DNS & BIND 9 DNSSEC WORKSHOP

JAN-PIET MENS, CARSTEN STROTMANN

Version 3.25, 29.09.2023

TABLE OF CONTENTS

1. COURSE NOTES AND EXERCISES . . . 1

1.1. General information . . . 1

2. NEW TERMINOLOGY USED IN DNS & BIND . . . 2

3. DNS 1X1 . . . 3

3.1. DNS - The Domain Name System . . . 3

3.2. DNS - The Domain Name System . . . 3

3.3. DNS namespace . . . 3

3.4. DNS namespace . . . 4

3.5. DNS namespace - "Label" . . . 5

3.6. DNS namespace - "Label" . . . 6

3.7. DNS namespace - "Label" . . . 7

3.8. DNS namespace . . . 8

3.9. Internet namespace . . . 11

3.10. Internet namespace . . . 11

3.11. DNS components . . . 12

3.12. DNS components - Hybrid-DNS . . . 13

3.13. DNS components - Authoritative Only . . . 14

3.14. DNS components - DNS Resolver . . . 15

3.15. DNS message format . . . 16

3.16. DNS Resource Records . . . 16

3.16.1. SOA Record . . . 16

3.16.2. Negative Caching . . . 17

3.16.3. NS record . . . 17

3.16.4. NS record . . . 18

3.16.5. Glue (1) . . . 18

3.16.6. Glue (2) . . . 19

3.17. Exercise: . . . 20

4. AN INTRO TO DNSSEC . . . 22

4.1. DNS Security (or lack of) . . . 22

5. CRYPTOGRAPHY IN DNS (SHORT) . . . 25

5.1. Cryptography for DNS Admins . . . 25

5.2. Symmetric Cryptography . . . 25

5.2.1. Confidentiality: Not the Goal . . . 26

5.2.2. Message Digest . . . 26

5.2.3. Message Authentication Codes . . . 26

5.3. Asymmetric Cryptography . . . 27

5.3.1. Two Applications for public key (PK) Encryption . . . 28

5.3.2. PK Application 2: Encryption for Authenticity . . . 29

5.3.3. PK Application 2: Encryption for Authenticity: Digital Signatures 29

5.3.4. Digital Signatures in DNSSEC 29

5.4. DNSSEC 29

5.5. TSIG vs. DNSSEC 30

5.6. DNSSEC:Design Choices Made 30

6. DNSSEC IN A NUTSHELL 32

6.1. DNSSEC records 32

6.1.1. RRSIG 32

6.1.2. DNSKEY 36

6.2. the chain of trust in DNS 38

6.2.1. DS Record 38

7. DNSSEC SIGNING AND VALIDATION 41

8. DNSSEC VALIDATION 45

8.1. Dnssec in DNS Messages 45

8.2. Basics of DNSSEC validation 46

8.3. Optimizations to the default configuration of a validating DNSSEC resolver 65

9. DNSSEC SIGNING WITH BIND 9 67

9.1. Manual zone signing 67

9.2. Manual signing with BIND 9 67

9.3. Exercise: an authoritative DNS Server 70

9.4. Exercise: Signing BIND 9.6 style (manual signing) 71

10. DNSSEC KEYS 73

10.1. Algorithms for DNSSEC 73

10.2. DNSSEC key sizes for RSA algorithm 74

10.2.1. Problems with large DNS response messages 74

10.3. KSK and ZSK 74

10.3.1. KSK: 74

10.3.2. ZSK: 75

10.4. BIND 9: KSK/ZSK signature over the DNSKEY record set 75

10.5. Lifetimes of DNSSEC keys 75

10.6. DNSSEC Signer "Best-Practices" 75

11. DNSSEC WITH DYNAMIC ZONES 77

11.1. DNSSEC with a dynamic zone 77

11.1.1. DNSSEC key creation with timing events 77

11.1.2. revoking DNSSEC keys 78

11.1.3. Zone definition for dynamic zone with DNSSEC 78

11.1.4. Adding a new record to a dynamic zone 79

11.1.5. Removing a DNS record from a dynamic zone 79

11.1.6. DS records for dynamic DNSSEC zones 79

11.2. Exercise: DNSSEC signing a dynamic DNS zone with BIND 9 80

12. DNSSEC 'INLINE'-SIGNING . . . 83
12.1. DNSSEC with 'inline-signing' . . . 83
12.2. Exercise: DNSSEC with 'inline-signing' . . . 84
12.3. Where is the AD-Flag? . . . 86
13. NSEC VS. NSEC3 . . . 87
13.1. Authenticated Denial of existence - NSEC vs. NSEC3 . . . 87
13.2. Issues with NSEC . . . 87
13.3. The NSEC3 record . . . 88
13.4. NSEC3 Chain . . . 88
13.5. NSEC3-Parameter . . . 88
13.6. NSEC3-Parameter Record . . . 89
13.7. NSEC3 iterations and salt . . . 89
13.8. Problems with NSEC3 . . . 89
13.9. RFC 9276 - Guidance for NSEC3 Parameter Settings . . . 89
13.10. NSEC3 needed? . . . 90
13.11. NSEC3 Narrow-Mode . . . 90
13.12. Exercise: sign a static zone with NSEC3 instead of NSEC . . . 90
13.13. Use of NSEC3 with inline-signing or dynamic signing . . . 90
13.14. Example for inline-signing: . . . 91
13.15. Example for dynamic zone signing . . . 91
13.16. Switching a dynamic zone from NSEC to NSEC3 . . . 91
14. SECONDARY DNS SERVER . . . 92
14.1. Exercise: a secondary server for the DNSSEC signed zone . . . 92
14.2. Exercise: Adjust the primary zone to contain the secondary server . . . 93
15. DNSSEC KEY-ROLLOVER . . . 94
15.1. Why DNSSEC rollover . . . 94
15.2. The Challenges . . . 94
15.3. DNSSEC Keyrollover Documentation . . . 94
15.4. Key rollovers, when and how often? . . . 94
15.5. ZSK Rollover . . . 95
15.5.1. ZSK - pre-publication - Step 1 . . . 95
15.5.2. ZSK - pre-publication - Step 2 . . . 95
15.5.3. ZSK - pre-publication - Step 3 . . . 95
15.5.4. ZSK - pre-publication - Step 4 . . . 95
15.5.5. ZSK - pre-publication - Step 5 . . . 95
15.5.6. ZSK - pre-publication in pictures . . . 96
15.6. Exercise: ZSK Rollover . . . 98
15.7. KSK Rollover . . . 99
15.7.1. KSK - double-signing - Step 1 . . . 100
15.7.2. KSK - double-signing - Step 2 . . . 100

15.7.3. KSK - double-signing - Step 3 . . . 100
15.7.4. KSK - double-signing - Step 4 . . . 100
15.7.5. KSK - double-signing in pictures . . . 100
15.8. Exercise: KSK Rollover . . . 103
16. DNSSEC KEY AND SIGNING POLICY . . . 105
16.1. Default policy . . . 105
16.2. Writing Custom DNSSEC Policies . . . 105
16.3. Additional configuration of custom policy . . . 106
16.4. Key-and-Signing-Policy (KASP) syntax . . . 106
17. EASY DNSSEC WITH BIND 9.16 "DEFAULT-POLICY" . . . 107
17.1. DNSSEC signing the zone with KASP . . . 107
17.2. Migration BIND 9 DNSSEC zu einer KASP Policy . . . 108
18. DNSSEC KEY-ROLLOVER AUTOMATION . . . 111
18.1. Automating DNSSEC Delegation Trust Maintenance (RFC 7344/8078) . . . 111
18.1.1. Bootstrapping DNSSEC with CDS/CDNSKEY . . . 115
18.1.2. Bootstrapping DNSSEC trust . . . 115
18.1.3. Further Reading . . . 116
19. DNSSEC BEST-PRACTICES . . . 117
19.1. DNSSEC br0k3n? . . . 117
19.2. SHA1 phase out in Red Hat EL 9 . . . 117
19.3. Avoid SHA1 . . . 118
19.4. Avoid UDP fragmentation . . . 118
19.5. Avoid UDP fragmentation - DNS authoritative Server operators . . . 118
19.6. Avoid UDP fragmentation - DNS Resolver operators . . . 118
20. DNSSEC TROUBLESHOOTING . . . 119
20.1. checking DNS resolution issues . . . 119
20.1.1. dig . . . 119
20.1.2. external web services to check DNS . . . 120
20.2. looking into DNSSEC validation issues . . . 120
20.2.1. DNSSEC validation troubleshooting with "delv" . . . 120
20.2.2. Message trace . . . 120
20.2.3. Validation trace . . . 121
20.2.4. DNSSEC path validation with drill . . . 121
20.3. Exercise: DNSSEC troubleshooting . . . 121
21. NEWS . . . 122
22. DNSDIST DNS LOAD-BALANCER . . . 123

CHAPTER 1. COURSE NOTES AND EXERCISES

1.1. GENERAL INFORMATION

- This guide and the slides can be accessed online at https://dnssec.defaultroutes.org [https://dnssec.defaultroutes.org]
- Please replace the wildcard characters `XX` in the guide with your participant number. You can find your participant number in the table on this page

CHAPTER 2. NEW TERMINOLOGY USED IN DNS & BIND

- The terms `master` and `slave` have been used to describe primary and secondary authoritative DNS servers in the past.

