode:`	NOERROR, REFUSED, FORMERR, . . .
`qrcount`, `ancount`, `aucount`, `adcount`:	counters for the number of records in each of the following sections

Wire Protocol: Header, cont'd: Flags

QR	Opcode	AA	TC	RD	RA	Z	AD	CD	Rcode

QR	If set, this is a response
AA	If set, Authoritative Answer
TC	If set, answer is truncated
RA + RD	Recursion Available + Recursion Desired
Z	Reserved, always 0
AD	Authenticated Data (used by DNSSEC)
CD	Checking Disabled (used by DNSSEC)

RCODEs

- There are 16 possible RCODEs of which the most common are:

`NOERROR`	No Error (Ta-da! :-)
`FORMERR`	Format Error
`SERVFAIL`	Server Failure
`NXDOMAIN`	Non-Existent Domain Name
`REFUSED`	Server cannot respond due to its configuration

- Note that there is a difference between an NXDOMAIN response and a NOERROR with zero records in the Answer section

    – In the former case the domain name does not exist, while in the latter the domain name exists, but does not have the requested RR type

Authority Section

- The Authority section has three different uses, depending on situation
    - if the response (i.e. this message) is **an answer**, then this section contains the NS records for the authoritative name servers for the zone where the answer is located
    - if the response is **a referral** (from a parent to a child) then this section contains the NS records that comprise the actual referral
    - if the response is a **negative answer**, then this section contains the `SOA` record of the zone where the answer should have been found if it had existed

A Typical Response

```
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 4
;; flags: qr aa rd ra; QUES: 1, ANS: 2, AUTH: 2, ADD: 2

;; QUESTION SECTION:
;;    www.axfr.net, type = A, class = IN

;; ANSWER SECTION:
www.axfr.net.       172800 IN CNAME odie.axfr.net.
odie.axfr.net.      172800 IN A 213.115.163.155

;; AUTHORITY SECTION:
axfr.net.           172800 IN NS idefix.johani.org.
axfr.net.           172800 IN NS ns.axfr.net.

;; ADDITIONAL SECTION:
ns.axfr.net.        86400 IN A   213.115.163.156
idefix.johani.org.  86400 IN A   192.71.80.122
```

2 DNS Implementations

DNS Implementations

- Throughout all the DNS courses we try to distinguish between DNS as a protocol, DNS as a database and DNS implementations
  - while the first two topics are generic, the configuration details will always be particular to each implementation
- On the following slides some of the more interesting implementations are listed
  - requirements for being listed included that the implementation **must** be **open source**, and fully support **DNSSEC** and **IPv6**
- During the course we will describe and use the following:
  - authoritative: NSD4
  - recursive: Unbound
  - also of interest are Knot-DNS (authoritative), Knot-DNS Resolver (recursive), PowerDNS (authoritative), PowerDNS Recursor (recursive), BIND9 (authoritative+recursive), and Yadifa (authoritative)
- In the labs you must choose what software to use for recursive service and what to use for authoritative service
  - any combination is ok

DNS implementations, cont'd: Authoritative Servers

Server	**“Status”**
NSD4	Designed as the ultimate slave server. No DDNS support. Dynamically reconfigurable. Developed by NLNetLabs
Knot-DNS	Small and fast, support for DDNS, YAML-based config in v2.0+. Developed by `NIC.CZ` (i.e. the `.CZ` registry)
BIND9	Giant monolith, by default both authoritative and recursive. In very wide use. Complex pseudo-C config language
PowerDNS	SQL database backend, popular with dns hosting providers
YADIFA	High performance (very fast), support for DDNS, XML based config language. Developed by EurID (i.e. the `.EU` registry)

DNS Implementations, cont'd: Recursive Servers

Server	"Status"
Unbound	Highly configurable and flexible (more so than BIND9). Config language based on attributes and values. Popular with many ISPs and in very wide use. Developed by NLNetLabs
Knot-DNS Resolver	Highly configurable and flexible. Modular. Scriptable config language based on Lua. Developed by `NIC.CZ`
PowerDNS Recursor	The recursive companion to PowerDNS. also scriptable in Lua. SQL database backend, quite popular
BIND9	Giant monolith, by default both authoritative and recursive. In very wide use

- There are fewer recursive servers (due to being more complicated and also that most funding comes from TLD registries that primarily care about authoritative service)
- Today, we consider Unbound to be the primary choice

2.1 NSD and Unbound

NSD: Name Server Daemon

- NSD ("Name Server Daemon") was initiated at RIPE NCC as a project to develop an **authoritative only** name server
  - because they believe that is a better design
- NSD development is now managed by NLNetLabs, see `http://www.nlnetlabs.nl/` and current release version is NSD 4.1 which is referred to as "NSD4"
- Several root name servers run NSD (at the moment they are still using various versions of NSD3)
  - at least `h.root-servers.net`, `k.root-servers.net` and `l.root-servers.net`
- NSD is **much** simpler than BIND9 and much faster

  - the speed is largely due to pre-compiling sections of the responses via a "zone compiler"
  - the speed of an authoritative server is usually not an issue, though (but complexity is...)

Unbound: A Recursive-Only Nameserver from NLNetLabs

- Unbound is the recursive server companion to the NSD authoritative server
  - see `http://www.unbound.net/`
- NSD4 and Unbound share the same configuration "language"
  - although as a recursive server is much more complicated, Unbound has many more options
- Unbound has full support for everything from DNSSEC and IPv6 (including synthetization of DNS64 responses), to complex **`forwarding`** and **`stub`** configurations.
  - (don't worry if you don't know what those things are)
- Many ISPs have switched to using Unbound (typically from BIND9), because of greater configuration flexibility in Unbound

Today Unbound is the recursive nameserver that we consider to be most mature, well-supported and fully featured.

3 Lab Environment

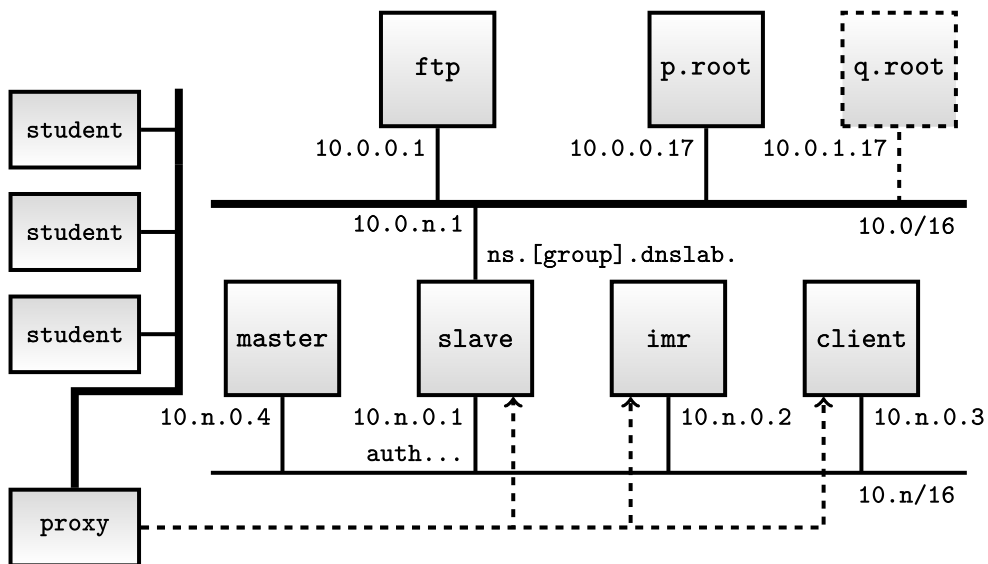
Lab Topology

- The lab exercises take place in a completely isolated network topology
- The lab environment is divided into "lab groups", where each lab group has a number (`1`, `2`, `3`, `...`) and a name (**`alpha`**, **`bravo`**, **`charlie`**, `...`)
- Each lab group will use several servers that will provide different services (recursive service, authoritative service, etc)
- The lab machines are **not** the machines in front of you!

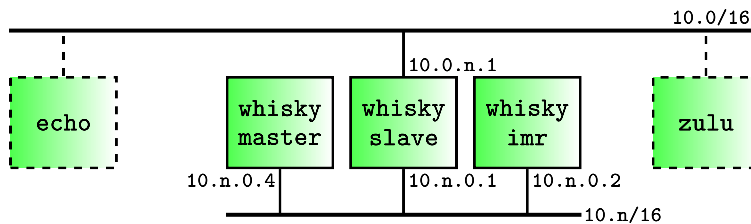
- to reach the lab machines it is necessary to login remotely to the IP address provided for remote access

- Examine the network topology drawing carefully

In the instructions group 23, **whisky.dnslab**, is used as an example. Replace with your own group name!



Your Servers in More Detail



- "**master**" will provide authoritative DNS service acting as a so-called "hidden master"

- "**slave**" (also known as "**server**") will provide authoritative DNS service on all interfaces (as a slave for zones transferred from **master**)

- "**imr**" will provide recursive DNS service

4 Recursive Server Configuration

4.1 The `root.hints` File

The `root.hints` File

- An iterative mode resolver (aka IMR or recursive name server) needs to know where to find the root name servers
- This is the purpose of the so-called root hints configuration
  - the hint file identifies the root name servers and their IP addresses
  - the hint file is **not** a zone file, because there is no **`SOA`** record
  - a good location for the hint file is in the configuration directory of the recursive name server
- Excerpt from the **`root.hints`** file:

```
.                      7200 IN NS a.root-servers.net.
.                      7200 IN NS b.root-servers.net.
.                      7200 IN NS c.root-servers.net.
...
a.root-servers.net.    7200 IN A 198.41.0.4
b.root-servers.net.    7200 IN A 192.228.79.201
c.root-servers.net.    7200 IN A 192.33.4.12
...
```

4.2 Unbound Configuration

Unbound Configuration

- The Unbound primary configuration is located in **`/usr/pkg/etc/unbound/unbound.conf`** (and the NSD configuration in **`/usr/pkg/etc/nsd/nsd.conf`**)
- The configuration file consists of **attributes** and **values**
  - the attribute keywords always have a trailing colon, “**`:`**”

```
directory:   /usr/pkg/etc/unbound
root-hints:  "root.hints"
```

an attribute

a value

- Quotes ("") on values are optional
- Some attributes have a set of "sub-attributes" as their value.
  - The sub-attributes are indented, i.e. they have white space to the left of the name.

```
server:
    directory:   /usr/pkg/etc/unbound
    root-hints:  "root.hints"
```

Unbound Configuration, cont'd

- The "**server:**" attribute is where global parameters are defined

```
server:
    directory:  /usr/pkg/etc/unbound
    logfile:    /usr/pkg/etc/unbound/unbound.log
    verbosity:  2
    interface:  10.23.0.1 # Listen on this
    interface:  20a1:b80:3f56::1035 # ...and this
```

- In practice, most of the Unbound configuration will be sub-attributes to the **server:** attribute
- See the **unbound.conf** documentation for all the sundry details

Unbound root hints Configuration

- As Unbound is a recursive name server it needs to know where the root name servers are located (to be able to do "priming").
  - this is achieved with the "**root-hints:**" attribute:

```
server:
    ...
    root-hints:  "root.hints"
```

Relative to `directory:`

- The hint information is used only at start-up to locate a root name server that is then queried for **current** data
  - this is called "priming"
- Only a **resolver**, i.e. a **recursive server**, needs a hint configuration

Access Control in Unbound

- By default, Unbound will only respond to queries from `::1` and `127.0.0.1` (i.e. "`localhost`")
  - this is often too restrictive, especially in a lab environment
- To relax this restriction the `access-control`: sub-attribute to `server:` is used:

```
#                      <IP block> <action>
server:
    ...
    access-control:    0.0.0.0/0   allow
    access-control:    ::/0        allow
```

- Note that **this particular config is only suitable for a lab environment!**

`unbound-checkconf` and `nsd-checkconf`

- `unbound-checkconf` is a stand-alone tool bundled with Unbound that does the same parsing and interpretation of the `unbound.conf` file that Unbound does when starting

```
# unbound-checkconf /usr/pkg/etc/unbound/unbound.conf
```

- `nsd-checkconf` does the same thing for NSD4
- See "**man unbound-checkconf**" and "**man nsd-checkconf**" for more details

4.2.1 The unbound-control Utility

unbound-control: Unbound Controller

- **unbound-control** is the control utility used to manage a running Unbound nameserver
- **unbound-control** is able to work remotely, and therefore it needs authentication (you don't want someone else to be able to issue a control command to stop the nameserver)
  - this is achieved via the **unbound-control-setup** utility, which must be executed **once** to set up certificates for authentication

```
# unbound-control-setup
```

  - in addition, the control interface must be activated in **/usr/pkg/etc/unbound/unbound.conf**:

```
remote-control:
    control-enable:  yes
```

The "unbound-control" Command, cont'd

- Some examples of **unbound-control** sub commands that can be sent to an Unbound name server:

start | stop	starting and stopping Unbound ("**start**" only works locally)
stats	show statistics of queries, cache hits, cache misses, etc
status	report status
verbosity	
flush name	flush common RR types (including **A**, **AAAA**, **NS** and **SOA**) at "**name**" from cache
flush_zone name	flush all data from "**name**" and below from cache
reload	re-read configuration and flush cache
lookup	look up name servers for name

Introduction to the "dig", "kdig" and "drill" Utilities

- "**dig**" is a stand-alone tool, distributed with BIND, for analysis and debugging of DNS data. It has similar functionality to the tools "**nslookup**" and "**host**"
  - "**kdig**" and "**drill**" are **dig** replacements. **kdig** is close to identical while **drill** has slightly different syntax. There is also a "**khost**" tool which replaces "**host**"
- In it's simplest form, the syntax for using **dig/kdig** (which uses the nameserver from the stub resolver config) is:

```
# kdig qname qtype +short
```

- Example:

```
# kdig www.dnslab a +short
10.0.0.1
```

- To query a specific nameserver, use "**@server**":

```
# kdig @::1 www.dnslab a +short
```

Send query to localhost over v6 transport

4.3 Nameserver Logging

Nameserver Logging

- Why log? There are several reasons:
  - To identify problems in the configuration
  - To locate misconfigured resolvers
  - To classify problems that a resolver may find in the behavior of nameservers belonging to others
  - As a general background check that everything is working
- Note that logging to search for mistakes is a complement but not a substitute for direct analysis with tools like **dig** or **nslookup**

Unbound Logging

- The logging infrastructure in Unbound is rather simplistic and to the point
- The primary mechanism is the `verbosity:` sub-attribute to the **server:** attribute, which controls the overall logging verbosity, i.e. how talkative the logging system is

```
server:
    verbosity:      1    # 1-5, higher => more talkative
    logfile:        /var/log/unbound.log
    use-syslog:     no   # default is "yes"
    log-time-ascii: yes  # default is "no"
    val-log-level:  1    # 0-2, default is "0" (off)
    log-queries:    yes  # default is "no" (off)
```

- The **val-log-level:** attribute specifies logging of DNSSEC validation issues, more on that in the DNSSEC part

5 dig

The "dig" Command #1

- One reason we prefer **dig** is that the output is more closely related to the actual protocol structure, thereby making it easier to see the actual details
  - another reason to use **dig** is that it provides greater and more robust functionality than **nslookup** (**nslookup** will eventually go away... the sooner the better)
- **dig** is part of the BIND9 distribution
  - and these days there seems to exist a binary distribution of just the "tools" part of BIND9, which makes installation much easier
  - although there exists some third party binaries of **dig** for Windows, we rather suggest just extracting the **dig.exe** binary from the Windows binary distribution

The "dig" Command #2

```
odie>dig odie.axfr.net. a
; <<>> DiG 9.1 <<>> odie.axfr.net. a
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTH: 2, ADD'L: 2
;; QUERY SECTION:
;;       odie.axfr.net, type = A, class = IN
```

The "dig" Command #3

```
;; ANSWER SECTION:
odie.axfr.net.          86400 IN A 213.115.163.165

;; AUTHORITY SECTION:
axfr.net.               86400 IN NS ns.axfr.net.
axfr.net.               86400 IN NS ns.johani.org.

;; ADDITIONAL SECTION:
ns.johani.org.          84732 IN A 192.71.80.122
ns.axfr.net.            86400 IN A 213.115.163.165

;; Total query time: 5 msec
;; FROM: odie.axfr.net to SERVER: default -- ::1
;; WHEN: Tue Feb 16 22:33:32 2016
;; MSG SIZE sent: 24 rcvd: 130
```

The "`dig`" Command #4

- Other flags of importance are:

`+dnssec`	indicate interest in receiving DNSSEC data
`+multi`	request pretty-printing of multi-line records
`+short`	request compact responses with only the requested data
`-y`	communicate keying information to the resolver (needed for instance when authenticating usingTSIG)
`-4 / -6`	lock down preferred transport (IPv4 or IPv6)

- Use "`dig -h`" to find out information about additional flags

6 Authoritative Server Configuration

6.1 Zone File Format

Master File Format

```
axfr.net.           IN SOA ns.axfr.net. hm.axfr.net. (
                       2016032001 ; serial
                       21600      ; 6h (refresh)
                       3600       ; 1h (retry)
                       604800     ; 7d (expire)
                       10800 )    ; 3h (neg. caching)
axfr.net.           IN NS     ns.axfr.net.
axfr.net.           IN NS     ns.johani.org.
axfr.net.           IN MX     10 mail.axfr.net.
ns.axfr.net.        IN A      192.71.80.122
ns.axfr.net.        IN AAAA   2a00:801:f:1::53
www.tech.axfr.net. IN A      213.115.164.155
```

Master File Format #2: Special Features

- Names that do not end with "`.`" (dot) automatically get a suffix (usually the name of the zone)
  - this happened to .SE on October 12, 2009 (the so-called "`.SE.SE` incident")...
  - bogaboo
- This suffix can be changed via the variable `$ORIGIN`:

```
$ORIGIN foo.axfr.net.
```

- "`@`" alone expands to the value of `$ORIGIN`
- Default `TTL` (earlier the last field in the `SOA` record) is controlled by the variable `$TTL:`

```
$TTL 7200
```

All the expansions are a function of a desire to be able to minimize the size of the zone during zone transfer

- i.e. the reason is **not** primarily to minimize typing

Master File Format #3

```
$TTL 86400
@            IN SOA ns hm (
                 2016032001 ;
                 21600      ; 6h
                 3600       ; 1h
                 604800     ; 7d
                 10800 )    ; 3h
             IN NS     ns
             IN NS     ns.johani.org.
             IN MX 10  mail
ns           IN A      192.71.80.122
             IN AAAA   2a00:801:f:1::53
$ORIGIN tech.axfr.net.
www          IN A      213.115.164.155
```

Zone Checker Tools: `validns`

Regardless of nameserver implementation, a correct zone file is obviously crucial. There are several zone checker tools available to help with syntax verification. "**`validns`**" is one.