  However this terminology is wrong and misleading, for reasons discussed in the Internet Draft *Terminology, Power, and Inclusive Language in Internet-Drafts and RFCs*: https://tools.ietf.org/html/draft-knodel-terminology [https://tools.ietf.org/html/draft-knodel-terminology]

- In this document, and in configuration examples, we are using the new terms `primary` (instead of `master`) and `secondary` (instead of `slave`) whenever possible.
- BIND 9 has started adopting the new terms with BIND 9.14, however the transition is not complete, and some terms in configuration statements still use the old terms. This will change with future releases

  If you use an older version of BIND 9, please substitute the new terms for the older ones

- The old terminology will also be found in older books and standards documents (RFCs and Internet Drafts)
- DNS terminology can be confusing and is sometimes overloaded. RFC 8499 *DNS terminology* (https://tools.ietf.org/html/rfc8499 [https://tools.ietf.org/html/rfc8499]) attempts to collect and document the current usage of DNS terminology.

CHAPTER 3. DNS 1X1

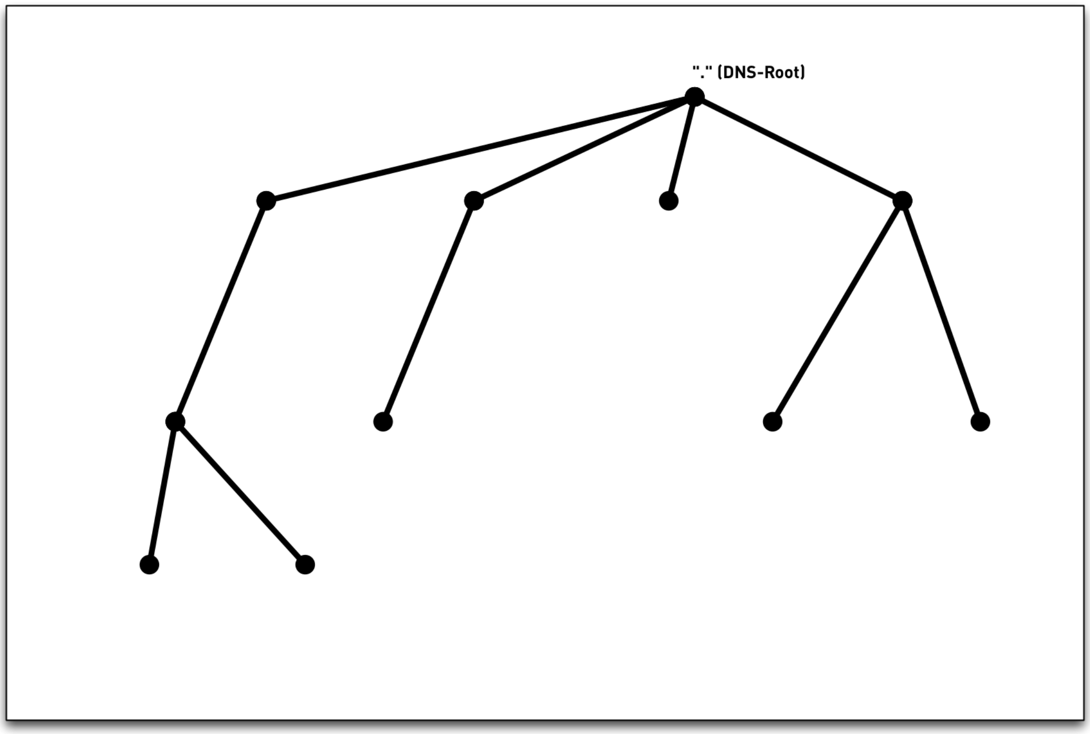
3.1. DNS - THE DOMAIN NAME SYSTEM

> DNS is like chess
>
> the rules are simple
>
> but the chances to loose the game
>
> are endless

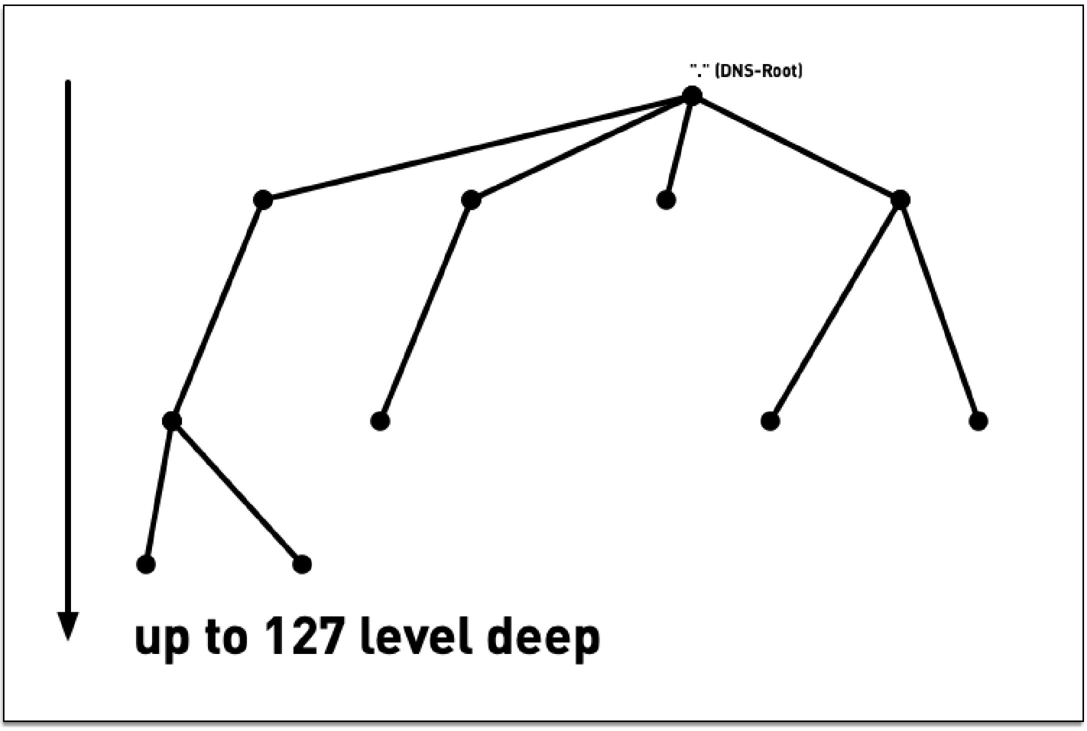
3.2. DNS - THE DOMAIN NAME SYSTEM

- First developments 1981-1983
- Introduced into the Internet between 1983 and 1988
- Replacement for the static `hosts.txt` file
- Continuously updated and expanded
- Scales with the growth of the Internet
- Minimal built-in security

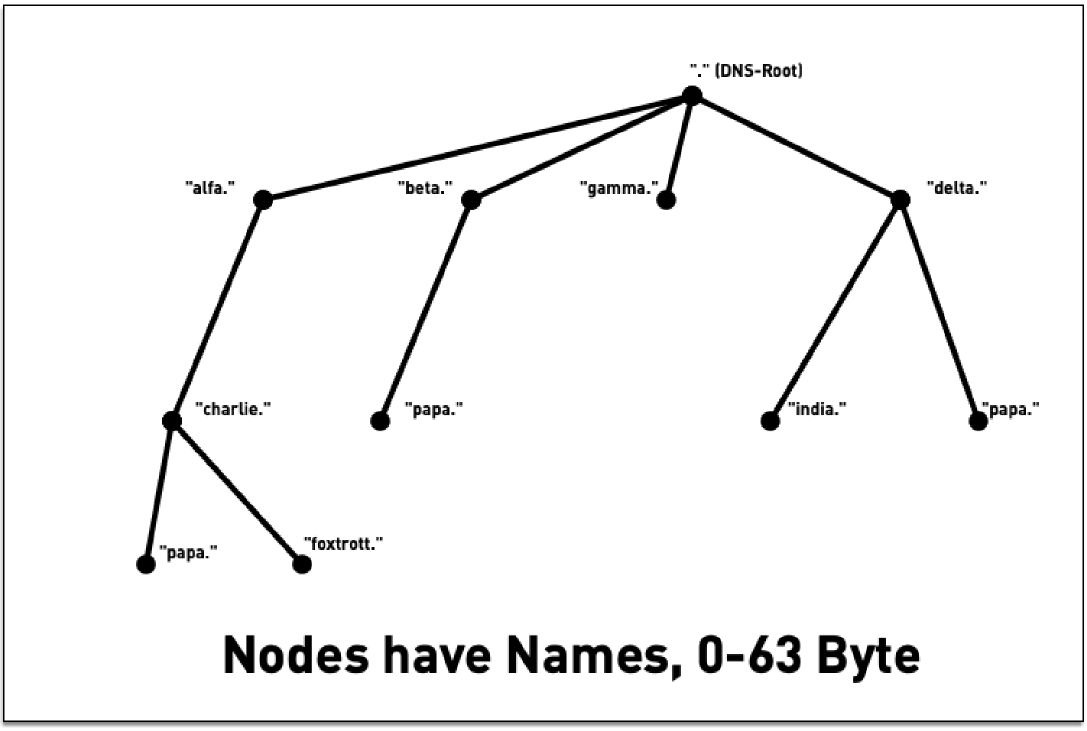
3.3. DNS NAMESPACE



3.4. DNS NAMESPACE

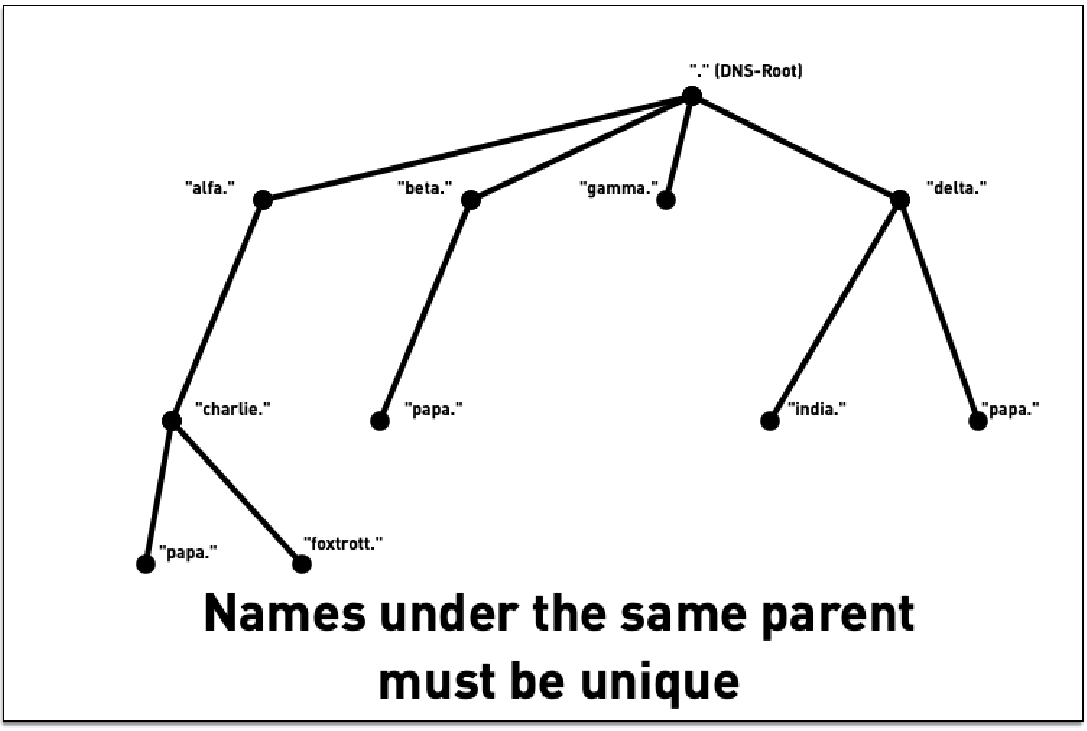


3.5. DNS NAMESPACE - "LABEL"



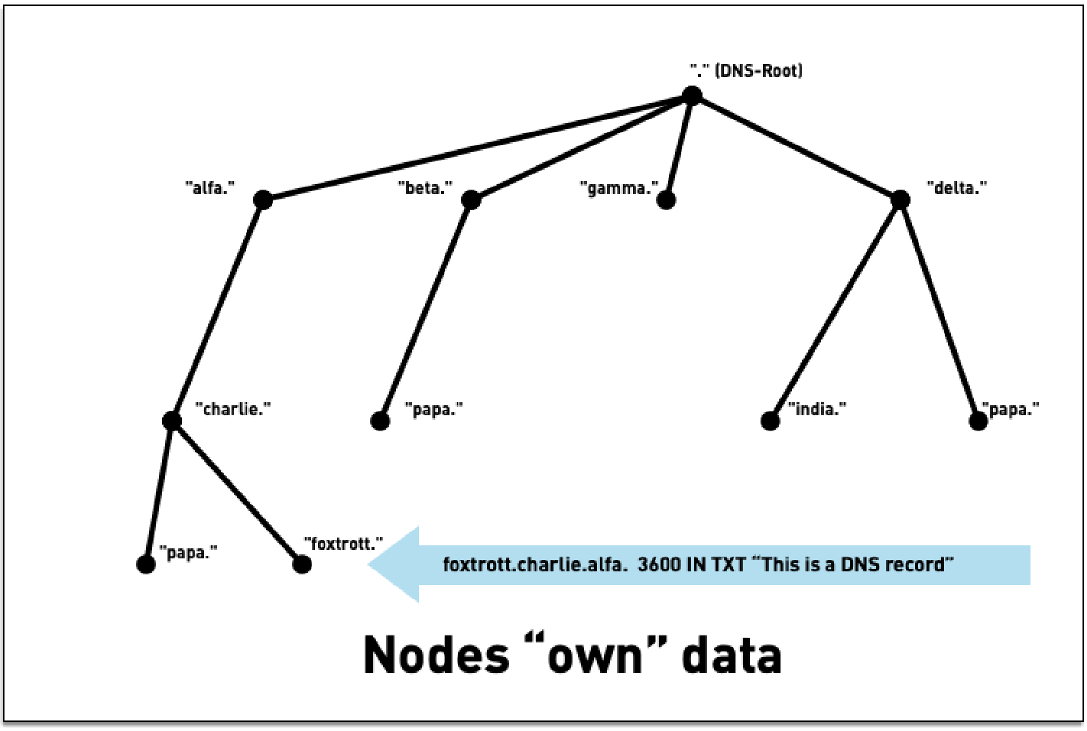
Nodes have Names, 0-63 Byte

3.6. DNS NAMESPACE - "LABEL"



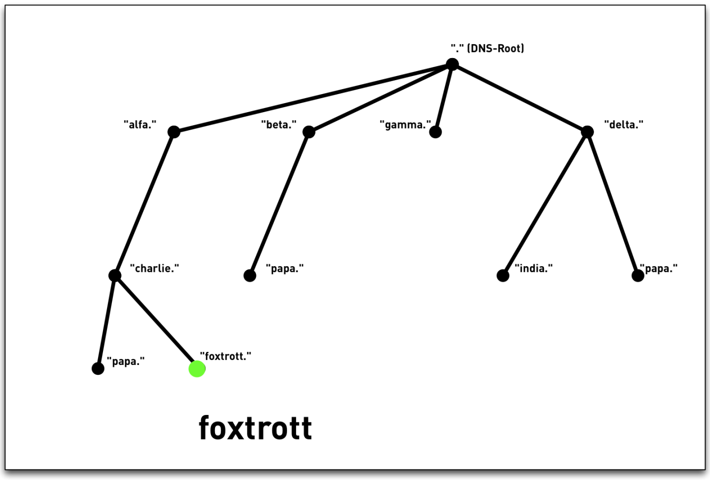
Names under the same parent must be unique