- In addition to syntactic checking, **`validns`** is also able to check the zone file against a set of "policies".
- **`validns`** does several DNSSEC related checks, including validation of DNSSEC signatures

```
server% validns -p single-ns -s -z zonename filename
```

a "policy" (annotation pointing to `single-ns`)

- Among the ten defined policies are: "**`single-ns`**", "**`dnskey`**", "**`dname`**" and "**`all`**"
- See `http://www.validns.net/`

Zone Checker Tools: `named-checkzone`

- **`named-checkzone`** is a stand-alone zone checker tool bundled with BIND9 that performs the same zone tests and verifications that BIND9 do when loading a zone
  - this does not include validation of DNSSEC signatures (on the other hand, there is yet another ISC tool for that, called **`dnssec-verify`**...)

check this out

```
server% named-checkzone zonename filename
```

- The modifier "**`-D`**" cause **`named-checkzone`** to produce a version of the zone in canonical format:

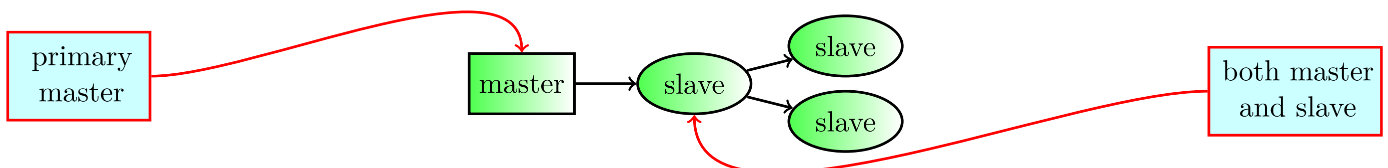
```
server% named-checkzone -D -o outfile zonename filename
```

- See "**`man named-checkzone`**" for more details

Authoritative Servers

Authoritative servers by definition know everything about the zone (or zones) that they are authoritative for. There are several types:

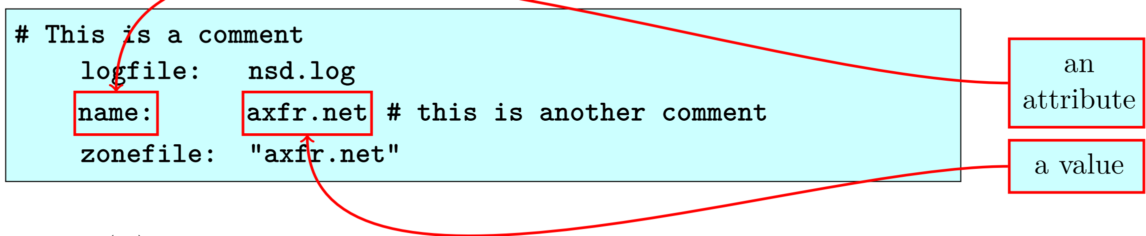
- A master server is a nameserver that allows outgoing zone transfers to at least some destinations (e.g., slave servers).
- A slave server receives the zone data from a master by requesting a zone transfer when its check of the **`SOA`** serial number indicates that there has been a change in the zone.
- A hidden master is only intended to supply the zone to the slave servers, i.e. it is not to be used by resolvers. There is no **`NS`** record for a hidden master.



6.2 NSD Configuration

NSD Configuration

- NSD has a text file as the primary configuration file. It is usually located in `/usr/pkg/etc/nsd/nsd.conf`
- The NSD configuration consists of **attributes** and **values**
  - the attribute keywords always have a trailing colon, "`:`"
  - the comment character is "`#`" and comments extend to the end of the current line

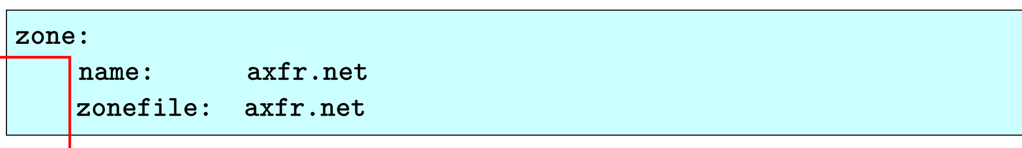


```
# This is a comment
    logfile:   nsd.log
    name:      axfr.net # this is another comment
    zonefile:  "axfr.net"
```


- Quotes (`""`) on values are optional

NSD Configuration, cont'd

- Compound attributes take a number of attribute/value pairs as their value:



```
zone:
    name:      axfr.net
    zonefile:  axfr.net
```

- At the top level only the attributes `server:`, `key:` and `zone:` are legal. NSD4 adds one more global attribute, `pattern:`
- There **must** be whitespace between keywords:

```
# Illegal syntax:
zone:name:  axfr.net...
```

NSD Configuration, cont'd

- All global parameters are grouped inside the "**server:**" attribute:

```
server:
    logfile:      /usr/pkg/etc/nsd/nsd.log
    verbosity:    1
    ip-address:   10.23.0.1
    ip-address:   2a00:801:f:1:1:4033
    zonesdir:     /usr/pkg/etc/nsd/zones
```

- I.e. the **server:** attribute is mostly equivalent to the BIND9 "**options**" block
- As expected, there are many more sub-attributes to the **server**: attribute. See the **nsd.conf** documentation

NSD: Zone Configuration

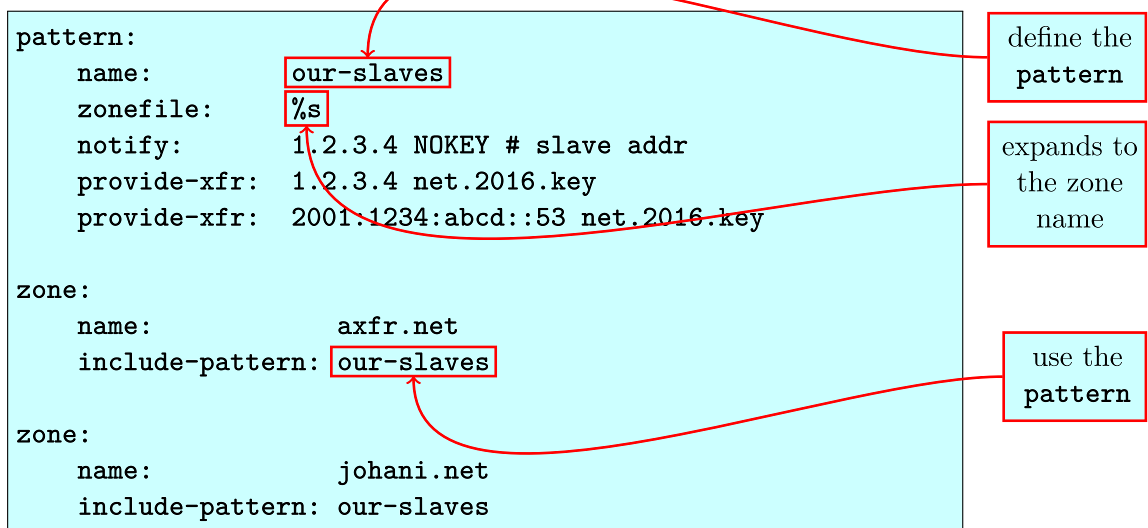
- In NSD configuration of a master zone is done with the **zone:** attribute :

```
zone:
    name:          axfr.net
    zonefile:      axfr.net
    notify:        1.2.3.4   NOKEY # slave addr
    provide-xfr:   0.0.0.0/0 NOKEY
    provide-xfr:   ::/0      NOKEY
```

- The"**notify**:" is needed, as NSD requires explicit notifies
- The zone file is located in **/usr/pkg/etc/nsd** unless changed with the "**zonesdir:**" attribute

NSD: Zone Configuration; `pattern:`

- NSD4 has a new attribute called **`pattern:`** which may be used to simplify configs with many zones:



NSD4 `pattern`, cont'd

`pattern:` is an aggregation of settings that will be reused for multiple (but not necessarily all) zones. The benefit of using **`pattern`** obviously comes when there are many zones

- It is possible to use one **`pattern:`** inside another **`pattern`**

```
pattern:
    name:             our-slaves
    notify:           1.2.3.4 NOKEY # slave addr
    provide-xfr:      1.2.3.4 net.2016.key
    provide-xfr:      2001:1234:abcd::53 net.2016.key
    include-pattern: some-other-pattern

zone:
    name:             axfr.net
    zonefile:         axfr.net
    include-pattern: our-slaves
```

- Used with some care, **pattern** is a great help in configs

6.2.1 nsd-control: The NSD4 Control Utility

nsd-control: NSD4 Controller

- **nsd-control** is the control utility used to manage a running NSD4 nameserver. It is of course modelled on the similar utility **unbound-control** for the Unbound server
- **nsd-control** is able to work remotely, and therefore it needs authentication
  - this is achieved via the **nsd-control-setup** utility, which must be executed **once** to set up certificates for authentication

```
# nsd-control-setup
```

  - in addition, the control interface must be activated in **/usr/pkg/etc/nsd/nsd.conf**:

```
remote-control:
    control-enable:  yes
```

nsd-control: The NSD4 Control Utility, cont'd

Some of the available commands are:

`start` / `stop`	start or stop server
`reload [zone]`	reload modified zonefiles from disk
`reconfig` / `repattern`	reload the config file
`log_reopen`	reopen logfile (for log rotate)
`status`	display status of server
`stats`	print statistics, reset counters
`stats_noreset`	peek at statistics
`addzone name pattern`	add a new zone
`delzone name`	remove a zone
`write [zone]`	write changed zonefiles to disk
`notify [zone]`	send NOTIFY messages to slaves
`transfer [zone]`	update slave zones to newest serial
`force_transfer [zone]`	update slave zones with AXFR, no serial check
`zonestatus [zone]`	print state, serial, activity
`serverpid`	get pid of server process
`verbosity number`	change logging detail

6.3 Query Logging

Query Logging in the Nameserver

Query logging is a bit of a contentious issue. Users want it, but implementors are concerned about the performance impact. I.e. the argument is that query logging is better managed outside of the name server (i.e. via packet capture in front of the server)

- Unbound has (somewhat controversially) introduced query logging via the `log-queries:` attribute
- NSD4 still does not offer query logging. We will see whether that holds or not

Traffic Logging with an External Tool (`tcpdump`)

An alternative to "expensive" query logging inside the nameserver is using an external tool like **tcpdump**.

```
# tcpdump -t -i vlan1 port 53
...
IP 10.1.0.5.6660 > 10.1.0.2.domain: 6016+ SOA? axfr.net. (26)
IP 10.1.0.2.domain > 10.1.0.5.6660: 6016 1/3/6 SOA (274)

IP 10.1.0.5.5621 > 10.1.0.2.domain: 308+ A? www.axfr.net. (30)
IP 10.1.0.2.domain > 10.1.0.5.5621: 308 1/3/6 A www.axfr.net (234)
```

recursion set

query id

answer is an A RR with this PTR reverse

number of RRs in Answer, Authority and Additional

msg size in octets

- There really is as much detail as with query logging inside the nameserver
- Plus advantages like seeing both query and response, implementation independence, host independence, etc

Traffic Logging à la dnstap

The drawback with traffic logging inside the nameserver is performance, but the performance hit mostly comes from the actual **writing** of the log. The drawback with external tools is that while the cost of logging doesn't directly affect the nameserver the external tool doesn't have access to the same information as the nameserver does (was this a cache hit or a cache miss? is the nameserver in-bailiwick or not, etc).

- **dnstap** is an emerging standard for how to achieve the best of both worlds. The idea is that the nameserver writes to a *socket* provided by a **dnstap listener**. Thereby the nameserver is separated from most of the costs associated with logging.
- There is support for **dnstap** in at least Unbound and Knot-DNS. BIND9 will get support from 9.11+ (to be released). Not yet in NSD4, but the **dnstap** design works sufficiently well that in spite of NSD4 being much performance focused it would probably be a good idea.

Traffic Logging with External Tools, cont'd

If traffic logging is of interest then we recommend looking into tools like:

- **dnscap**: a **tcpdump**-like capture utility that is specialized for DNS capture, deals with capture over TCP, various selection patterns, etc. End result is a PCAP-file just as with **tcpdump**.

- **`dnstop`**: a utility to either read directly from a network device or from a stored PCAP-file and display present data like most active resolver, various forms of most queried for names, records types, etc.
- **`packetq`**: a utility to query DNS data stored in a PCAP-file via a limited form of SQL syntax. Incredibly flexible and powerful. Available on GitHub.

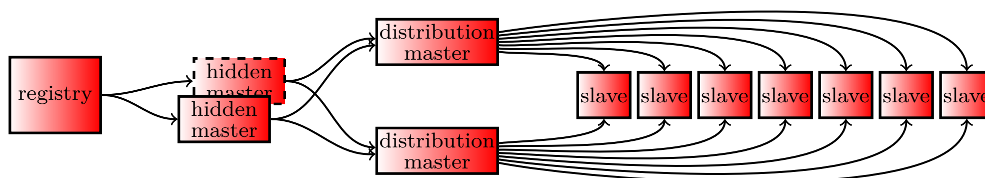
7 DNS Theory #3

7.1 Zone Transfer

7.1.1 AXFR

Zone Transfer: AXFR

- The operation where a slave server copies information from a master server is called zone transfer, AXFR
- A zone transfer can take place from any authoritative server
  - including another slave server
- It is possible and sometimes sensible to create multi-level, parallel, transfer hierarchies
  - think central infrastructure
- AXFR uses TCP, not UDP



NOTIFY

- The original synchronization model in DNS (slaves periodically fetching the `SOA` and looking at the serial number) leads to latencies in synchronization
  - a better model is for the master to inform the slave when a change occurs, which is what NOTIFY (RFC1996) is about
- When a slave gets a NOTIFY message, the time until the next refresh is set to zero, which leads to a `SOA` fetch and a serial number comparison as usual
  - with NOTIFY it is often possible to keep servers in sync within a couple of seconds (depending on the size of the zone)
- If there are stealth servers for a zone, the master server will not find them unless the master is statically configured to also send notifies to them

Zone Transfer: IXFR

- Full zone transfer (AXFR) when the zone is updated is often inefficient because usually only a small fraction of the records have changed
- IXFR (Incremental Zone Transfer, RFC1995) is an alternate type of zone transfer that only transfers the changes made since the slave was last updated
  - the effect is basically "send me what's new"
  - because of this the master needs to keep state tracking the deltas between versions of the zone
  - it is up to the master whether IXFR is supported or not and therefore the slave must be prepared to receive a full AXFR in response to an IXFR request

7.2 Slave Server Configuration

NSD Slave Zone Configuration

- Configuration of a slave zone in NSD also (uses the "`zone:`" attribute):

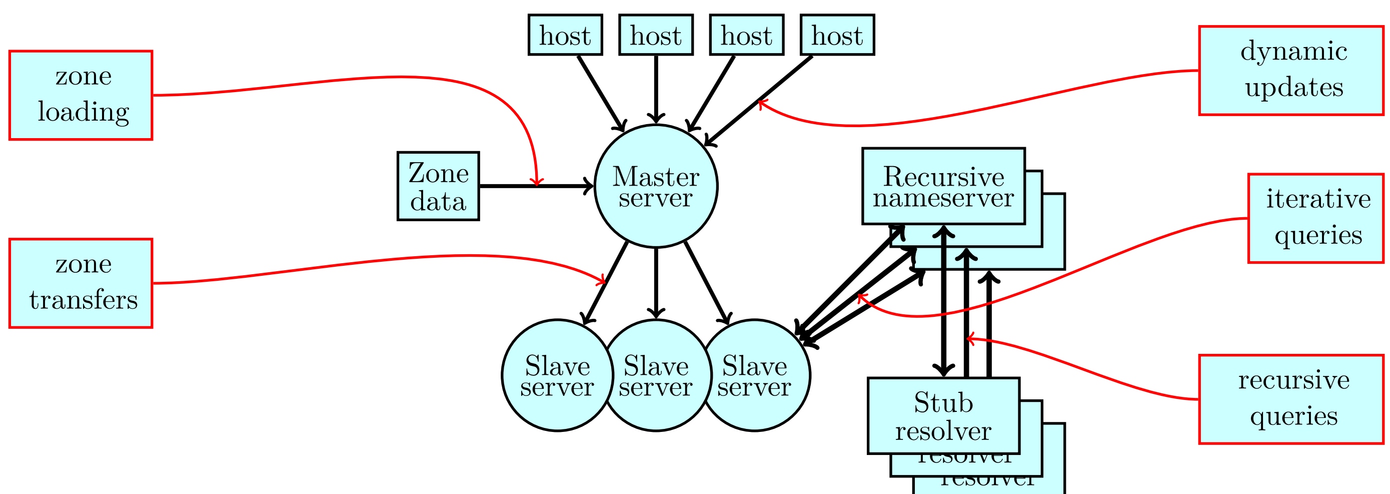
```
zone:
    name:           axfr.net
    zonefile:       axfr.net
    allow-notify:   192.71.80.12 NOKEY # master
    request-xfr:    AXFR 192.71.80.12 NOKEY
```

- The "`NOKEY`" keyword is needed to tell NSD that messages will **not** be TSIG-signed
  - otherwise NSD will require a TSIG signature
- The "`AXFR`" keyword is needed **only** if the master doesn't support IXFR
  - like NSD, which only supports **incoming** IXFR zone transfers
- If using NSD4 it is of course possible to use "`pattern`"

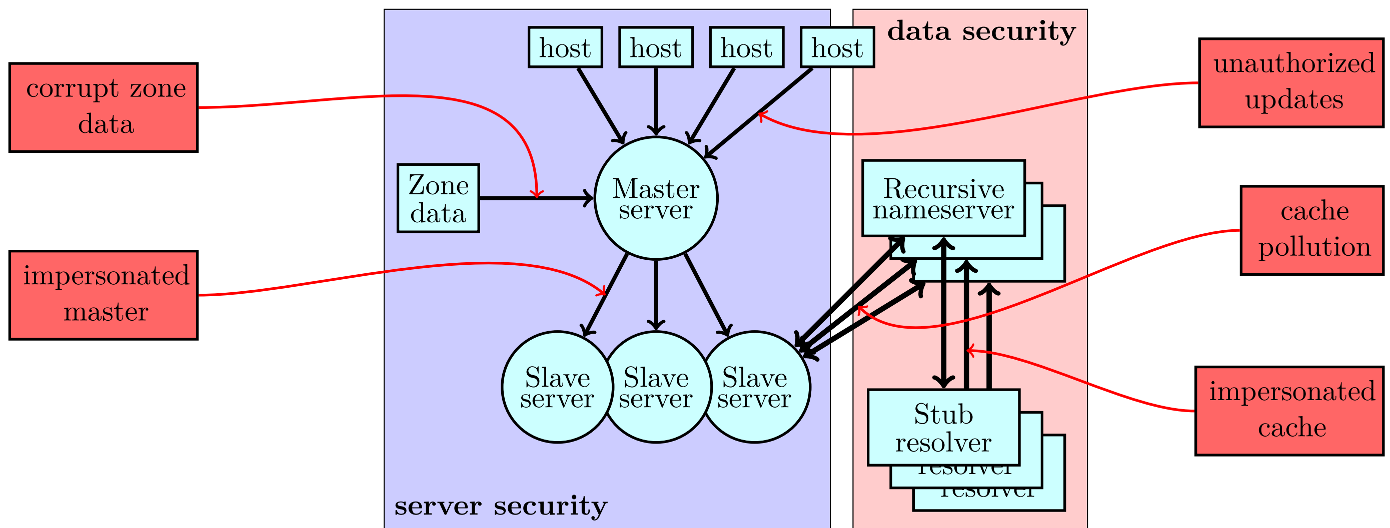
8 DNS Security

8.1 DNS Vulnerabilities Overview

DNS TransactionsDNS Vulnerabilities



DNS Vulnerabilities



9 EDNS(0): Extended DNS

9.1 Background

EDNS(0): Extended DNS

The DNS protocol was designed to allow for "future growth" by allowing the addition of new record types (and also new classes). However, several other protocol constraints have caused problems over the years.

- 512 octets per UDP packet is tight
  - forces non-trivial queries to fall back on using TCP transport
  - prior to the `DNSSEC OK` bit the size constraints were a major issue for DNSSEC deployment
- 16 possible RCODEs were almost used up

- 4 possible label types were entirely used up
- There was and is a need for new options and the possibility of negotiating supporting functionality

EDNS(0): Extended DNS, cont'd

EDNS is a framework for DNS protocol extensions, defined in RFC 2671 (and updated in RFC 6891)

- EDNS(0) extends several different fields in the DNS packet, both use and size wise
- An "`OPT` record" (located at the end of the Additional Section) announces the resolver "reassembly buffer size", EDNS protocol version, and a list of options of variable length
- If an RCODE of "`FORMERR`" is returned then the server is not EDNS-aware
  - this information is cached before trying again without EDNS(0)
- Each side announces an EDNS version number, then only $min(us, them)$ functionality is used
- EDNS(0) is EDNS version 0 (as in "zero")
  - there have been proposals for both an EDNS(1), which was rejected, and EDNS(0.5) which failed to gain traction

EDNS(0): Extended DNS, examples

- The addition of the so-called "`DNSSEC OK`" flag bit
  - this is used by a resolver to indicate its interest in receiving DNSSEC security information together with the answer
- The `NSID` option which may be used to identify a responding name server via a binary blob of octets in the response

```
ardbeg# dig @a.ns.se se soa +dnssec +nsid
...
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do ; udp: 4096
; NSID: 73 33 2e 74 61 69 (s) (3) (.) (t) (a) (i)
```

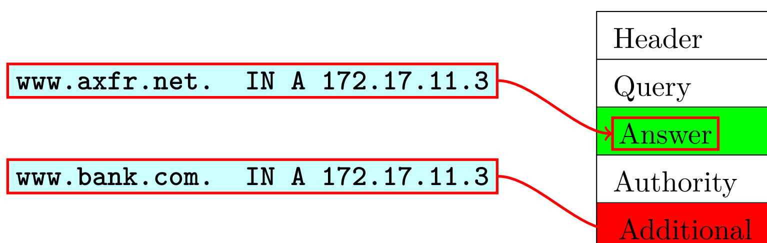
- The signalling of "buffer reassembly size" (from resolver to authoritative server) enables the larger responses needed for DNSSEC and IPv6.

10 DNSSEC

10.1 "Cache Pollution"

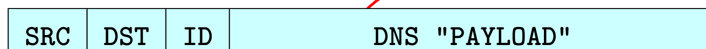
"Cache Pollution"

- It is possible for the server to send irrelevant information in the Additional Section
  - by definition it **should** contain answers to questions that have not (yet) been asked



"Cache Pollution"

- Usually DNS uses UDP for both queries and responses
  - a DNS UDP packet has a very simplistic internal structure



  - "the sender" is able to lie about anything inside the packet: about the answer, about its own identity, about the "query id", etc

  - "the recipient" (i.e. the resolver) has very limited possibilities to verify a received UDP packet and no way to verify the actual contents of the packet

- "Cache pollution" is the commonly used term for what happens when a recursive, caching name server "successfully" is fooled by forged data

  - so that it then in the next step can fool its own clients (i.e. all the stub resolvers)...

DNS Security, cont'd

- Threat model

  - there are bad guys in the water
  - what can we do from a DNS perspective to defend against this?
  - what should we not do?

- DNSSEC design goals

  - interoperability with standard DNS, standard data should be augmented with security information
  - allow gradual phase-in, no flag days possible

- DNSSEC design

  - what was the result?

- DNSSEC operations

  - very different from zone owner POV and validating resolver POV
  - key managment, re-signing data, tracking trusted-keys, ...

DNS Threat Model

- It is clear that there are DNS vulnerabilities

- It is clear that forged DNS responses are hard to detect

  - hence "spoofing" is a real problem

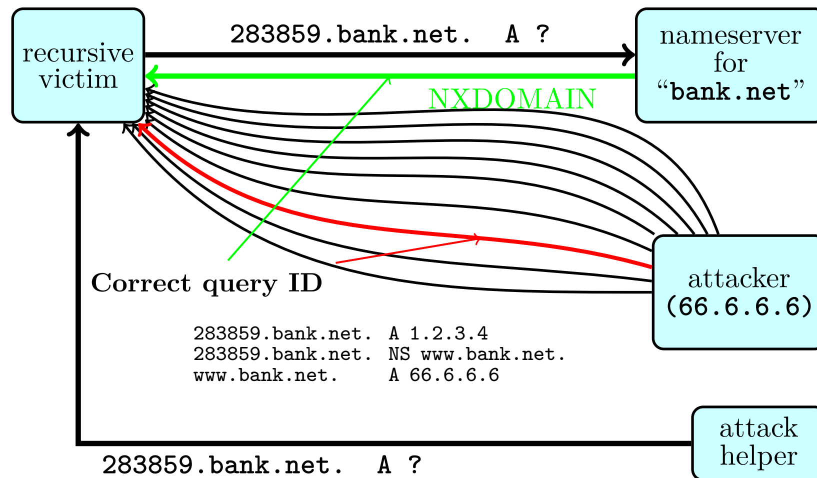
- It is also clear that we cannot easily replace DNS
  - instead we need to "augment it in place", i.e. improve the system in situ
- With such in situ improvement some limitations follow
  - the added security needs to be actively queried for to help
  - most stub resolvers are entirely dependent on the goodwill of their recursive server
- Also DNS security can never shield against an attack, it will only provide mechanisms to detect it

10.1.1 The "Kaminsky" Attack

The "Kaminsky Attack"

- In August 2008 the so-called "Kaminsky attack" made headline news across the Internet
- The basic idea is to increase the number of queries that can be attacked significantly thereby improving the odds
  - Kaminsky's idea **drastically increased** the efficiency compared to "standard DNS spoofing"
  - there is proof-of-concept code available that "spoofs" a recursive name server within a few seconds
- The "trick" is to tickle the recursive server to always query for new things
  - thereby defeating the hold-down effect of caching
  - the actual "spoofed" records are not the Answer to the random query, but instead the **Authority** and **Additional** data (clever)

The “Kaminsky Attack”, cont’d



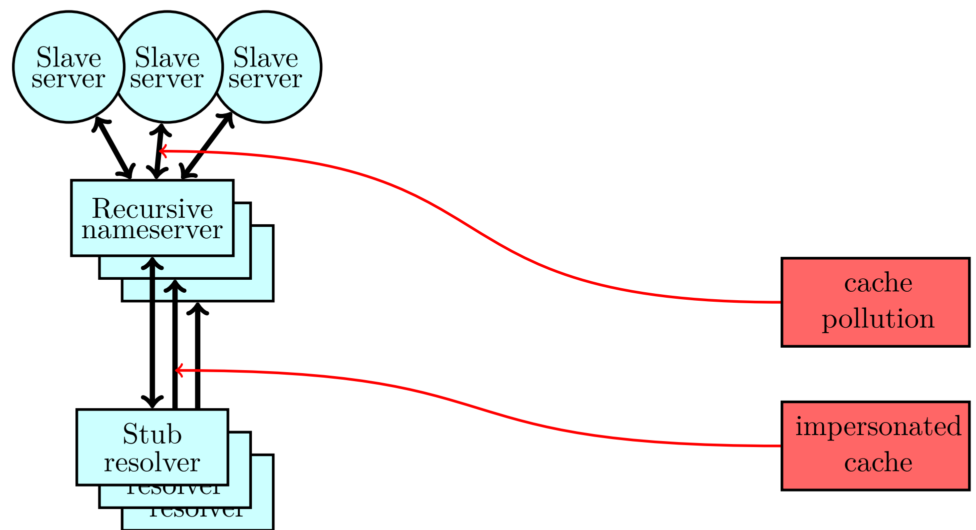
The “Kaminsky Attack”, cont’d

- The problem with the Kaminsky attack is that it really attacks a weakness in the DNS protocol, not an implementation bug
  - short term the fix is to increase the available entropy to make guessing harder (i.e. make the odds worse for the attacker)
- The major fix so far is to use a **randomized source port**
  - gains another 14 bits of entropy (i.e. changes the attack from “seconds” to “hours”)
  - potentially has various issues with firewall and NAT configurations where it is common to lock down the source port
- Various other suggestions all have problems
- Long term DNSSEC seems to be the only viable option

Introduction to DNSSEC

- The main problem is that there are millions of recursive, caching name servers on the Internet

- and they all need to be able to look up data from millions of individual zones
- there simply is no way to administer (or secure) a system based upon exchange of secret keys for millions of participants
- hence “server protection” is bound to fail as a protection mechanism against spoofed data



Introduction to DNSSEC, cont’d

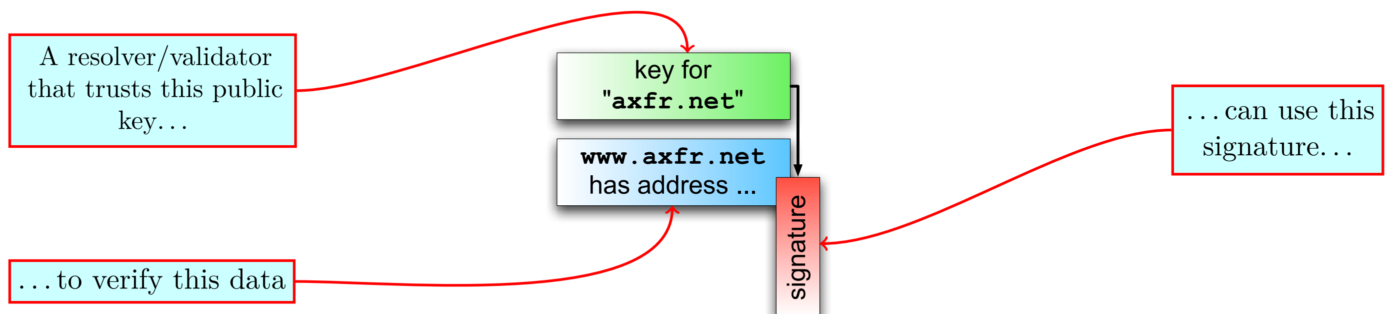
- The solution must therefore be based on public key technology
- What to do with the two keys:
  - the private, secret, key is used to “sign” traditional DNS data
  - the public key is published via DNS so that validators can retrieve it
  - the public key is used to verify the authenticity of the generated signatures (and, thereby, the DNS data that the signature covers)

10.2 Data Protection

10.2.1 Signature Validation

Protection of DNS Data

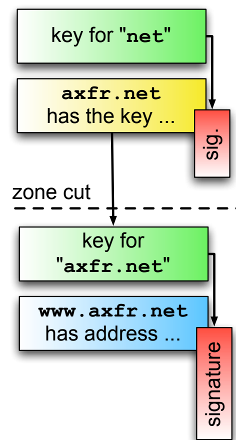
- The central concept in DNSSEC is the idea that standard DNS data is augmented by a "signature"
- A "validator" (a resolver that verifies signatures) can verify that the data is authentic (i.e. is supplied by the correct owner and has not been modified)
  - for the verification the public key is needed
- **Please note: there is a problem here in that every "validator" somehow needs to get the key for axfr.net**
  - it is not quite as easy as just looking it up in DNS
  - because stuff looked up in DNS may be spoofed...



Protection of DNS Data, cont'd

- To achieve better scaling properties it is possible to do this in several levels
  - modulo some extra complexity in the zone cuts
- Now it is no longer necessary for every validator to have the key for "`axfr.net`", because it is sufficient to have the key for "`net`" or even "."
  - with the help of the key for "`net`" it is possible to verify the authenticity of the key for "`axfr.net`" and therefore it is possible (i.e. safe) to look up the latter in the DNS

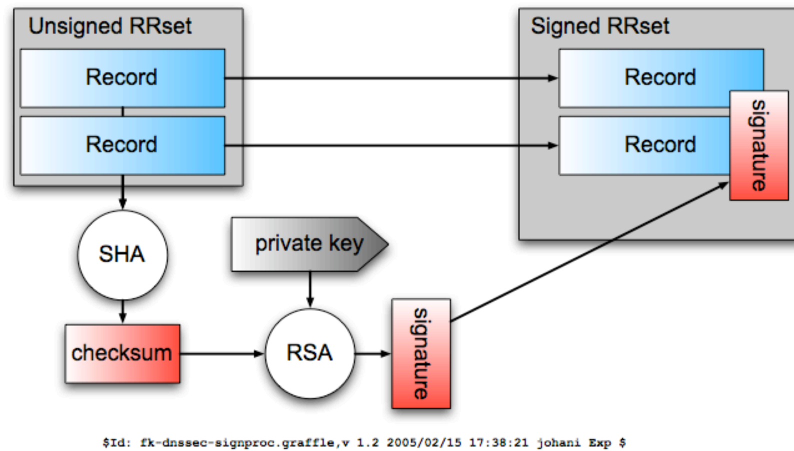
– to avoid being spoofed the "top most" key (the one used to verify everything else) must be acquired by some "outside" method (i.e. not just look it up in DNS)



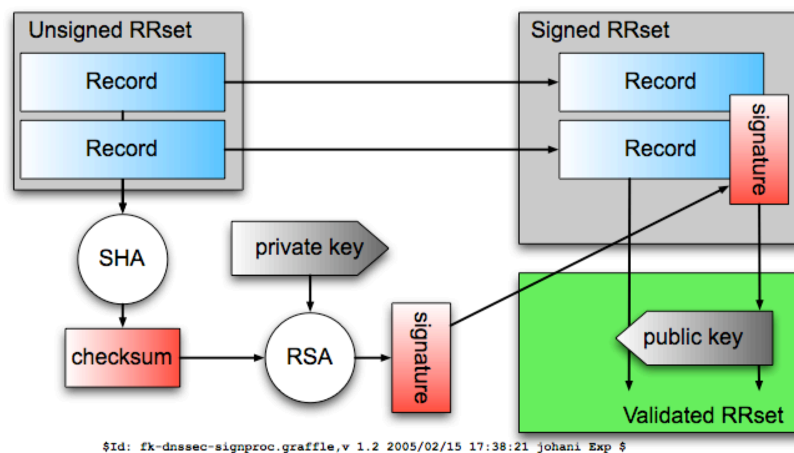
Main DNSSEC Services

- Distribution of public keys
- Digital signatures of data (i.e. DNS records)
  - the signature is intended to prove that the DNS data has not been modified while in transit from the zone owner
  - therefore asymmetric encryption is used, with one private key (used to generate the signature) and one public key (used to verify the signature)
- DNSSEC has consequences in the form of changed semantics for TTL, CNAMEs and delegations (among other things).