3.7. DNS NAMESPACE - "LABEL"

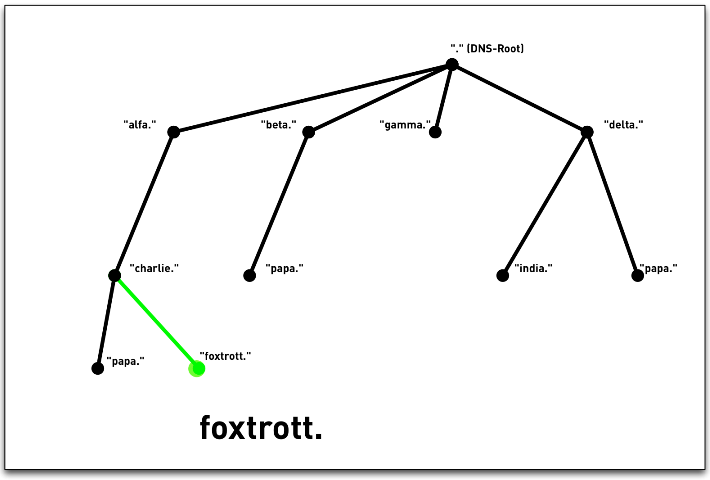


Nodes "own" data

3.8. DNS NAMESPACE



foxtrott



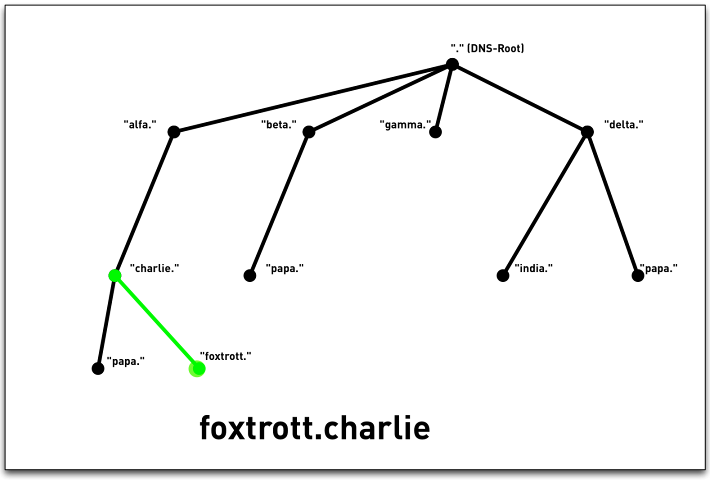
"." (DNS-Root)

"alfa." "beta." "gamma." "delta."

"charlie." "papa." "india." "papa."

"papa." "foxtrott."

foxtrott.



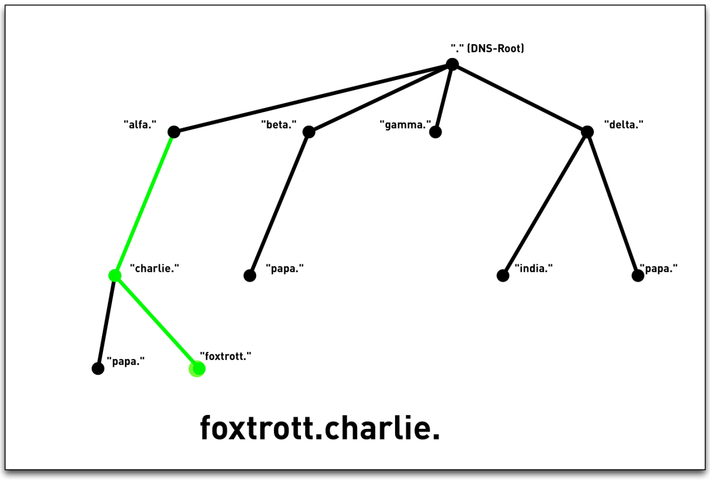
"." (DNS-Root)

"alfa." "beta." "gamma." "delta."

"charlie." "papa." "india." "papa."

"papa." "foxtrott."

foxtrott.charlie



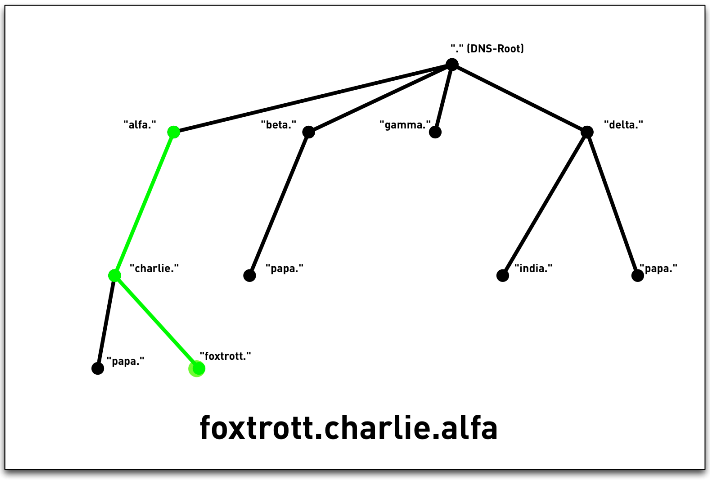
"." (DNS-Root)

"alfa." "beta." "gamma." "delta."

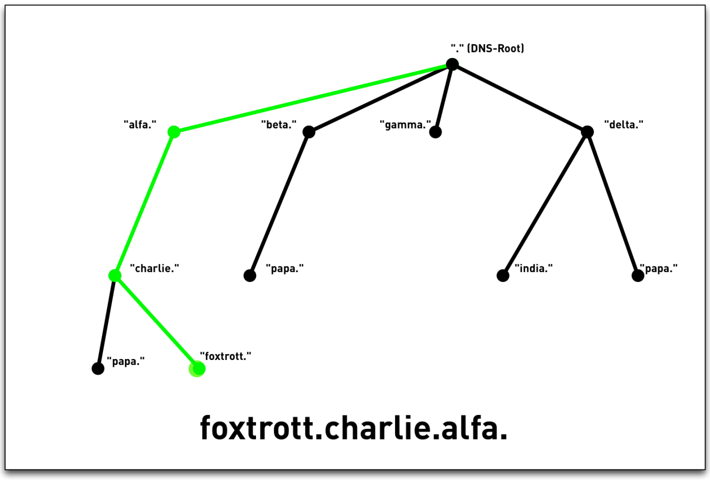
"charlie." "papa." "india." "papa."

"papa." "foxtrott."

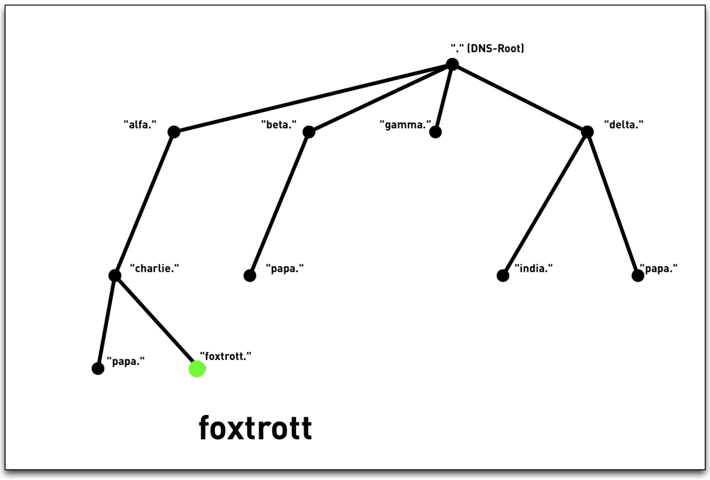
foxtrott.charlie.