Signing Data



Signing . . . and Validating Data



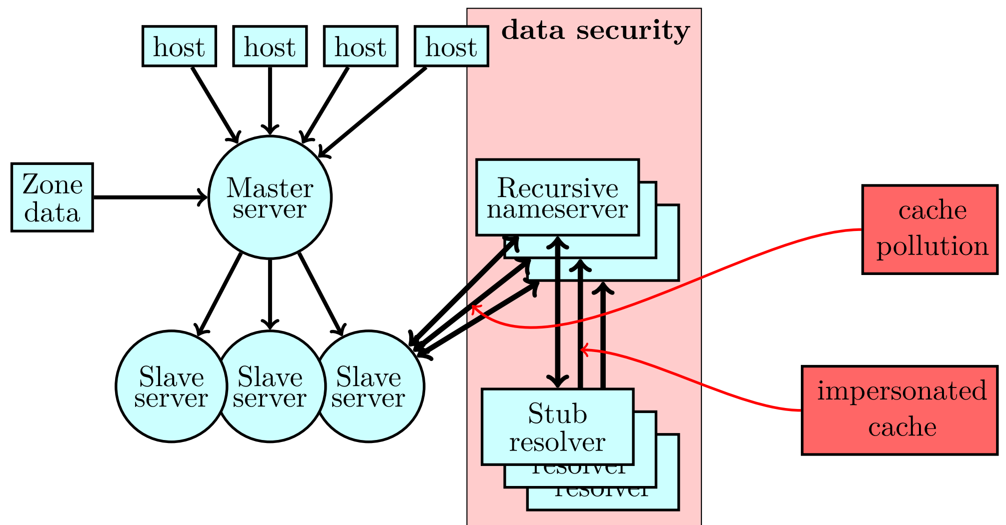
DNSSEC is not intended for

- Confidentiality
  - for such purposes some sort of TLS (Transport Layer Security) should be used, e.g. SSL
  - but, as always, first question **why**. . .

- also see later slides about "zone enumeration"

- Access control
  - authentication of servers and clients for zone transfer and query management
  - use other mechanisms (like TSIG) for that

Vulnerabilities that DNSSEC will protect against



10.2.2 Trust Anchors

Verifying DNS Data Via Signatures

- During validation of a chain of signatures the validating resolver must be able to verify all signatures against keys it implicitly trusts
  - such keys are called `"trust anchors"` or "`trusted-keys`"

- Therefore for a validator there are primarily three possible types of DNS data:
  - signed data covered by a trusted key
  - signed data not covered by a trusted key
  - unsigned data
- The management issue is not the DNS data at all but instead the trusted keys and how to acquire them
  - with a single island of trust the problem is manageable, but with many islands it explodes
  - with many islands automated tools will be necessary (this is often referred to as “automatic rollover” systems)

What Keys Should a Validator Trust?

- The problem with the trusted keys is that they need to be acquired out-of-band
  - because the point is to use them to verify data in DNS they cannot themselves easily be fetched from DNS because then the trusted key itself could be tainted
- It is possible for a validator to have any number of trusted keys, typically one per signed zone it needs to validate data from
  - but in practice configuring trusted keys for every zone is not feasible. Therefore there is support in the DNSSEC protocol that drastically cuts down on the number of trusted keys that are really needed. This will be described later

Configuring Trust Anchors in Unbound

- Unbound trust anchors may be configured in several ways
- The “**`trust-anchor`**:” attribute accepts keys in the shape of either “`DNSKEY` records” or “`DS` records”

  - these records have not yet been described, but in essence a `DNSKEY` is what BIND9 uses, while `DS` is a secure hash of a key

- The "`trust-anchor-file:`" attribute refers to a bunch of `trust-anchor:` statements in an external file
- The "`auto-trust-anchor-file:`" attribute provides automatic updating according to RFC5011 of the trust anchors in the file.

```
# unbound.conf
server:
    trust-anchor:     "axfr.net. DNSKEY 257 3 8 AQPzzTWMz..."
    trust-anchor-file:       my-trusted.keys
    auto-trust-anchor-file:  root.key
```

The Trust Anchor for Root

The most important (in many cases the **only**) trust anchor that is needed is the root trust anchor

- How does one get that in a secure way? I.e. with out-of-band authentication
- A convenient method is to use the `unbound-anchor` utility that comes with Unbound:

```
#unbound-anchor -v
/usr/pkg/etc/unbound/root.key does not exist
success:  the anchor is ok
#cat /usr/pkg/etc/unbound/root.key
; autotrust trust anchor file
;;id:  .  1
;;last_queried:  1295346020 ;;Tue Jan 18 10:20:20 2016
;;last_success:  1295346020 ;;Tue Jan 18 10:20:20 2016
;;next_probe_time:  1295387757 ;;Tue Jan 18 21:55:57 2016
...
.  172800 IN DNSKEY 257 3 8 AwEAAagAIK ...
7knNnulqQxA+Uk1ihz0=
;id = 19036 (ksk), size = 2048b ;;state=2 [ VALID ] ;;count=0
```

The Trust Anchor for Root, cont'd

- **`unbound-anchor`** fetches the root anchor (the trusted key for root) via **`https`** and then verifies the key against a compiled in X.509 certificate that belongs to ICANN
- It is possible to get Unbound to automatically maintain this trust anchor via a method described in RFC5011; more on that later

> Note that it is not possible to use **`unbound-anchor`** to configure the trust anchor in the lab environment because of the compiled in certificate!
> Here you need to do this manually!

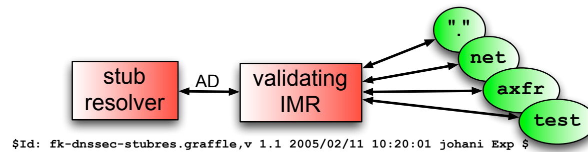
Security Policies

- Note that the "**trust anchor**" configuration is not only a question of defining keys
  - the configuration also defines a security policy for this validating resolver
- If there is a "**trust anchor**" configured then success in validated lookups in the sub tree that the trust anchor covers will be defined by the signature chains verifying against this key or not
  - it can be dangerous to "forget" old trust anchors, since they may cause lookups to fail once the zone owner no longer signs the data with that (old) key
  - the validating resolver needs to track key rollovers and ensure that the keys it decides to trust are current

Stub Resolver View

- Active validation of signatures is likely to mostly be done by IMRs, iterative mode resolvers, i.e. recursive name servers
- Usually IMRs serve a population of "stub resolvers" . The stub resolvers are typically not able to perform their own validation of DNSSEC signatures

- A stub resolver is capable of looking for a sign from the recursive server that indicates whether the data verified or not.
  - this “sign” is implemented via a flag bit called “**AD**”



The AD Bit

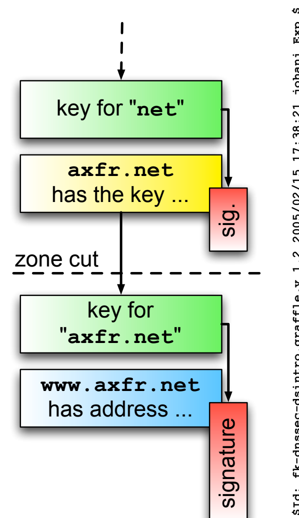
- A status bit in the Header section of the DNS packet
  - prior to DNSSEC this bit was not used (should be zero)
  - only used in responses from a validating name server
  - the **AD** bit is **not** set by an authoritative name server
- **AD**: Authenticated Data
  - 1 = the validating name server has successfully verified the Answer
  - 0 = the Answer has not been verified by the name server
- Note that “trusting” the **AD** bit is a viable strategy only for a stub resolver that has to trust its upstream IMR (recursive name server) regardless

The CD Bit

- A status bit in the Header section of the DNS packet
  - prior to DNSSEC this bit was not used (should be zero)
  - only used in queries
- CD: Checking Disabled
  - 1 = checking disabled. The client accepts non-verified responses
  - 0 = checking enabled. The client wants verified responses for signed data but accepts non-verified responses for unsigned data.
    - i.e. “please check the signatures for me when they exist”

"Signature Chain"

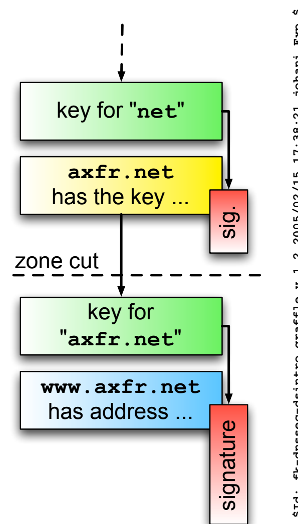
- For a validating resolver to be able to trust an answer it needs to be verified
- This is the process where the validator authenticates data by examining associated signatures and keys and DS records from the data all the way to a "known key" (i.e. a trusted key)
  - if an answer is signed by an unknown key then that key needs to be looked up (and verified)
  - at a zone cut the parent holds a signed "secure fingerprint" of the child key rather than a signature of the key itself
  - verification proceeds until we reach a known key that is implicitly trusted



"Signature Chain", cont'd

- Theoretically only a single trust anchor will be needed (the key for ".")
  - i.e. all zones will be securely tied together with secure fingerprints of each child key at every parent

- But in practice it is expected that there will be multiple "islands of security" that initially share a signed ancestor
  - then a trusted key will be needed for the apex (i.e. the top) of each "island"



10.3 Creating a Signed Zone

Creating a Signed Zone

- The zone management process changes considerably in a DNSSEC infrastructure
- Keys need to be periodically generated and replaced in the zone
  - there are tools that do this but they need to be used
  - in some cases the public key for the zone needs to be distributed for use as a trusted-key by validators (more on that later)
- "Signatures" need to be periodically re-generated (there are tools that does this too but they need to be used)
- Keys and signatures are stored in new record types (called "`DNSKEY`" and "`RRSIG`" respectively)

Different Roles for Keys

- Because DNSSEC signatures cover entire RRsets it is not possible to replace individual keys without regenerating the signature
  - this could have caused a problem at a zone cut, if the parent generated the signature over the child's `DNSKEY` RRset
  - because then the parent would be involved every time a key was changed
  - therefore instead of the parent signing the child key RRset the parent signs a **fingerprint** (a so-called "hash") of only one of the child keys, the so-called Key Signing Key, or KSK
  - the KSK is then used to sign the entire RRset of all child keys
    * which in addition to the KSK includes so-called Zone Signing Keys, ZSK
- KSKs and ZSKs will be covered in more detail later when "key rollovers" (the mechanism for changing keys) are discussed

Manual Signing of a DNS Zone

To "sign" a DNS zone several steps are needed:

- for each zone, **keys need to be generated** with a key generator tool
- the **current keys are used to sign** the unsigned zone, creating a signed zone (i.e. a version of the unsigned zone where DNSSEC information has been added)
- the nameserver **configuration has to be changed** to load the zone from the file containing the signed zone rather than the unsigned zone

In production use most of these steps will be more or less automated, but let's start with how this works manually.

- Later on we will look at how to make this more convenient and finally in some cases fully automatic

10.3.1 Generating DNSSEC Keys

Generating Keys with "dnssec-keygen"

- The BIND9 distribution includes "**dnssec-keygen**", which may be used to generate asymmetric keys. This is the same tool that was used to generate symmetric keys for TSIG.
- Usage (entire command on one line):

```
dnssec-keygen -a algorithm -b bits -n ZONE [options] name
```

- Arguments:
- `-a algorithm`: Subset of asymmetric algorithms: `RSA/MD5`, `RSA/SHA1`, `RSA/SHA256`, `DSA`, `ECCGOST`, `ECDSA/P256SHA256`
- `-b length`: size of key, in bits. Both `RSA/SHA1` and `RSA/SHA256` use a key length in the interval [512..4096]
- `-f KSK`: flag to indicate that this key is intended as a Key Signing Key and not a ZSK (more on that later)

Generating Keys with "dnssec-keygen", cont'd

- `-e` Use large exponent for `RSA/SHA1`. Recommended.
- `-n nametype`: `ZONE | HOST`. Always "`ZONE`" for DNSSEC usage (the "`HOST`" alternative is used for symmetric TSIG keys)
- `name`: owner name of the key (= name of zone)

  Other arguments:
- `-r randomdev` (a device or file containing random data)

- `-v verbosity`

  Result, two new files:

  `K<name>+<algorithm>+<keyid>.key`

  `K<name>+<algorithm>+<keyid>.private`

Generating Keys with "ldns-keygen"

- The **ldns** distribution includes "**ldns-keygen**", which may be used to generate asymmetric keys. This is the same tool that was used to generate symmetric keys for TSIG.
- Usage:

```
ldns-keygen -a algorithm -b bits [options] name
```

- Arguments:
- `-a algorithm`: Subset of asymmetric algorithms: `RSA/MD5`, `RSA/SHA1`, `RSA/SHA256`, `DSA`, `ECCGOST`, `ECDSA/P256SHA256`
- `-b length`: size of key, in bits. Both `RSA/SHA1` and `RSA/SHA256` use a key length in the interval [512..4096]
- `-k`: flag to indicate that this key is intended as a Key Signing Key and not a ZSK (more on that later)

Generating Keys with "ldns-keygen, cont'd"

- `name`: owner name of the key (= name of zone)

  Other arguments:

- `-r randomdev` (a device or file containing random data)
- `-v verbosity`

Note: While **ldns-keygen** reuses the same file format as **dnssec-keygen**, it is important to be aware that the latter has evolved to add more "timing" metadata to the format. This affects the DNSSEC zone signing tools as will be shown shortly.

Generating an Asymmetric Key Pair

Example:

```
server% dnssec-keygen -a RSASHA256 -b 1024 -n ZONE axfr.net.
```

```
server% ldns-keygen -a RSASHA256 -b 1024 axfr.net.
```

- In production use a real source of entropy should be used
  - so that the generated keys are stronger
- The result is two new files:
  - `Kaxfr.net.+008+23456.key`
  - `Kaxfr.net.+008+23456.private`
- If using **`ldns-keygen`** a third file is also created:
  - `Kaxfr.net.+008+23456.ds`

  The **`.ds`** file contains a secure hash of the public key and will be explained in detail when discussing so-called "key rollovers"

Choice of Key Algorithm

- RSA: Slow signing, quick validation
  - "Recommended technology" in DNSSEC
  - There are several "secure hash" algorithms to choose between for RSA keys
    - MD5, i.e. algorithm "RSA/MD5" [obsolete]
    - SHA1, i.e. algorithm "RSA/SHA1" [ok, but next one is better]
    - SHA256, i.e. algorithm "RSA/SHA256", introduced in BIND 9.7.0 [recommended choice and used for the root zone]
- DSA: Validation slower than signing (both slower than RSA)
  - "Required technology" in DNSSEC (but not much used)

- GOST: a “family” of crypto algorithms used in Russia
  - apparently russian organizations may have to use a GOST algorithm for legal reasons
  - The algorithm “`ECCGOST`” is supported in modern versions of Unbound
- ECDSA (Elliptic Curve): Much smaller signatures and faster signing (but slower validation)

10.3.2 Signing a Zone with `dnssec-signzone`

Signing a Zone with “`dnssec-signzone`”

- Another tool included in the BIND9 distribution is “`dnssec-signzone`”, which is used to sign the data in a zone with one or more keys.
- `dnssec-signzone` is a powerful tool that performs several operations:
  - merge sort zone data from multiple sources
  - remove duplicates, sort in lexicographic order, etc
  - sign the RRsets in the zone
  - several other tricks that we will describe later [generate `DS`, `NSEC` and `NSEC3` records]
- `dnssec-signzone` is (unfortunately) fairly complicated
  - but there are reasonable default values for most arguments

`dnssec-signzone`, cont’d

```
server% dnssec-signzone [options] zonefile [zsks]
```

- Important parameters:

Parameter	Description
`-s YYYYMMDDHHMMSS` | `-s +offset`	signature start time (absolute or offset from now) [default `now`]
`-e YYYYMMDDHHMMSS` | `-e +offset`	signature end time (absolute or offset from `start`) [default start +30 days]
`-v debuglevel`	verbosity [default 0]
`-r randomdev`	a device (or file) providing random data [system random device]

10.3.3 Signing a Zone with `ldns-signzone`

Signing a Zone with "`ldns-signzone`"

- "`ldns-signzone`" is a simple but functional zone signer tool, quite similar to "`dnssec-signzone`", on which is is modelled
- Just like `dnssec-signzone`, `ldns-signzone` performs several operations:
  - merge sort zone data from multiple sources
  - remove duplicates, sort in lexicographic order, etc
  - sign the RRsets in the zone
  - several other tricks that we will describe later [generate `DS`, `NSEC` and `NSEC3` records]

`ldns-signzone`, cont'd

```
server% ldns-signzone [options] zonefile [keys]
```

- Important parameters:

Parameter	Description
`-i YYYYMMDDHHMMSS`	signature inception time (absolute) [default `now`]
`-e YYYYMMDDHHMMSS`	signature end time (absolute) [default start +30 days]
`-A`	sign with all keys instead of minimal set
`-b`	include helpful comments in output file to help debugging

`dnssec-signzone` "Smart Signing"

- `dnssec-signzone` is able to automatically find the keys to include in the zone file and use to sign with
  - In BIND9 terminology, this is called "smart signing"
  - It uses timing meta data stored with the keys to know which keys to use if there are several (more on this later)
  - Invoke smart signing by using the flag, `"-S"`:

```
# dnssec-keygen -a RSASHA256 ...  axfr.net
# dnssec-keygen -a RSASHA256 ...  -f KSK axfr.net
# dnssec-signzone -S axfr.net
Fetching KSK 32005/RSASHA256 from key repository.
Fetching ZSK 43509/RSASHA256 from key repository.
...
axfr.net.signed
```

Important note: Because "smart signing" relies on timing meta data stored in the key file it **does not work** with keys generated by other tools. I.e. the keys **must** be created with `dnssec-keygen`.