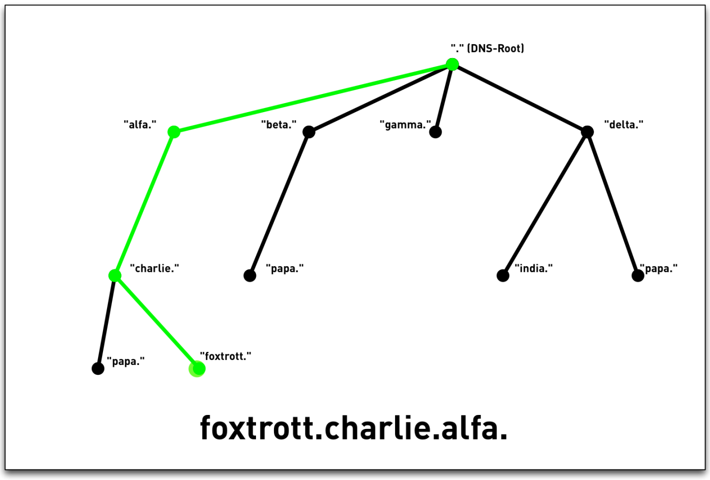
foxtrott.charlie.alfa



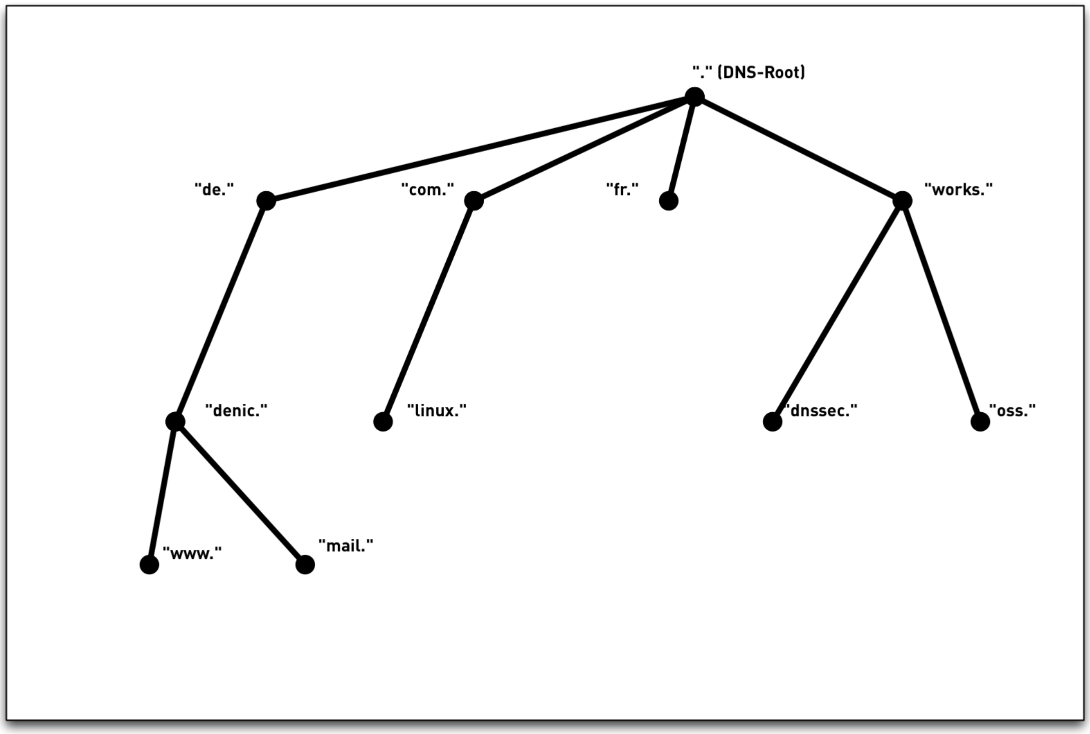
foxtrott.charlie.alfa.



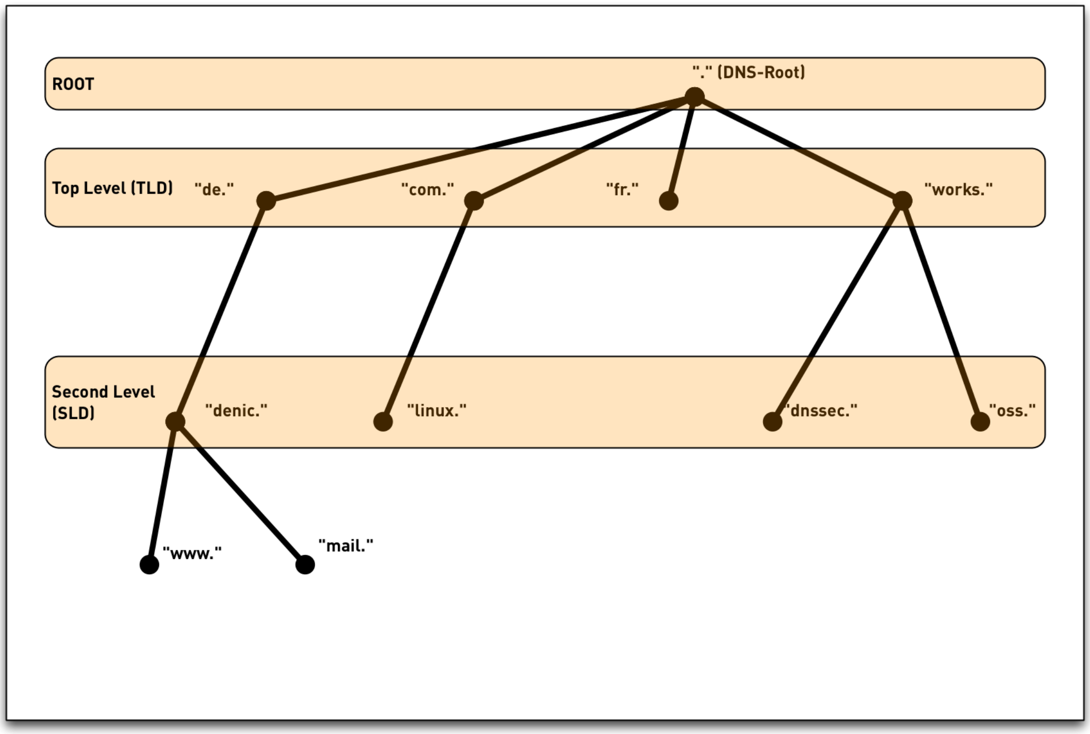
foxtrott



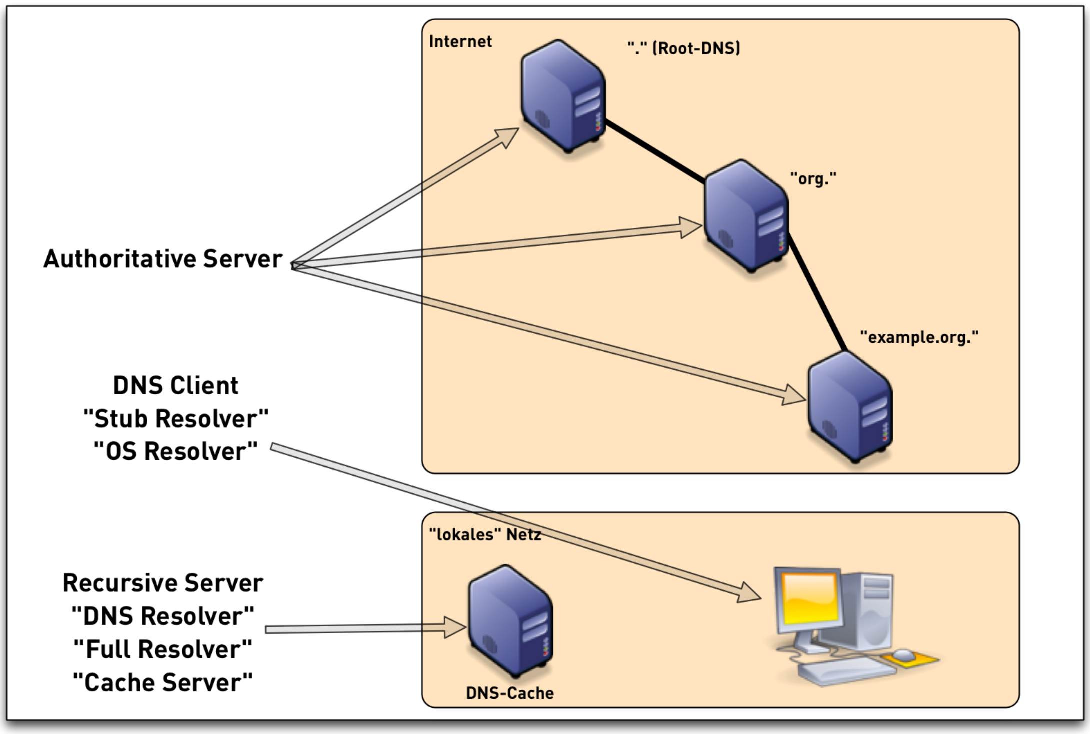
3.9. INTERNET NAMESPACE



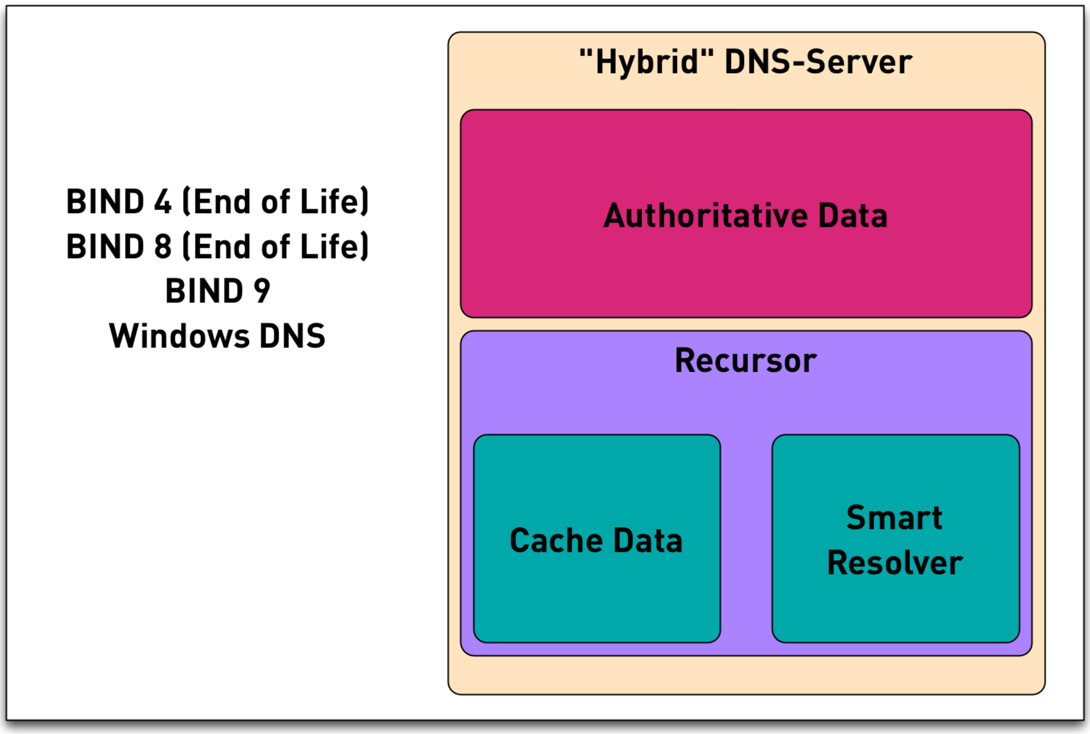
3.10. INTERNET NAMESPACE



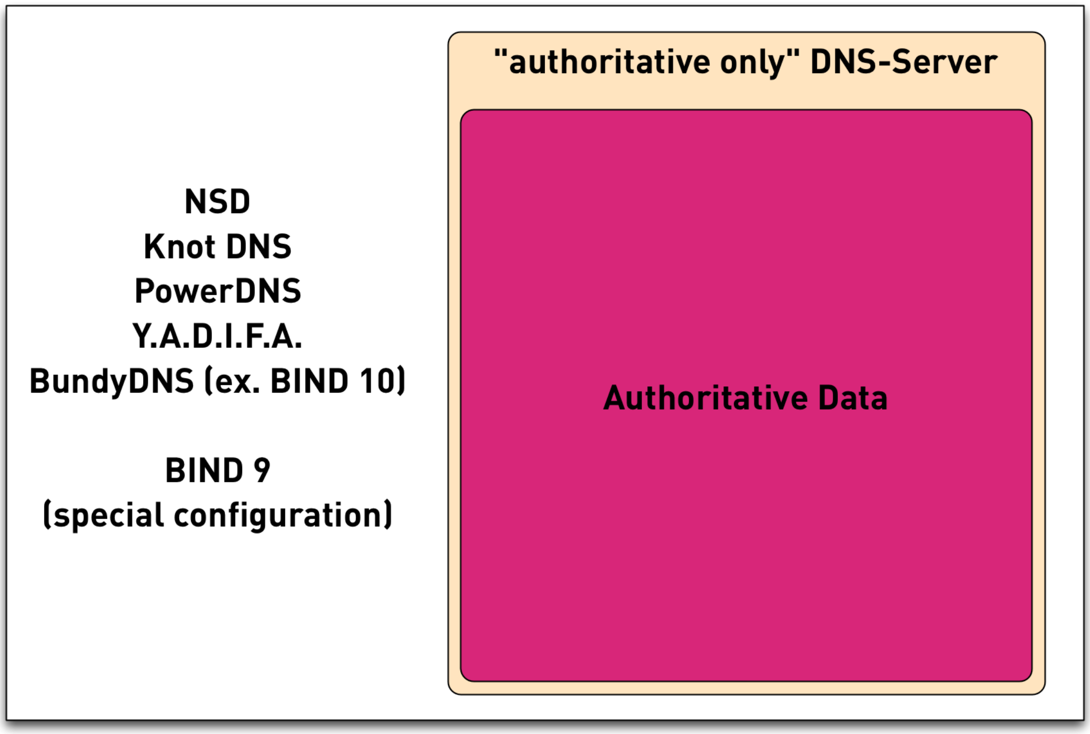
3.11. DNS COMPONENTS



3.12. DNS COMPONENTS - HYBRID-DNS



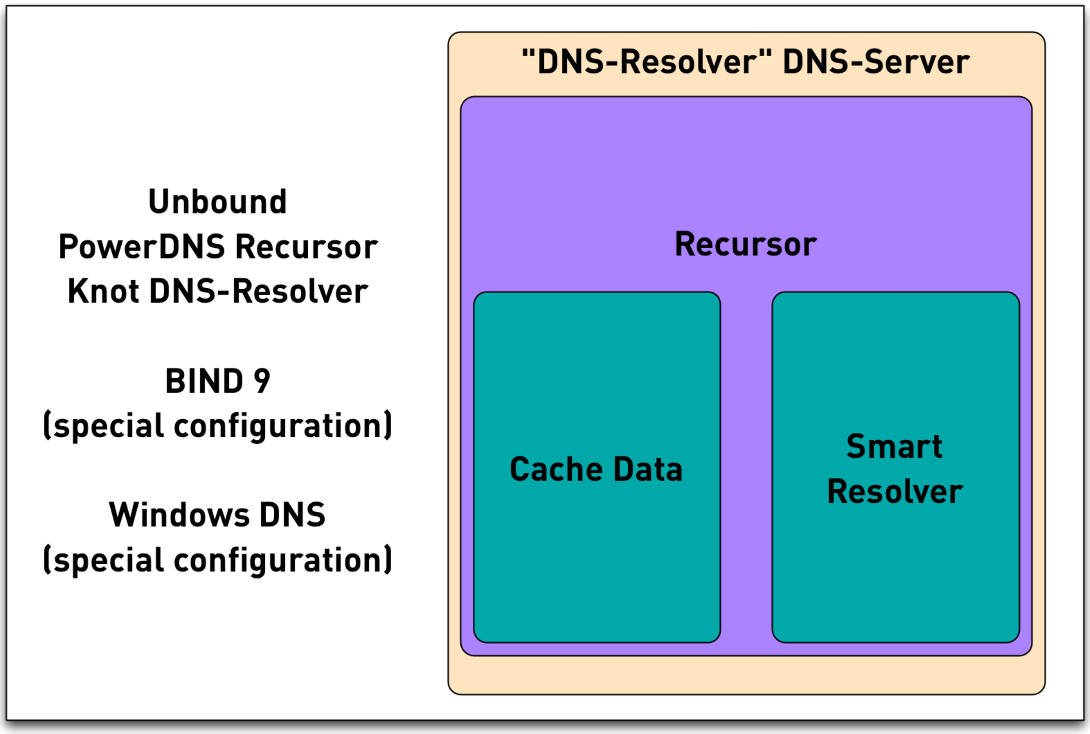
3.13. DNS COMPONENTS - AUTHORITATIVE ONLY



NSD
Knot DNS
PowerDNS
Y.A.D.I.F.A.
BundyDNS (ex. BIND 10)

BIND 9
(special configuration)

3.14. DNS COMPONENTS - DNS RESOLVER



Unbound
PowerDNS Recursor
Knot DNS-Resolver

BIND 9
(special configuration)

Windows DNS
(special configuration)