`dnssec-signzone`, cont'd

- In production use a real entropy source should be used
  - otherwise the generated signatures will not be nearly as strong as the key length indicates
- Note that it is some times a good idea to re-sign an already signed zone file, e.g.

```
server% dnssec-signzone axfr.net; mv axfr.net.signed axfr.net
```

  - this is the reason for the `-i`, `-j` and `-N` flags that define "resigning interval", "jitter" and "SOA serial format" respectively and thereby simplify such repeated signing

- Result (from the command on the previous slide):

`axfr.net.signed`	the signed zone file
`dsset-axfr.net.`	a file containing a secure hash of the KSK (will be discussed later)

10.3.4 The `DNSKEY` and `RRSIG` Records

Public Keys: The `DNSKEY` Record

- DNSSEC public keys are distributed via the new record type **`DNSKEY`**. It consists of:
  - various flags
  - an identifier for which crypto algorithm is being used
  - previously the **`DNSKEY`** record (or rather its predecessor the **`KEY`** record) was open for any type of keys, including PGP, but that has been changed to make the design less complex
  - the actual keying material
- **`DNSKEY`** records are sent in the Additional Section

The `DNSKEY` Record

```
# dig axfr.net dnskey +multi +dnssec
; <<>> DiG 9.7.0 <<>> axfr.net dnskey +multi +dnssec
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44785
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 2

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

;; ANSWER SECTION:
axfr.net.    7200 IN DNSKEY 256 3 8 (
                AQOu/+gpUc33OnyFZL06vxj74rraso2RTXhp10aTmkHA
                41hwlhhVO2DI8g3EiFe1iOdRyYSYrxpkrC0q52wZKuJt
                24RQAyGzxf9szqdFwjfKUnJAlamys4sHOTSz/x+bpvGU
                wO1SqNMOOMGgIn7c4oLVw== ) ; key id = 22139
axfr.net.    7200 IN DNSKEY 257 3 8 (
                AQOm4CS4/kzTGmQxQH6gsRf/KPd3HX9RmEVzveRzCZVX
                ALq9GbPfyor5rI414X8Kx1zmj0QaPgJJrkoEaDAPP5Pf
                6W+djWF0feyOaHkwq9RhvD+ywolqgGmOa6YMsg+OK/FW
                022qm9LCxUw/twqV1PLbQ== ) ; key id = 36393
```

Use `+multi` or `+rrcomments` to print key id

key bits

Flag bits:

bit 7	always 1
bit 15	1 if KSK
rest must be 0	

protocol: (always "3")

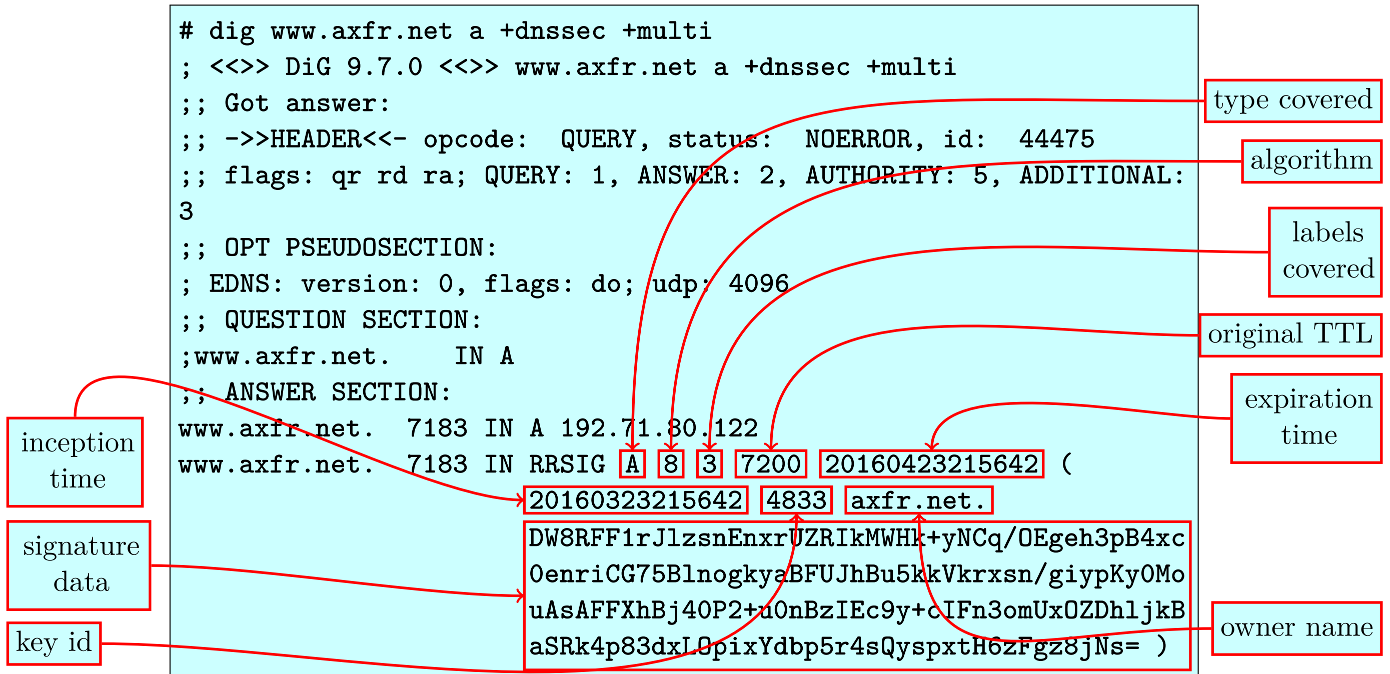
Algorithm:

5	`RSA/SHA1`
8	`RSA/SHA256`
...	...

Signatures: The `RRSIG` Record

- **Signatures** that authenticate DNS RRsets are transported as another new record type: `RRSIG`
- `RRSIG` is a sub typed record and exists per RRset
  - i.e. there will be one `RRSIG` for each record type that exists at a particular name, `A`, `MX`, `SOA`, etc
- There will be multiple `RRSIG` records for the same RRset if the RRset is signed with multiple keys
- `RRSIG`s are only generated for authoritative data
  - i.e. at a zone cut there will be `RRSIG NS` in the child zone, but not in the parent zone

The RRSIG Record



10.4 Key Management and Key Rollovers

Key Management

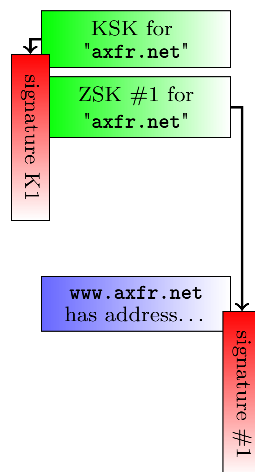
- It is not sufficient to only generate the key (or keys) once
  - every system dependent upon cryptography with secret keys needs a mechanism for dealing with key rollovers
- There are many reasons for key rollovers:
  - accidental compromise of a key

– a security policy that requires change when a trusted employee changes employment
– routine change after a certain amount of exposure (to avoid providing attackers with an unbounded amount of crypto material and time)

- However, there is also at least one reason **not** to do a key rollover
  – it adds complexity and there is a risk of mistakes

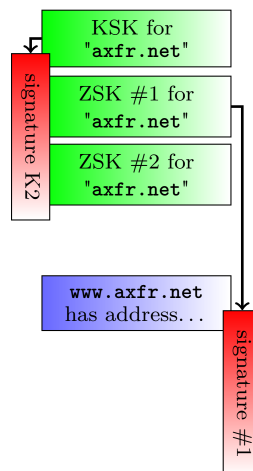
Key Rollover

- To make a key rollover simpler DNSSEC keys are divided into two categories:
  – Zone Signing Keys and Key Signing Keys
- During a key rollover event the zone data must be signed by both the old key and the new key
  – because there may be validating resolvers that have cached either old signatures or old keys
- Key rollovers can (in theory) be either manual or automatic



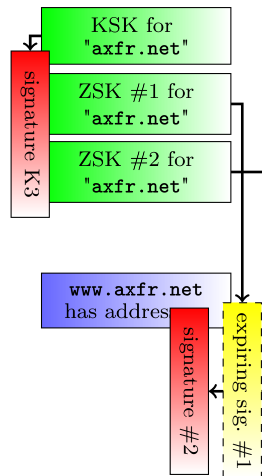
Key Rollover, cont'd

- ZSK, zone signing keys, are keys used to sign the zone data KSK, key signing keys, are keys used only to sign the DNSKEY RRset at the zone apex
  - i.e. the KSKs sign the RRset containing the union of all ZSKs and KSKs
- By distinguishing the ZSKs from the KSK it becomes possible to replace individual ZSKs without affecting the parent.
  - it is sufficient to re-sign the DNSKEY RRset with the (unchanged) KSK



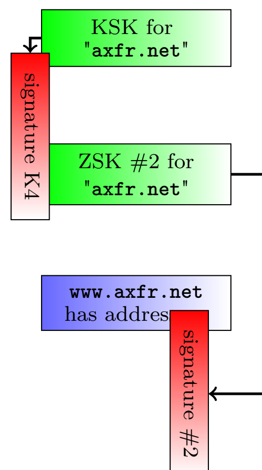
Key Rollover, cont'd

- When replacing the ZSK the first step is to add a new ZSK.
  - this will change the DNSKEY RRset, which in turn requires a new signature over this RRset
  - furthermore, the new ZSK generates a new signature over the zone data
- Note that only one signature at a time is needed
  - but for some time the old signature is present in slave servers and caches



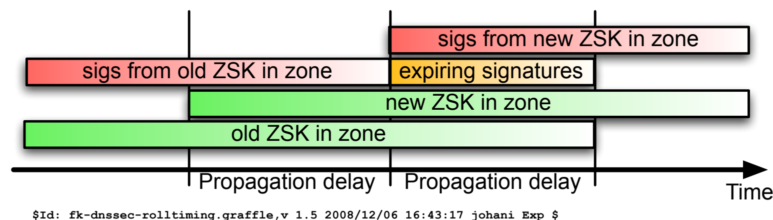
Key Rollover, cont'd

- When all old signatures are finally gone the old ZSK can be removed
  - this triggers yet another update of the signature over the DNSKEY RRset
- This is the key rollover operation for Zone Signing Keys
  - there is also need for rollover of the KSK, Key Signing Keys



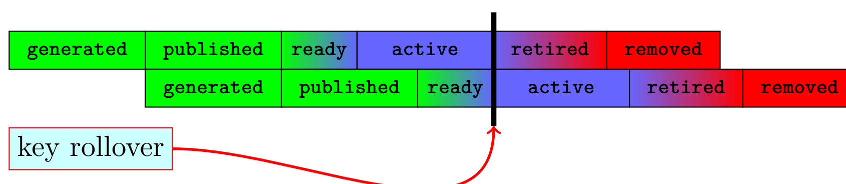
Key Rollover, cont'd

- What would happen if a cache contains signatures generated by a key that has been replaced?
  - if validators can not lookup the key the validation process will fail… which is obviously not good
- Therefore it is necessary to temporarily have both the old key and the new key present in the zone
  - until all RRSIG records generated by the old key have expired



The Key Management Problem

- From an operational point-of-view the problem with key management is one of complexity
  - keys have many **states**: keys must be **generated**, **published** (added to the zone), **active** for a period (used for signing), **retired** (no longer used for signing) and then finally **removed**
  - keys should be stored securely



- Keys also have to be managed on time, because signatures expire
- There's an obvious need for software support here and later on we will take a look at some of the alternatives

Key Rollover, cont'd

- How should "rollover" of Key Signing Keys be managed?
  - the key to the problem is that it involves the parent (because the parent must **sign** the child's keys one way or another)
  - and this makes the KSK rollover operation more complex
- The right solution turned out to be yet another record type, Delegation Signer (`DS`)
  - the crucial change with DS is that it removes the need to install `DNSKEYs` or `RRSIGs` in the other party' s zone file
  - i.e. either the child into the parent zone or vice versa, which turned out to be way too difficult in practice

10.4.1 The `DS` Record

Delegation Signer: The `DS` Record

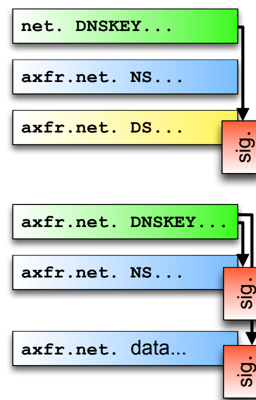
- The `DS` record is **only** present in the parent zone
  - i.e. the `DS` record for "`axfr.net`" is located in the "`net`" zone
  - even though the "owner name" for the record belongs to the child
- This is the **only** record that is authoritative in "the wrong zone" like this (all other records related to a delegation are authoritative in the child, not in the parent)
- The `DS` record consists of several parts:

algorithm	same numerical representation as for `DNSKEY` and `RRSIG`
digest	digest type, 1=SHA1, 2=SHA256
hash	hash data, BASE64 encoded

Delegation Signer

With Delegation Signer a delegation is changed to contain

- `NS` records for the child and possible glue (as usual)
- a secure hash of the child's "key signing key" (this is the `DS` record)
- an RRSIG generated by a parent key of the **hash** (i.e. of the DS record) to prove that the DNS record is authenticated by the parent



The `DS` Record

```
oban# dig axfr.net ds +multi +dnssec
; <<>> DiG 9.7.0 <<>> axfr.net ds +multi +dnssec
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status:  NOERROR, id:  23676
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 4, ADDITIONAL:
3
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;axfr.net.   IN DS
;; ANSWER SECTION:
axfr.net.   3521 IN DS 19105 5 1 (
                   F9556EB9F9A5E5C582B4D307C13E69E2D36F5C9C )
axfr.net.   3521 IN DS 19105 5 2 (
                   CAD472D20247A1CEB13A206F83FA6EA7CC373FB57133
                   099F0F2B3479E48B602D )
axfr.net.   3521 IN RRSIG DS 5 2 3600 20161218220924 (
                   20161212041750 18123 net.
                   bUmS5baRaTI3ITzMySr6Lzav4Cv8Nq ...   )
```


Creating DS Records with dnssec-signzone

- DNSSEC signing tools like **dnssec-signzone** must be able to generate the DS records needed for secure delegation of signed child zones
- To make **dnssec-signzone** generate DS records for signed children two things are necessary:
  - either the child KSK or a DS thereof must be present in the "signing directory". This is the reason for the "**dsset-child**" file generated in the child
  - **dnssec-signzone** must be invoked with the new flag "**-g**", i.e.

```
server% dnssec-signzone [-S] ... -g ... {zonefile}
```

10.5 DNSSEC Debugging

DNSSEC Debugging

- When DNSSEC works everything is fine
- When DNSSEC doesn't work you need to find the problem. How do you do that? There are a number of reasons why this is a bit difficult:
  - While it is possible to look things up with **`dig`** (or similar), it is not easily possible to see whether an **`RRSIG`** validates the RRset or not
  - It is not always easy to see which **`DNSKEY`** has what **`keyid`**
  - While it is possible to see when an **`RRSIG`** is no longer valid, it is lots of text to wade through for each **`RRSIG`** and there may be many **`RRSIG`**s to examine
  - When a validation chain breaks, how do you most easily find the point where it breaks?
- All of this basically boils down to the need for tools that can chase signed data, validate signatures and report on failures
- One such tool that is worth taking a look at is **`drill`**

DNSSEC Debugging, cont'd

It should not be forgotten, though, that the first thing to do is usually to ensure that your name servers actually **log** information about DNSSEC issues (validation problems in particular)

- In Unbound the attribute "**`val-log-level:`**" controls the amount of detail about validation problems that will be logged:

```
server:
    ...
    val-log-level:  2
```

DNSSEC Debugging, cont'd

Another thing that should not be forgotten is that in the end there are primarily two possible causes for failure:

- Some piece of software has a bug. It could be a nameserver providing wrong data data in response to a valid query, or it could be a resolver misinterpreting a valid response.
- Somewhere someone loaded the wrong data into an authoritative nameserver.