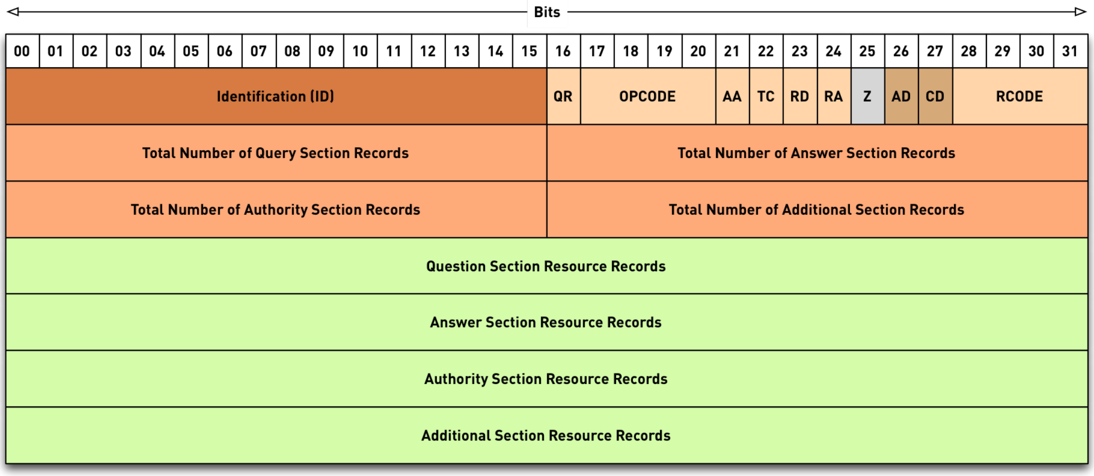
"DNS-Resolver" DNS-Server

Recursor

Cache Data

Smart Resolver

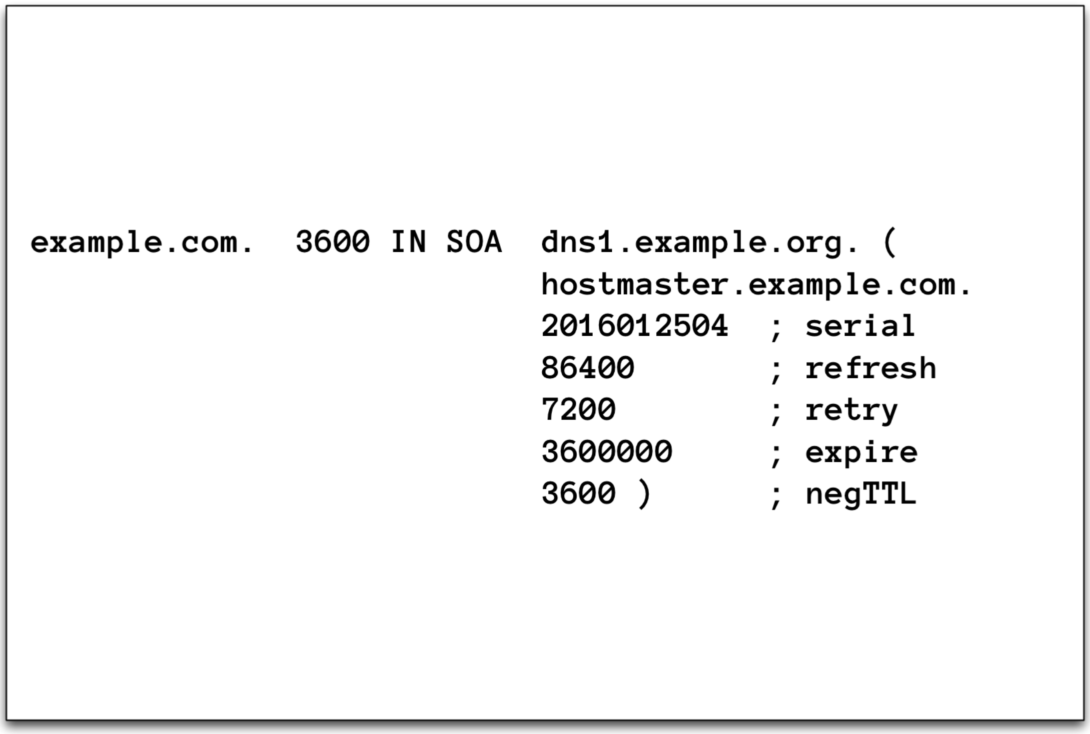
3.15. DNS MESSAGE FORMAT



<table>
<tr><th colspan="32">Bits</th></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Identification (ID)</td><td>QR</td><td colspan="4">OPCODE</td><td>AA</td><td>TC</td><td>RD</td><td>RA</td><td>Z</td><td>AD</td><td>CD</td><td colspan="4">RCODE</td></tr>
<tr><td colspan="16">Total Number of Query Section Records</td><td colspan="16">Total Number of Answer Section Records</td></tr>
<tr><td colspan="16">Total Number of Authority Section Records</td><td colspan="16">Total Number of Additional Section Records</td></tr>
<tr><td colspan="32">Question Section Resource Records</td></tr>
<tr><td colspan="32">Answer Section Resource Records</td></tr>
<tr><td colspan="32">Authority Section Resource Records</td></tr>
<tr><td colspan="32">Additional Section Resource Records</td></tr>
</table>

3.16. DNS RESOURCE RECORDS

3.16.1. SOA Record

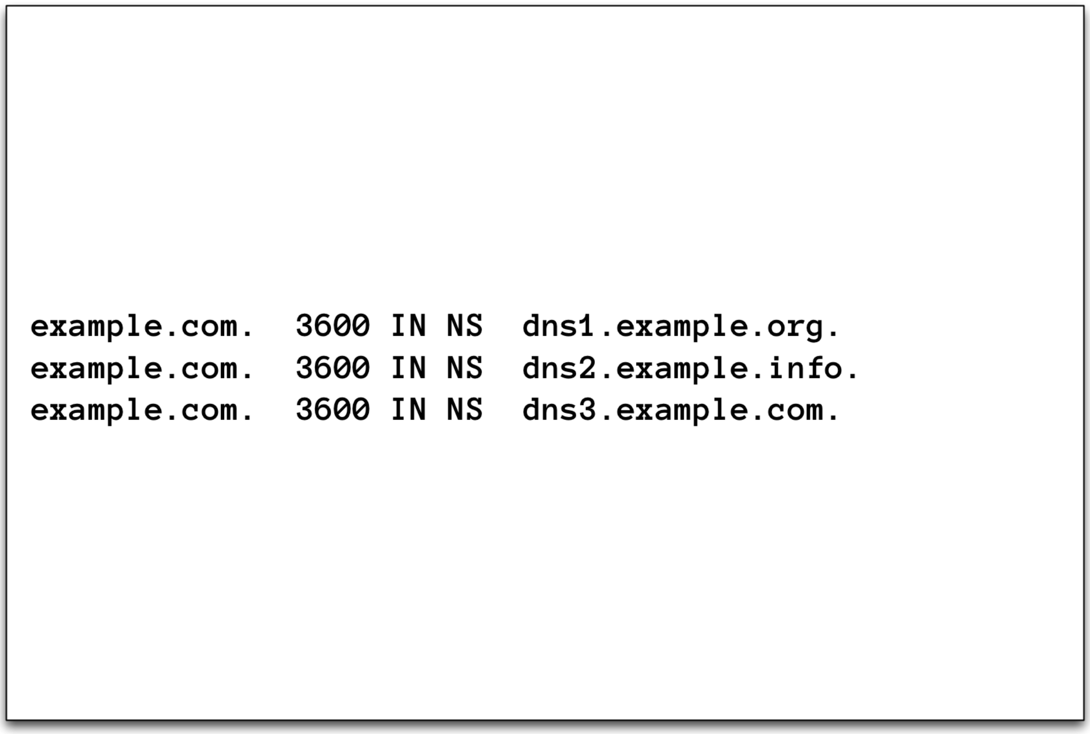


```
example.com.  3600 IN SOA  dns1.example.org. (
                           hostmaster.example.com.
                           2016012504  ; serial
                           86400       ; refresh
                           7200        ; retry
                           3600000     ; expire
                           3600 )      ; negTTL
```

3.16.2. Negative Caching

- The last value of the SOA record (together with the TTL-value of the SOA record, the lowest value will be used) controls how long negative DNS responses are stored in a DNS cache
- Negative responses are NXDOMAIN and NXRRSET/NODATA
- Error-Responses (FORMERR, REFUSED …) will not be cached
- SERVFAIL is cached for `servfail-ttl` seconds (default 1, maximum is capped to 30)

3.16.3. NS record

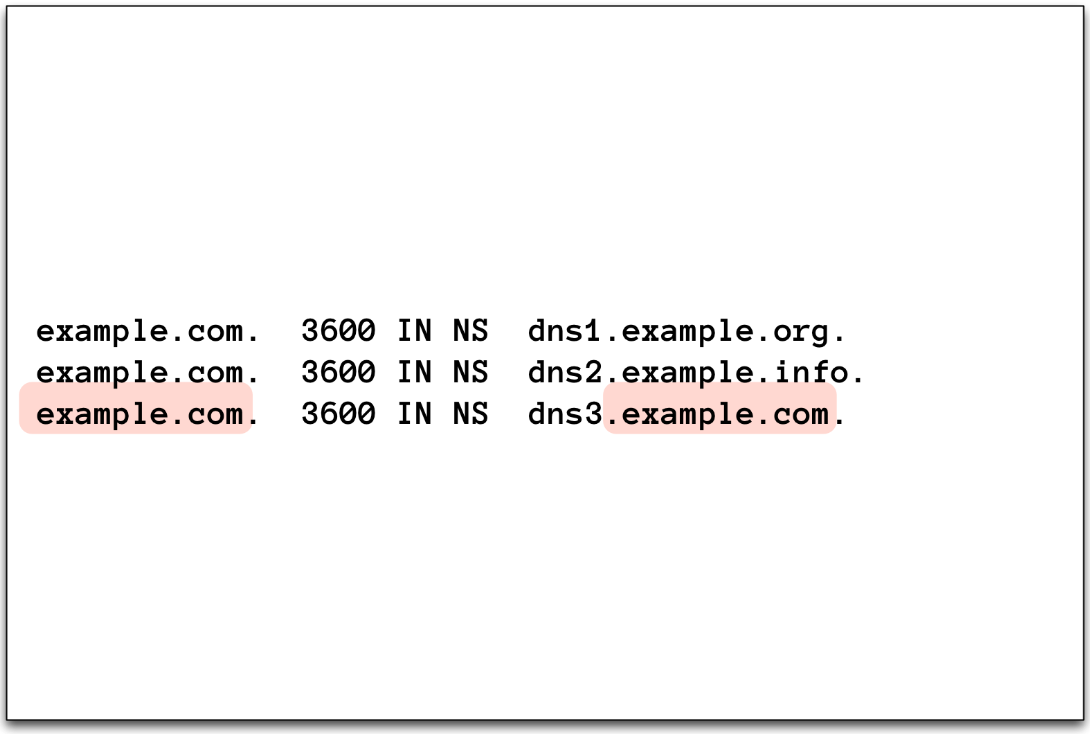


```
example.com.  3600 IN NS  dns1.example.org.
example.com.  3600 IN NS  dns2.example.info.
example.com.  3600 IN NS  dns3.example.com.
```