The first cause (bugs) is obviously a concern, but the situation is continually and more and more quickly improving, as DNSSEC sees more and more production use.

The second case is really where the problems are.

- Therefore, do not forget "old school" stuff, like verifying the zonefile before loading it into a nameserver.

Zone Checker Tools: `validns`

Regardless of nameserver implementation, a correct zone file is obviously crucial. There are several zone checker tools available to help with syntax verification. "**`validns`**" is one.

- In addition to syntactic checking, **`validns`** is also able to check the zone file against a set of "policies".
- **`validns`** does several DNSSEC related checks, including validation of DNSSEC signatures

```
server% validns -p single-ns -s -z zonename filename
```

a "policy"

- Among the ten defined policies are: "**`single-ns`**", "**`dnskey`**", "**`dname`**" and "**`all`**"
- See `http://www.validns.net/`

Zone Checker Tools: `named-checkzone`

- **`named-checkzone`** is a stand-alone zone checker tool bundled with BIND9 that performs the same zone tests and verifications that BIND9 do when loading a zone
  - this does not include validation of DNSSEC signatures (on the other hand, there is yet another ISC tool for that, called **`dnssec-verify`**...)

check this out

```
server% named-checkzone zonename filename
```

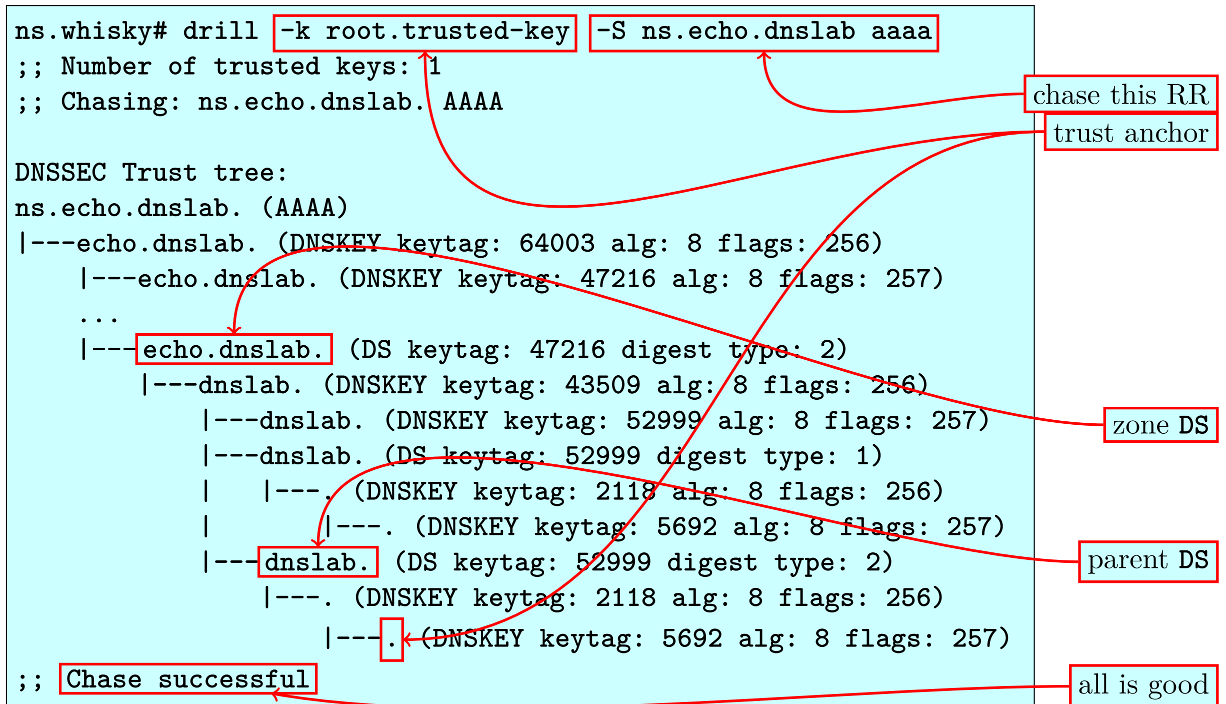
- The modifier "**`-D`**" cause **`named-checkzone`** to produce a version of the zone in canonical format:

```
server% named-checkzone -D -o outfile zonename filename
```

- See "**`man named-checkzone`**" for more details

DNSSEC Debugging using `drill`

`drill` is an alternative to **`dig`**, which among other features is quite good at following and showing a validation chain



DNSSEC Debugging using `drill`, cont'd

```
ns.whisky# drill -k root.trusted-key -S ns.hotel.dnslab aaaa
;; Number of trusted keys: 1
;; Chasing: ns.hotel.dnslab. AAAA

DNSSEC Trust tree:
ns.hotel.dnslab. (AAAA)
|---DNSSEC signature has expired:
ns.hotel.dnslab.  10 IN RRSIG AAAA 8 3 600 ... WSrGXc13... ;{id = 2152}
For RRset:
ns.hotel.dnslab.  10 IN AAAA 3ffe:b80:1:bb::8
With key:
hotel.dnslab.     10 IN DNSKEY 256 3 8 AwEAAZlu7S... ;{id = 2152 (zsk)}
|---hotel.dnslab.  (DNSKEY keytag:  2152 alg:  8 flags:  256)
    |---DNSSEC signature has expired:
hotel.dnslab.      9 IN RRSIG  DNSKEY 8 2 ... THLTZgBxK... ;{id = 55216}
For RRset:
hotel.dnslab.     10 IN DNSKEY 256 3 8 AwEAAZlu7S... ;{id = 2152 (zsk)}
hotel.dnslab.     10 IN DNSKEY 257 3 8 AwEAAbltGl... ;{id = 55216 (ksk)}
With key:
hotel.dnslab.     10 IN DNSKEY 257 3 8 AwEAAbltGl... ;{id = 55216 (ksk)}
    ...
No trusted keys found in tree: first error was: DNSSEC signature has expired
;; Chase failed.
```

the cause of the failure

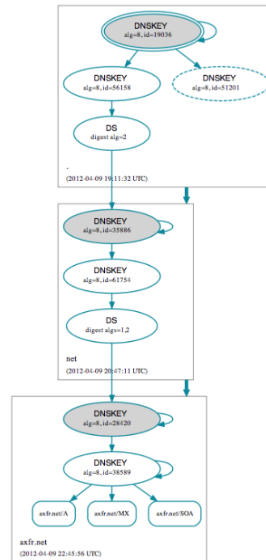
id of key that generated expired RRSIG

oops, not good

DNSSEC Debugging: `DNSViz`, etc al

`DNSViz` is a web-based diagnostic service with a graphical presentation interface. At present source is not available, and hence it can only be used to analyze DNSSEC chains on the public Internet

- Different types of information is conveyed via choice of colour, shape, line types, etc
- By visually presenting **`DNSKEY`**s, **`DS`** records and what key has signed what information it is possible to "see" where a problem is located
- See `http://dnsviz.net/` for details
- Yet another debugging service is found at `http://dnssec-debugger.verisignlabs.com/`



DNSSEC Operations: Parent-Child Interaction

- In the DNSSEC environment the interaction between parent and child changes from:

a one-time thing at time of inception (or at least very infrequent)

  to:

a periodic exchange to communicate new `DNSKEYs`, what key to generate the `DS` record for, etc

- These exchanges obviously need to be authenticated

10.6 DNSSEC Operations

DNSSEC Operations: Changes

- Zone Owner:
  - Significant changes: keys need to be generated and rolled, “signatures” need to be periodically re-generated

  - when the zone `KSK` is rolled over the parent needs to be informed so that a new `DS` record is generated
  - when a child `KSK` is rolled over the child will request a new `DS` record to be generated

- Authoritative Server:
  - The simplest task in a DNSSEC migration: as long as the software is DNSSEC capable there is no key management, no new interaction, no data generation. I.e. **no change**.

- Validator:
  - The management issue here is the trusted keys and how to acquire them
    * with a single island of trust (typically ".") the problem is manageable, but with many islands it explodes
    * with many islands automated tools will be necessary (these are often referred to as "automatic rollover" systems)

10.7 Autenticating Negative Responses

Authenticating Negative Responses

- `RRSIGs` can only sign existing records
  - how then is "denial of existence" authenticated?
  - i.e. how is a "negative answer" signed?
- Yet another record type is needed for that: `NSEC`
  - `NSEC` is a record that "spans" the empty space between two names
  - when a queried domain name does not exist the `NSEC` that covers the interval where the domain name should have been located is returned instead, thereby proving the non-existence
  - `NSEC` is also returned when the domain name does exist, but does not carry the requested record type
    * e.g. query for an `MX` but only an `A` record exists
- The `NSEC` records (and their `RRSIGs`) are sent in the Authority section of the response packet

Importance of Authenticated Denial of Existence

- ADE (Authenticated Denial of Existence) is more crucial to the DNSSEC design than may be immediately apparent
- Without ADE it becomes possible to "spoof away" signed data
  - i.e. a signed RRset could be replaced by an "unsigned NXDOMAIN"
- Among the things that may be "spoofed away" are the DS records that show a delegation to be secure
  - i.e. without ADE it becomes possible to fool a validator to believe a whole signed hierarchy to be unsigned and unsecure

10.7.1 The `NSEC`, `NSEC3` and `NSEC3PARAM` Records

The `NSEC` Record

- `NSEC` "ties together" the empty space between nodes in the hierarchy
- For the "empty space" to be identified it is necessary to sort the existing records in lexicographic order:

```
alpha.axfr.net.   IN A ...
alpha.axfr.net.   IN MX ...
alpha.axfr.net.   IN NSEC bravo.axfr.net. A MX
bravo.axfr.net.   IN MX ...
bravo.axfr.net.   IN NSEC charlie.axfr.net. MX
charlie.axfr.net. IN SRV ...
```

The `NSEC` Record, cont'd

- The response to a query between "**`alpha`**" and "**`bravo`**":

```
; HEADER SECTION: ... rcode: NXDOMAIN
; QUESTION SECTION:
; qname=bar.axfr.net., qclass=IN, qtype=A
; AUTHORITY SECTION:
axfr.net.          IN SOA ...
axfr.net.          IN RRSIG SOA ...
axfr.net.          IN NSEC alpha.axfr.net. SOA RRSIG...
axfr.net.          IN RRSIG NSEC ...
alpha.axfr.net.    IN NSEC bravo.axfr.net. A MX
alpha.axfr.net.    IN RRSIG NSEC ...
; ADDITIONAL SECTION:...
```

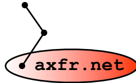
non-existing qname

A Very Simple Zone File

```
$TTL 900
axfr.net.       IN SOA ns.axfr.net. hm.axfr.net. (
                       2016030801  ; serial
                       2700        ; refresh
                       900         ; retry
                       3600        ; expire
                       900 )       ; ncache ttl

axfr.net.       IN NS  ns.axfr.net.
axfr.net.       IN NS  ns.cafax.se.
axfr.net.       IN MX  10 mail.mailservice.net.
ns.axfr.net.    IN A   192.71.80.122
www.axfr.net.   IN A   213.115.163.54
```

A Very Simple Signed Zone File



```
$TTL 900
axfr.net.  IN SOA ns.axfr.net.  hostmaster (
                2016030801 ; serial
                2700       ; refresh (45 minutes)
                900        ; retry (15 minutes)
                3600       ; expire (1 hour)
                900 )      ; ncache (15 minutes)
axfr.net.  IN RRSIG SOA 5 2 900 20160408140155 (
                20160308140155 33635 axfr.net.
                RVEAJhOpptmjkRWRSbK6dy5Ew3RuqM+SdXpb
                zl4TjTc0bm1B4nWvdosSQ0kOmL2UQy+ZJbf2
                PTyYuOEKr9m99ArnGiTIb5lXAtcPTCv3jxA3
                7HXYmkC+zUWECypBu9U3 )

axfr.net.      IN NS ns.axfr.net.
axfr.net.      IN NS ns.cafax.se.
axfr.net.      IN RRSIG NS 5 2 900 20160408140155 (
                  20160308140155 33635 axfr.net.
                  OyGwXu/TLVoIDXNx7Fs/aa/vRI66xjRhvy9v
                  gkZxL2lOG+kbEEWWlMaxKf9GxzBfgLTf8Wc7
                  6mlabJW/8z4qqNXS6dHg8E2jjguRZhS8Axpp
                  s5r9j+5pe0AfoMJIKAaX )
axfr.net.      IN MX 10 mail.mailservice.net.
axfr.net.      IN RRSIG MX 5 2 900 20160408140155 (
                  20160308140155 33635 axfr.net.
                  L79ZanTnCVSyY9ZuracPkylsF5xpSgESGClW
                  ykCIDN10NFKQbeBAujCVpfco+rpIZwO6PQoZ
                  ljS/aS4AtWrPLVYbzV6Q1zbcokL1VJ/u28yY
                  C/yERJrnnSROJgP6nJ+C )

axfr.net.      IN NSEC ns.axfr.net. NS SOA MX RRSIG
                  NSEC DNSKEY
axfr.net.      IN RRSIG NSEC 5 2 900 20160408140155 (
                  20160308140155 33635 axfr.net.
                  BkymQUuNo9NYYIut9ZinUT054d2YI28Yg1VR
                  qml345MHc9i3gi8ceGipiSpV/Y6AHjlkl70P
                  WCZkzI/y3RCKiG9G4fh5AXa3o3AsXUYqRmgV
                  B5bPWWotIDKQRlmVNCVP )
axfr.net.      IN DNSKEY 256 3 5 (
                  AQO8v8I2foH5fYmU21sUjZ/rDFZWbZnHc+LY
                  i0+nXw19sNENJP+mcCDPI/Loh08+U1bYgtVK
                  FhKGfjy815MYOapNEToHla8BXsxtj3cW6MR+
                  HjiRPeTc6on6tK= ) ; key id = 33635

axfr.net.      IN DNSKEY 257 3 5 (
                  AQPSQsNvgAlAMbWZNp1/hRFk3a3uf6VwjCZH
                  y0Lg2EHnDpQgy/metDNCAKi8szieNJ8dpqmH
                  eea2tu7BVOqyydJUn5oCUQ2hkq9en9csreAN
                  t+Un1bjZHBCFxJX= ) ; key id = 8324
axfr.net.      IN RRSIG DNSKEY 5 2 900 20160408140155 (
                  20160308140155 8324 axfr.net.
                  iM5ylXEJ4ndYyRKKYHcwZRE2OfGB6Vu9v/8l
                  ahhwjqJssM9IRN/FSjgycJp1dB5n6Ue0LTfW
                  K/it+f7jVBAQNH7DaMwMCoXDq/Xbi8dspQBL
                  La9vqRFNN71HQjIT4Gyq )
axfr.net.      IN RRSIG DNSKEY 5 2 900 20160408140155 (
                  20160308140155 33635 axfr.net.
                  bsJSsyvyNuCt1U9Unm6sc2T4AQ9boSgq+Ft3
                  [2 lines abbrev'd]...64X/pF+Pm+3oIfydUf0h )
```

```
ns.axfr.net.    IN A 192.71.80.122
ns.axfr.net.    IN RRSIG A 5 3 900 20160408140155 (
                    20160308140155 33635 axfr.net.
                    gvXQLedVV6G+KdGGSk0FgmAefhFJH418D2Qq
                    94FJLj1XtflXgqvsptY++YE82zRA+ELWI9s7
                    RbiEu+1KQRUyGH5m2e4KurE0NcdPr8Q/DRqD
                    a7bZEVfHgA6hd5Y04xWF )
ns.axfr.net.    IN NSEC www.axfr.net.  A RRSIG NSEC
ns.axfr.net.    IN RRSIG NSEC 5 3 900 20160408140155 (
                    20160308140155 33635 axfr.net.
                    ai9/l8nro0mqu0aadlN88Knk6i3Kq80rhM8S
                    CgTPU725MM7i+DrfFqvIox9sPQAjo7n59R0R
                    3U0uq3QsTMAb7khjU1N2ceXFdtKHe1hi+gfh
                    XNDRVUCwLVPe8jdo9hXV )

www.axfr.net.  IN A 213.115.163.54
www.axfr.net.  IN RRSIG A 5 3 900 20160408140155 (
                    20160308140155 33635 axfr.net.
                    dT/ibni+/ra0JVXV+JW2YSNq4HV4LG3PmT7T
                    4nx7tl7zeNBzOSmJCyoTiPwaaRf3pZeN4rWc
                    BhaT+0VX39vUhw9gEV8YQerIj6fsmvm6Clx/
                    gjeCJ7NG+ZCMGbwoWevw )
www.axfr.net.  IN NSEC axfr.net.  A RRSIG NSEC
www.axfr.net.  IN RRSIG NSEC 5 3 900 20160408140155 (
                    20160308140155 33635 axfr.net.
                    DbsJ1KqcNowPa8gRmMzCcr7c9a2E8zdvM4fA
                    l7zfuVP8W/THMYAJXa0dgMDtuGHb8QHmJCNV
                    /riItYMKeImT02h0Ugcq4eNdHC407QL2kYi0
                    UO8DWbeZiXV6YNDS+Urz )
```

The NSEC3 Record

- The **NSEC** record is conceptually easy to grasp, but it does have one essential drawback
- It does allow the entire zone to be listed
  - a resolver is able to query for "**zone. NSEC**" to get the first name in the zone and then "**first.zone. NSEC**" to get the second name, etc, etc
- For many zones this is not an issue
- For some it is a show stopper and for that reason a new method of providing authenticated denial of existence has been developed, the **NSEC3** record
- The **NSEC3** record is documented in RFC 5155

The NSEC3 Record, cont'd

- The main idea with the **NSEC3** record is to instead of chaining together the nodes in the namespace it chains together hashed names in a hashed namespace

– but all names are relative to the zone name, i.e. end up in the signed zone

```
...
ardbeg.axfr.net.    900 IN A 195.54.233.67
ardbeg.axfr.net.    900 IN RRSIG A 5 3 900 ...
mpemmv6m3p3rs3723ct23na4k3qvtum0.axfr.net.
                  3600 IN NSEC3 1 0 1 -
                         scs2b3b9u6cp0n3pf167ed93qm0fh21b A
RRSIG
mpemmv6m3p3rs3723ct23na4k3qvtum0.axfr.net.
                  3600 IN RRSIG NSEC3 5 3 3600 ...
idefix.axfr.net.    900 IN A 192.71.80.122
idefix.axfr.net.    900 IN RRSIG A 5 3 900 ...
scs2b3b9u6cp0n3pf167ed93qm0fh21b.axfr.net.
                  3600 IN NSEC3 1 0 1 - ...  A RRSIG
```

hashed version of “ardbeg”

hashed version of “idefix”

The NSEC3 Record, cont'd

- The complex structure of the **NSEC3** record is mostly due to the need for details about how to compute the hash
  - in the “flags” field, the only defined flag is **OPTOUT**, which signals whether the **NSEC3** chain chains together all nodes or only the authoritative nodes (in the latter case “opting out” the non-authoritative nodes in the zone)

```
...
ardbeg.axfr.net.    900 IN A 195.54.233.67
ardbeg.axfr.net.    900 IN RRSIG A 5 3 900 ...
mpemmv6m3p3rs3723ct23na4k3qvtum0.axfr.net.
                  3600 IN NSEC3 1 0 1 -
                         scs2b3b9u6cp0n3pf167ed93qm0fh21b
A RRSIG
mpemmv6m3p3rs3723ct23na4k3qvtum0.axfr.net.
                  3600 IN RRSIG NSEC3 5 3 3600 ...
idefix.axfr.net.    900 IN A 192.71.80.122
idefix.axfr.net.    900 IN RRSIG A 5 3 900 ...
scs2b3b9u6cp0n3pf167ed93qm0fh21b.axfr.net.
                  3600 IN NSEC3 1 0 1 - ...  A RRSIG
```

hash algorithm, 1=SHA1

flags

hash iterations

salt

types covered

The NSEC3 OPTOUT Flag

The **OPTOUT** flag is really a protocol tweak for one particular purpose: to allow a gradual growth of the DNSSEC zone size for large "delegation mostly" zones, i.e. Top-Level Domains

- **without OPTOUT** the **NSEC3** chain will chain together **all** nodes in the zone (including all unsigned delegations in a TLD zone)
- **with OPTOUT** the **NSEC3** chain will only chain together the **authoritative** nodes, i.e. the authoritative data in the zone plus the signed delegations (unsigned delegations are skipped)

The "cost" of using **OPTOUT** is that it is no longer possible to provide authenticated denial of existence for **unauthoritative** data in this zone

- this is a non-issue for TLDs but may be important to others
- **authoritative** data is still protected (as part of the **NSEC3** chain)

The NSEC3PARAM Record

- Because **NSEC3** chains together hashed versions of the domain names the **authoritative server** has to be able to perform the hash computation
  - it has to compute the hash of the query name to be able to return the correct **NSEC3** segment
  - the **NSEC3PARAM** record exists at the zone apex and lists the hash parameters for the benefit of the authoritative server
  - in the "flags" field, the **OPTOUT** flag is not used and hence the value is today always zero

```
...
axfr.net.           3600  IN  NSEC3PARAM  1 0 1 -
axfr.net.           3600  IN  RRSIG  NSEC3PARAM 5 2 3600 ...
...
ardbeg.axfr.net.     900  IN A  195.54.233.67
ardbeg.axfr.net.     900  IN RRSIG  A 5 3 900 ...
mpemmv6m3p3rs3723ct23na4k3qvtum0.axfr.net.
                     3600 IN NSEC3 1 0 1 -
                          scs2b3b9u6cp0n3pf167ed93qm0fh2lb A RRSIG
```

hash parameters

Using NSEC3 instead of NSEC: dnssec-signzone

- To use NSEC3 semantics instead of NSEC for a zone it is typically only necessary to
  - use keys with an algorithm that is "NSEC3 ok" (eg. "RSA/SHA256 or better")
  - add the keys to the zone (or use "smart signing") and sign the zone with flags for NSEC3 parameters (hash iterations and salt):

```
server% dnssec-signzone -S -3 "abab" -H 1 whisky.dnslab
```

use NSEC3, with salt "abab"

1 hash iteration

    (in the example the zone name must be equal to the file name)
  - to use OPTOUT when signing, use the flag "-A"
  - if needed, update the DS record in the parent zone

Using NSEC3 instead of NSEC: ldns-signzone

- To use NSEC3 semantics instead of NSEC when signing a zone with ldns-signzone it is necessary to
  - use keys with an algorithm that is "NSEC3 ok" (eg. "RSA/SHA256 or better")
  - sign the zone with flags for NSEC3 parameters (hash iterations and salt):

```
server% ldns-signzone -n -s "cafe" -t 1 whisky.dnslab [keys]
```

use NSEC3, with salt "cafe"

1 hash iteration

    (in the example the zone name must be equal to the file name)
  - to use OPTOUT when signing, use the flag "-p"
  - unfortunately the NSEC3 flags are completely different from what is used by dnssec-signzone
  - if needed, update the DS record in the parent zone

Using NSEC3 instead of NSEC, cont'd

- It is possible to have both NSEC and NSEC3 records in a signed zone at the same time

- This may be needed during migration from `NSEC` to `NSEC3` (or vice versa)
- The authoritative server will use either `NSEC` semantics or `NSEC3` semantics but never both
  - if there is an `NSEC3PARAM` record in the zone, then `NSEC3` will be used
  - therefore, if migrating to `NSEC3`, the `NSEC3PARAM` record must be added last, after all the `NSEC3` records are already added

10.8 DNSSEC Policy Issues

DNSSEC Policy Issues

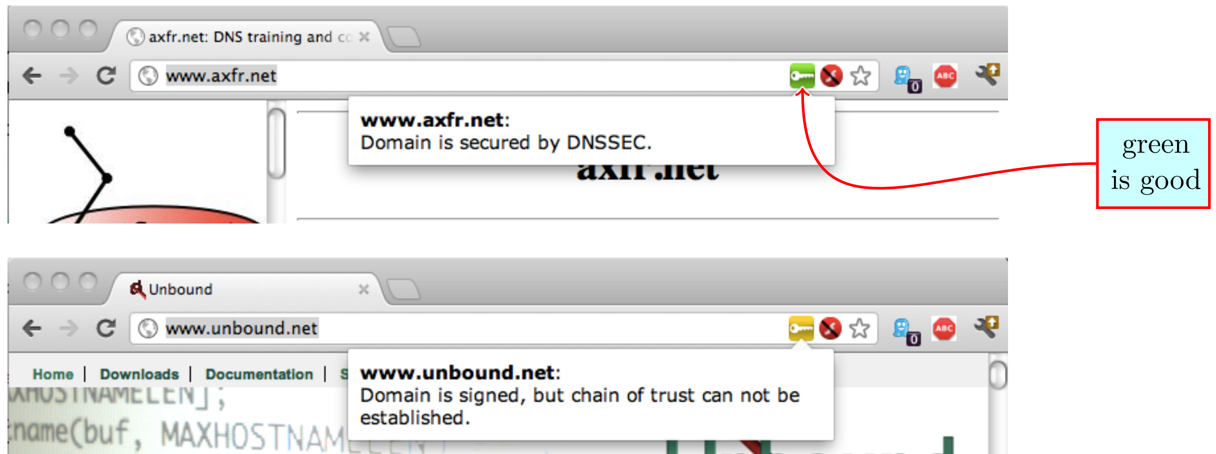
"Wall-to-wall DNSSEC" is a laudable goal, but it is not necessarily the case that this will always be the best choice (especially not during a transition to DNSSEC). Among the policy decisions to be made are:

- Should all zones be signed, or only some of them?
- Should all validation errors be kept from stub resolvers or just validation errors for "critical zones"?
- Should some data not be validated at all even if it is signed?
- Should old design choices (like split-DNS) be kept, even if they become more complicated to maintain in a DNSSEC environment?
- What key rollover schedule should be used? Note that "no rollovers" is a possible choice

11 DNSSEC for Applications

DNSSEC Plugins for Web Browsers

The perhaps most obvious application for DNSSEC is a web browser that does look for validated responses to DNS lookups.



- There are plugins available for most common browsers (at least Firefox, Chrome, Safari and. . . umm. . . Internet Exploder)

`SSHFP`: Secure Shell Fingerprints in DNS

`ssh` is a well-known utility for remote login across the Internet.

For authentication of both the user and the remote server, **`ssh`** uses key pairs.

To be able to know that you're logging in to the **right** machine, the remote machine will identify itself (by sending its fingerprint)

- If the fingerprint is known to the user, login proceeds.

```
imr# ssh www.axfr.net
The authenticity of host 'www.axfr.net' can't be established.
RSA key fingerprint is 65:0d:35:91:63:e7:2f:c1:7d:43:0c:23:8b:fc:b2:8e
Are you sure you want to continue connecting (yes/no)?
```

- If the fingerprint is **not** previously known to the user, it will be printed on the terminal and the user is asked to decide. **Here everyone always just select "yes" and hope for the best**.

A better method: verify the fingerprint from the remote machine against a **validated** `SSHFP` record for that machine (which contains the fingerprint)

SSHFP: Example Configuration

- Compute the **SSHFP** record from the host keys for this machine:

```
# ssh-keygen -r www.axfr.net.
www.axfr.net. IN SSHFP 1 1 b79610d64cac153d78d6703e5...04d2
www.axfr.net. IN SSHFP 2 1 fdaeefa73ec914383180d5a39...4943
```

- Add the **SSHFP** records to the zone and resign the zone
- Update the **ssh** configuration to trust keys that match **SSHFP** fingerprints **if the RRSIG for the SSHFP validates**:

```
Host *
    VerifyHostKeyDNS yes
```

- Result:

```
imr# ssh www.axfr.net
www.axfr.net#
```

11.1 DANE

(Non-DNS) Trust Infrastructure

The Trust infrastructure on the Internet is mostly based on TLS and PKIX (RFC 5280)

- Certificate Authorities verify that a cryptographic keypair belongs to a named entity
- All CA signatures are equally valid
- An average browser trusts 1500 of them
  - If **company.net** has acquired a cert from one CA there is nothing precluding one of the other CAs from making a mistake and issue a cert for **company.net** to some other entity

- To be able to do "bad things" it is sufficient to compromise one Certificate Authority (and this has happened way too many times already)

If the DNS infrastructure is secure, can this be improved?

- If nothing else, it would be nice to be able to lock down **exactly which CA** our cert should come from

`DANE`: DNS-Based Authentication of Named Entities

The main idea of the **`DANE`** working group in the IETF is that that a certificate may be authenticated via a DNSSEC signature instead of (or in addition to) being authenticated by a PKIX CA

- There are a number of possible alternatives depending on whether it is the end cert, the CA cert, a hash of either, etc, which is stored in DNS

Benefits that **`DANE`** will bring:

- Tie a certificate to a named service outside of TLS-sessions
- Allow only 1 CA to issue certificates for an organization
- Create your own CA
- Self-signed certificates

How does it work?

- The certificate data is signed and published in DNS
- The DNSSEC Chain of Trust is used for authentication

`DANE` published its first RFC autumn 2012, which specifies the new record type **`TLSA`**

DANE: The TLSA Record (RFC 6698)

- Given the number of alternatives that needs to be covered, the **TLSA** record has to be rather complicated
- It is clearly not intended for human consumption...

```
_443._tcp.www.axfr.net. IN TLSA ( 3 0 1
                      5819d4c63da043785bf88a9c1ae6f4d3
                      f56a4072376d64d7fb89be242bce65b1 )
```

Usage: how should the matched certificate be used:

0	**PKIX-TA**: CA certificate
1	**PKIX-EE**: End Entity, must chain to a CA certificate
2	**DANE-TA**: Use as trust anchor
3	**DANE-EE**: End Entity, no need for a CA

Certificate Association Data: the exact bytes to be matched, represented in hex

Matching Type: how the association data is matched:

0	Full certificate
1	SubjectPublicKeyInfo

Selector: what part should be matched:

0	Full data
1	SHA-256 hash
2	SHA-512 hash

DANE for TLS: SMTP

This is now an RFC (7672), and there is lots of interest and already implementations with released DANE support

Problems with TLS support prior to DANE include:

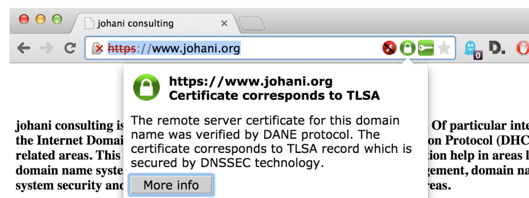
- STARTTLS downgrade attack (due to the need to support both TLS and non-TLS connections)
- Insecure server name without DNSSEC
- Too many CAs able to issue certificates for any domain

- Postfix v2.11+ (released spring 2014) has **DANE** support.
- Exim has **DANE** support in development.

See **http://datatracker.ietf.org/doc/draft-ietf-dane-smtp-with-dane/** for specification status

More DANE for TLS (HTTPS in particular)

- NIC.CZ has created a TLSA validator plugin for most common web browsers



- Note that the poster child service, "web browsing", is very important, but also one of the more complicated services to implement, because web browsers are used in all sorts of devices and in all sorts of broken network environments.
  - There is ongoing work on adding **DANE** support directly in Mozilla/Firefox and the most likely path is by integrating **libunbound**, i.e. the API towards the Unbound validating resolver
  - Meanwhile, the plugin from NIC.CZ works fine

DANE: Beyond TLS

- TLS was the obvious first step. The next steps are to specify how to use **TLSA** together with various application protocols, and it is clear that there is a lot of interest in utilizing the "DANE model".
- Among the use cases currently implemented or under discussion are:
  - **DANE**-for-OpenSSL
    - There is experimental **DANE** support in OpenSSL (added May 2013). Not yet complete, but very interesting none the less
  - **DANE**-for-S/MIME
  - **DANE**-for-**XMPP** (i.e. the **jabber** instant messaging protocol)
    - In the **XMPP** case it seems that the most likely path forward will be to use the **DANE** support in OpenSSL, once that stabilizes
- It is clear that the field of applications and infrastructure that may be improved by leveraging from DNSSEC availability is growing

Another approach to improve on X.509: `CAA` (RFC 6844)

The **`CAA`** record is also designed to address the problem with certificates issued by the wrong CA. However, instead of aiming for the users (via validating recursive servers) to check the idea is that the **Certificate Authorities** should check for the existence of the **`CAA`** record

- If the CA is not listed, then it should not issue a certificate

```
axfr.net.      CAA 0 issue "ca.example.net"
axfr.net.      CAA 0 iodef "mailto:security@axfr.net"
```

- The "**`issue`**" tag specifies the CA authorized to issue certificates for this domain.
- The "**`iodef`**" tag specifies a contact URL for possible certificate policy violations

DNSSEC is strongly recommended for **`CAA`**

12 DNSSEC Tools

How to Keep the Zone Signed

An unsigned DNS zone will not stop working, regardless of how old it is (as long as the master allows slaves to verify that it has not changed).

- This obviously changes when deploying DNSSEC, because all the **`RRSIG`**s have expiration times.
- Suddenly it therefore becomes crucially important to find a process to **keep** the zone signed, i.e. with periodically regenerated signatures.

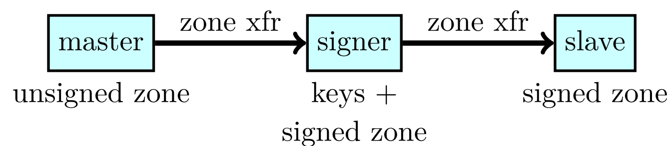
The old method of "if we make a change to the zone then we'll bump the serial number and reload the master" is no longer sufficient.

- What should the new method be?

Keeping the Zone Signed, cont'd

The obvious solution is to move the responsibility for resigning the zone and reloading the nameserver **from** the DNS administrator **to** software.

- The most popular approach for this seems to be a "bump-on-the-wire" design, where the new operation happens outside of the current zone generation machinery



- The "signer engine" runs continously, and re-signs RRsets as needed
- Not requiring any changes to the current zone maintenance process is a major benefit from an operational point-of-view

Keeping the Zone Signed, cont'd

From an operations perspective there are two primary alternatives to achieve the bump-on-the-wire design

- New software, i.e. run a so-called "signer engine" between the zone generator and the authoritative slaves.
  - There are several software alternatives that we will look at.
- Signing-as-a-service, i.e. outsource the signing operation.
  - This is mostly talked about in a TLD context, where DNS service providers are trying to provide a more sophisticated service.
  - ...but it is beginning to appear also in the general DNS market, especially for customers willing to outsource also the zone generation to a DNS provider.

Interesting DNSSEC Software

Here are a few pointers to things that are available, under development and generally of interest:

- OpenDNSSEC (`http://www.opendnssec.org/`). A major effort signer engine (see coming slides).
- **`softhsm (http://www.opendnssec.org/)`**. A HSM (secure key store) emulator developed by the OpenDNSSEC folks
- **`trustman / dnssec-tools (http://www.sparta.com/)`**. A signer engine plus various surrounding tools.
- **`phreebird / phreeload (http://dankaminsky.com/phreebird/)`**. Intended as a "zero configuration" DNSSEC alternative.
- **`zkt (http://www.hznet.de/)`**. A kludge in the master that will sign zones automatically, given contraints. Bad choice, but in quite wide use.

12.1 OpenDNSSEC

OpenDNSSEC

- A major effort, initially by a consortium of TLD registries, now by NLNetLabs
  - including `.SE`, `.UK` and `.NL` (and a number of consultants)
  - in use (or in evaluation) by many other TLDs and also the root
- There are several key design ideas in OpenDNSSEC that distinguish it from other implementations:
  - keys are always stored in HSMs
  - the signer is intended to be run continously, and sign zones (or parts of zones) as needed
  - strong support for separating policy from implementation based on work developing "KASP" (Key and Signing Policy)
- As other implementations, OpenDNSSEC will take care of all the key management automatically
- First release version appeared early 2010

OpenDNSSEC Versions

The initial phase of the OpenDNSSEC project (with development spread across `.SE`, `.UK` and NLNetLabs) concluded in 2014 with version `1.4.x`.

At that point the project was transferred to NLNetLabs and they initiated a significant restructuring of the internals. This work has led to the release of OpenDNSSEC `2.0` in July 2016 (current version is `2.0.1`).

The differences between `1.4.x` and `2.0` is mostly on the inside, i.e. the restructuring work was needed to make the code more easily maintained and more robust, the aim was not to make major changes to the usage model of OpenDNSSEC.

12.1.1 HSMs and `softhsm`

HSM: Hardware Security Module

- A HSM is a [tamper-resistant] device that provides the cryptographic facilities necessary for securing transactions. Its services consist of secure key generation and storage, signature generation and key pair management.
- A primary advantage of a HSM is that it they make it really difficult to make a **copy** of a private key
- To communicate with a HSM device the API PKCS#11 is used.

softhsm

softhsm is developed by the same people developing OpenDNSSEC but it is an independent piece of software

- it is possible to use **softhsm** as the key store for any PKCS#11–aware application, not only OpenDNSSEC
  - including BIND 9.7.0 and later
- the idea is that, given **softhsm**, there is just no reason **not** to store the keys in an HSM, which brings a number of advantages
- in the labs you will use **softhsm** for key storage
- to initialize the **softhsm** key store the "**softhsm**" command line utility will be used

```
# softhsm --init-token --slot 0 --label Frobozz
```

What is the Purpose of the HSM?

The HSM is clearly intended to improve the integrity of the DNSSEC keys. That, however, is not the only benefit.

- Without a HSM the previous mess of lots of "key files" with unpredictable names remain
- It is not obvious which key files represent currently active keys, which are for future use and with are even for retired keys
  - tracking keys and deleting old keys is error prone and dangerous
- The HSM "cleans this up" by moving all the keys out of sight, and standardizing the access mechanism (PKCS#11)

Therefore, a HSM (even if it is just an emulator, like **softhsm**) is a major improvement from a management point-of-view

- Given recently published cryptographic attacks against many HSMs (July 2012), it is likely that in many cases the management benefit is the most important issue

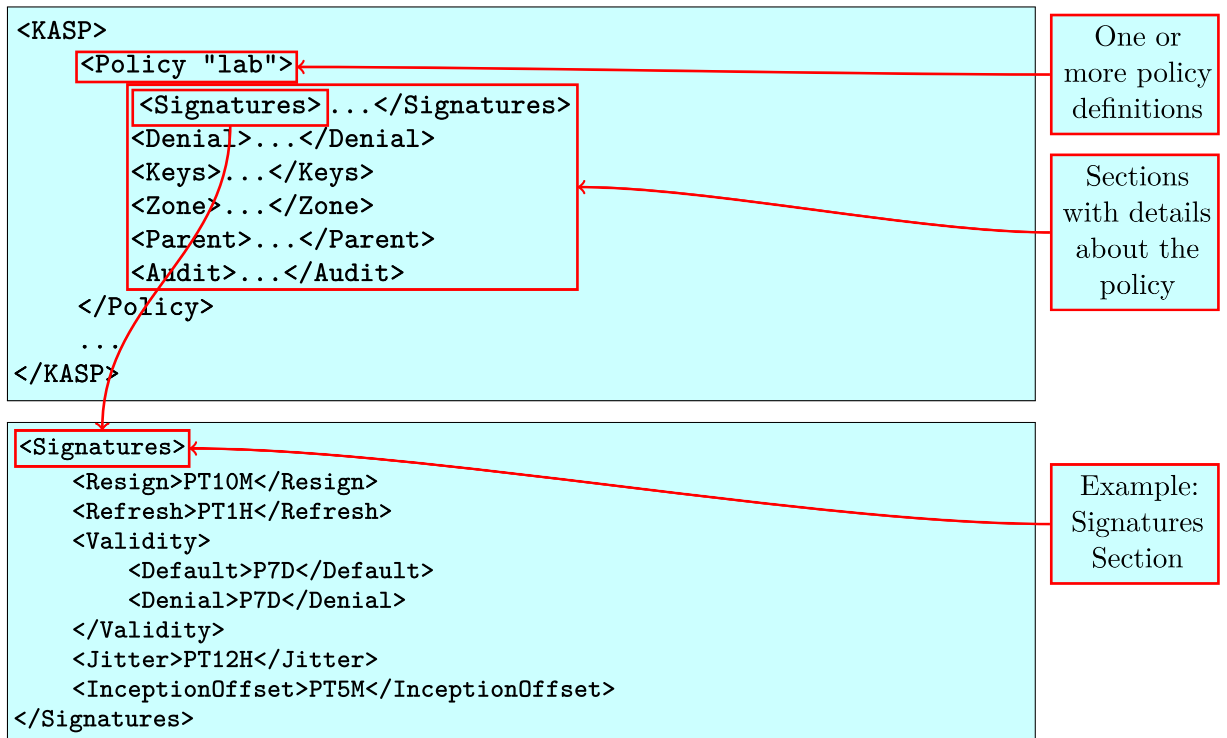
OpenDNSSEC: Configuration

OpenDNSSEC configuration is defined in XML-format in text files. The intent is to over time develop CLI tools to manage the XML, but at present it is mostly done with a text editor

- The main configuration files are

File	Purpose
`conf.xml`	general configuration
`kasp.xml`	policies
`zonelist.xml`	zone configuration

OpenDNSSEC: `kasp.xml`: Policies



OpenDNSSEC, cont'd

- There are many components to OpenDNSSEC. A few of the more central are listed here:
- Command line utilities:
  - `ods-ksmutil`: a tool to manage zones and policies
  - `ods-signer`: a tool to interact with the signer engine, inquire about upcoming tasks, force immediate re-signing, etc
- Other, less visible, components:
  - `ods-enforcerd`: the "signer engine", always running, which will invoke different signer components as needed
  - `ods-auditor`: a zone verification tool

12.2 Updating Trust Anchors

Key Management: The Role of RFC5011

- Apart from the timing issues there is another problem with key rollovers and that is the risk of blind-siding validators
- A "protocol" to let the zone owner signal to validators that a rollover is coming up so that they will be aware of the need to update the trust anchor is very important
- This is exactly what RFC5011 specifies:
  - new keys are added to the zone and are signed by the old key (which the validator has configured as a trust anchor)
  - because the new key is signed by the trusted key the new key may be trusted and actually be configured as a new trust anchor
  - old keys that are no longer needed are "revoked" which signals to the validator that they should be removed
- While the key rollover also with RFC5011 may be manual, the validator side must have software support for automation.

12.2.1 Unbound

RFC5011 Implementations: Unbound

The `auto-trust-anchor-file:` attribute is a variant of the **trust-anchor-file:** attribute described previously

- as the name "**auto-**" implies the difference is that trust anchors in the file will be tracked by RFC5011 probing several times per month and automatically updated as needed

```
# unbound.conf
server:
    auto-trust-anchor-file: root.anchors
    trust-anchor-file:      example.com.anchors
```

- as the RFC5011 tracked trust anchors are updated in the defined file (in the example "**root.anchors**") that file must be in a writable location

dnssec-signzone "Smart Signing"

- "**dnssec-signzone -S**" has been mentioned earlier
  - it depends on "timing metadata" stored with the keys
  - therefore **dnssec-keygen** now supports timing parameters for key state transitions:

```
# publish now, activate in two days, retire in a month
# revoke five days after retirement and then delete
server% dnssec-keygen ... -P now -A now+2d -I now+30d
                          -R now+35d -D now+40d axfr.net
```

- This makes it possible to generate a series of keys with different parameters and then just periodically sign:

```
% dnssec-keygen ... -P now    -A now    -I now+4w ...
% dnssec-keygen ... -P now+3w -A now+4w -I now+8w ...
% dnssec-keygen ... -P now+7w -A now+8w -I now+12w ...
% dnssec-signzone -S axfr.net
```

dnssec-signzone "Smart Signing", cont'd

- The **dnssec-settime** utility enables updating of the timing metadata for existing keys:

```
# publish now, activate in two days, retire in a month
# revoke five days after retirement and then delete
server% dnssec-settime -P now -A now+2d -I now+30d
                       -R now+35d -D now+40d
                       Kaxfr.net.+008+61489
```

- There is also a separate tool, **dnssec-revoke**, to revoke a key immediately (subsequent re-signing needed)

  ```
  server% dnssec-revoke Kaxfr.net.+008+61489
  ```

  - key revokation is (only) used for implementing RFC5011 key rollover signalling
  - note that revocation changes the KEYID of the key

OpenDNSSEC vs BIND 9.9 "inline-signing"

- **The most interesting part is that so much is happening**
- From a purist point-of-view, the "bump-on-the-wire" approach championed by OpenDNSSEC has major benefits
  - the functional separation of "generating (unsigned) zone", "signing zone" and "serving zone" is very good

- via "**`inline-signing`**" BIND9 now also supports this model

- OpenDNSSEC has a better approach to key generation and storage with the decision to always use an HSM
  - although, BIND 9.7+ also has some HSM support
- From an "simplicity" POV, the BIND 9.9 design has major benefits, with no new components to deal with
- From a "complexity" POV BIND9 is morphing into something more than a name server, which may be bad from a "vendor lock-in" perspective
  - both BIND9 and OpenDNSSEC trades simplicity of operation for complexity elsewhere, which is probably unavoidable

Delegation

- Both parent and child have (as usual) `NS` records for the child zone
  - only the child has `RRSIGs` for the `NS` RRset (since the child is authoritative)
  - only the parent has the `DS` record (since the DS record breaks the normal authority algorithm)
- When an signed parent gives a referral this will be in the form of

for a "secure" child:	`NS` + `DS` + `RRSIG(DS)`
for an "unsecure" child:	`NS` + `NSEC` + `RRSIG(NSEC)`

CNAME

- DNSSEC has the consequence that
  - the signing tool (`dnssec-signzone` or `ldns-signzone`) must generate `RRSIG`s for `CNAME` records in the same way as for other records in the zone
  - traditional semantics change to allow for `DNSKEY`, `RRSIG` and `NSEC` and `NSEC3` records for the same owner name as a `CNAME`
  - a resolver is not allowed to follow a `CNAME` if the information it is looking up is a `DNSKEY`, `RRSIG` or `NSEC/NSEC3`
  - if a `CNAME` is found the associated `RRSIG` should be propagated downstream automatically

Issues when Deploying DNSSEC

- New requirements that DNSSEC introduce:
  - significantly larger messages
    - a lookup of an A record may easily return 2K data
    - more data to transfer and store (in particular if `NSEC` or `NSEC3` w/o `OPTOUT`) is used
  - potential for issues with various firewalls and other border protection that may disallow the large packets
  - a small risk of increased frequency of TCP transactions, although EDNS(0) is designed to minimize this
- To verify keys (all the way up to the root) will generate more queries than today
  - the referral chain is complemented by a signature chain and the result is a greatly increased number of chained records
- However, performance is **not** a DNSSEC issue to be concerned about. The real issues are the **process changes** and potential for **new failure modes**

Things Not Covered Here

While this is an advanced course that assumes that all the material in the Introductory DNS course has already been covered, there are additional topics that did not fit here. Among them are:

- DNS and firewalls: forwarding, split-DNS, `stub` zones, complexity
- `TSIG` and `TKEY`
  - Dynamic generation of `TSIG` secrets between servers
- DHCP and the interaction between DHCP and dynamic DNS updates
- DHCPv6 and alternatives for address management in IPv6 networks
- Dynamic DNS Updates
  - both for open source environments (BIND) and Windows
  - TSIG-based and SIG(0)-based authentication
  - usage models and authorization policies

Thank you!

dns-training@axfr.net

Index

$ORIGIN ... 20
$TTL ... 20
iterative mode resolver, IMR ... 12

A record ... 20
attributes ... 12
Authority section ... 7
autotrust ... 88
AXFR ... 29

BIND10 ... 7
BIND9 ... 7
- inline-signing directive ... 94
- logging ... 17

CAA record ... 86
cache pollution ... 34

DANE ... 84
dig
- flags ... 19

dig ... 16, 18
DNS protocol ... 4
DNS spoofing ... 36
DNSKEY record ... 56
DNSSEC
- DNSKEY record ... 56
- DS record ... 63
- NSEC3PARAM record ... 78
- NSEC3 record ... 76, 77
- NSEC record ... 72
- RRSIG record ... 57, 58
- authenticated denial of existence 72
- key management ... 58
- tools ... 88

dnssec-keygen ... 50
dnssec-signzone ... 53
DNSViz ... 70
drill ... 16, 68
DS record ... 63–65

EDNS(0) ... 32

Hardware Security Module, HSM 88, 89
Header section ... 5

inline-signing (BIND9 directive) 94
IXFR ... 30

Kaminsky attack ... 36
KASP ... 88
kdig ... 16
key management ... 58, 62
key rollover ... 62, 63
key signing keys ... 59
key states ... 62
khost ... 16
Knot-DNS ... 7
Knot-DNS Resolver ... 7
KSK ... 59

lab environment ... 10
lab imr machine ... 10
lab server machine ... 10
ldns-keygen ... 51
ldns-signzone ... 54

MX record ... 20

named-checkzone ... 22, 68
NLNetLabs ... 9
NOTIFY ... 30
NS record ... 20
NSD ... 7, 9, 10, 23
- AXFR keyword ... 30
- NOKEY keyword ... 30

server: attribute 24
zone: attribute 23, 24, 30
attributes 23
configuration 23
nsd-checkconf 14
nsd-control (NSD4) 26
commands 26
nsd-control-setup (NSD4) 26
nsd.conf 23, 24
NSD4 7
pattern: attribute 25
remote-control: attribute . . 26
NSEC record 72
NSEC3 record 77
NSEC3PARAM record 78

ods-auditor 91
ods-enforcerd 91
ods-ksmutil 91
ods-signer 91
OpenDNSSEC 88

pattern: (NSD4 feature) 25
PowerDNS 7
PowerDNS Recursor 7
priming 13
protocol 4

query 4
query logging
tcpdump 27
Unbound 27

RFC5011 92
support in Unbound 92
RIPE NCC 9
root.hints 12
RRSIG record 57, 58, 62

SOA record 20
softhsm 88, 90
SSHFP record 81

tcpdump 27
terminating dot 20
timing metadata 93
TLSA record 84
tools
dig 16, 18
dnssec-keygen 50
dnssec-signzone 53, 65
drill 16
kdig 16
khost 16
ldns-keygen 51
ldns-signzone 54
named-checkzone 22, 68
validns 21, 67
autotrust 88
dnssec-keygen 93
dnssec-revoke 94
dnssec-settime 94
dnssec-signzone 93
DNSViz 70
drill 68
nsd-checkconf 14
nsd-control (NSD4) 26
nsd-control-setup 26
ods-ksmutil 91
ods-signer 91
OpenDNSSEC 88
softhsm 88, 90
unbound-anchor 44
unbound-checkconf 14
unbound-control 15
unbound-control-setup 15

Unbound 10, 12
log-queries: attribute 27
access control 14
access-control: attribute . . 14
attributes 12
auto-trust-anchor-file:
attribute 43, 93

configuration 12
`interface:` attribute 13
query logging 27
`remote-control:` attribute . . 15
root hints configuration 13
`root-hints:` attribute 12
`server:` attribute 12
`unbound-anchor` 44
`unbound-checkconf` 14
unbound-control
commands 16
`unbound-control` 15, 16
`unbound-control-setup` 15
`unbound.conf` 12
validation problems
logging 66
`validns` 21, 67

Yadifa 7

`zkt` 88
zone file
`$ORIGIN` 20
`$TTL` 20
zone file format 20
zone signing keys 59
ZSK 59