3.16.4. NS record

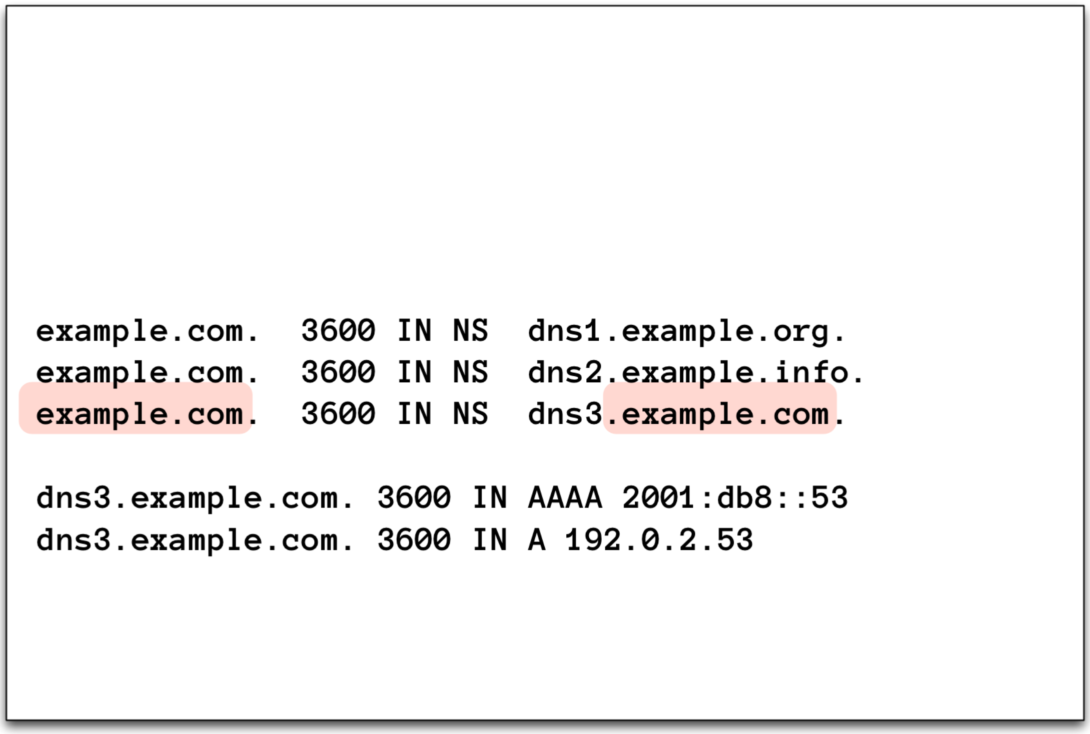
- NS records create the DNS delegation
- The NS record in the parent zone (delegation) SHOULD always match the NS records in the (delegated) zone (see *When parents and children disagree*)
- Is the domain name of the authoritative DNS servers for a zone inside the delegated domain, the parent zone must be augmented with "Glue" records (A/AAAA records of the authoritative DNS Servers) to make the delegation work

3.16.5. Glue (1)



```
example.com.  3600 IN NS  dns1.example.org.
example.com.  3600 IN NS  dns2.example.info.
example.com.  3600 IN NS  dns3.example.com.
```

3.16.6. Glue (2)



```
example.com.  3600 IN NS  dns1.example.org.
example.com.  3600 IN NS  dns2.example.info.
example.com.  3600 IN NS  dns3.example.com.

dns3.example.com. 3600 IN AAAA 2001:db8::53
dns3.example.com. 3600 IN A 192.0.2.53
```

3.17. EXERCISE:

- Login to the DNS Resolver machine `dnsr.zXX.dane.onl` (Username `user`, password `dnssec-2023`)
- Become user `root` with `sudo bash`
- Install the BIND 9 DNS Server and the DNS lookup tools

```
apt install bind9 dnsutils
```

- On Debian Linux, the BIND 9 DNS Server will automatically start. The BIND 9 service name is `named` (Name Daemon)

```
systemctl status named
```

- By default, BIND 9 is configured as an DNSSEC validating full-resolver
- Try to answer the questions below by sending queries to the local BIND 9 DNS Resolver:

  Is the domain `dnssec.works` DNSSEC secured?

```
  dig @localhost dnssec.works SOA
```

- What is the negative Time To Live (TTL) value for negative answers from the zone `dnssec.works`?

```
  dig @localhost xyz.dnssec.works
```

- What is the *effective* negative TTL for negative answers from the zone `dnssec.works`?

- Does the domain name `www.dnssec.works` have an IPv4 and/or an IPv6 address?

```
    dig @localhost www.dnssec.works A
    dig @localhost www.dnssec.works AAAA
```

- Does the domain name `dnssec.works` have a `HTTPS` record?

```
    dig @localhost dnssec.works HTTPS
```

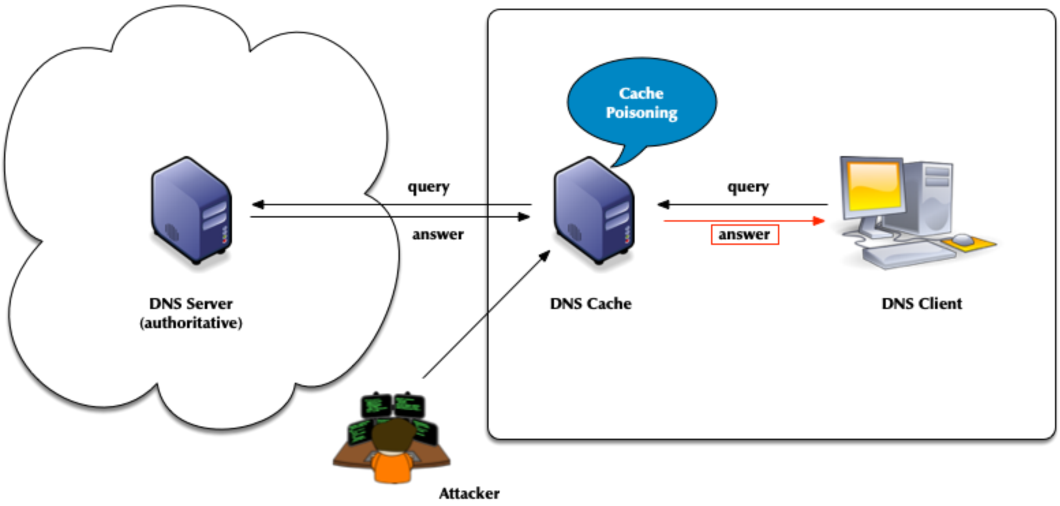
- Does the domain name `dnssec.works` does have a `CAA` record?

```
    dig @localhost dnssec.works CAA
```

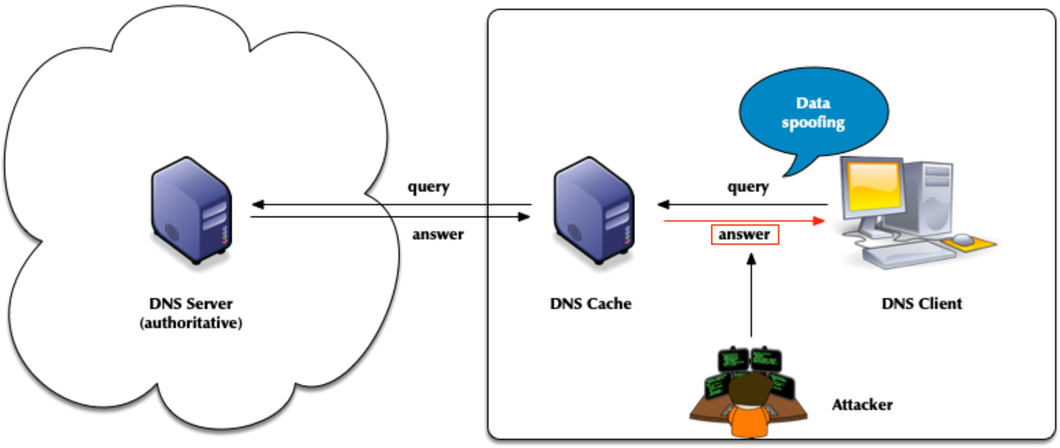
CHAPTER 4. AN INTRO TO DNSSEC

4.1. DNS SECURITY (OR LACK OF)

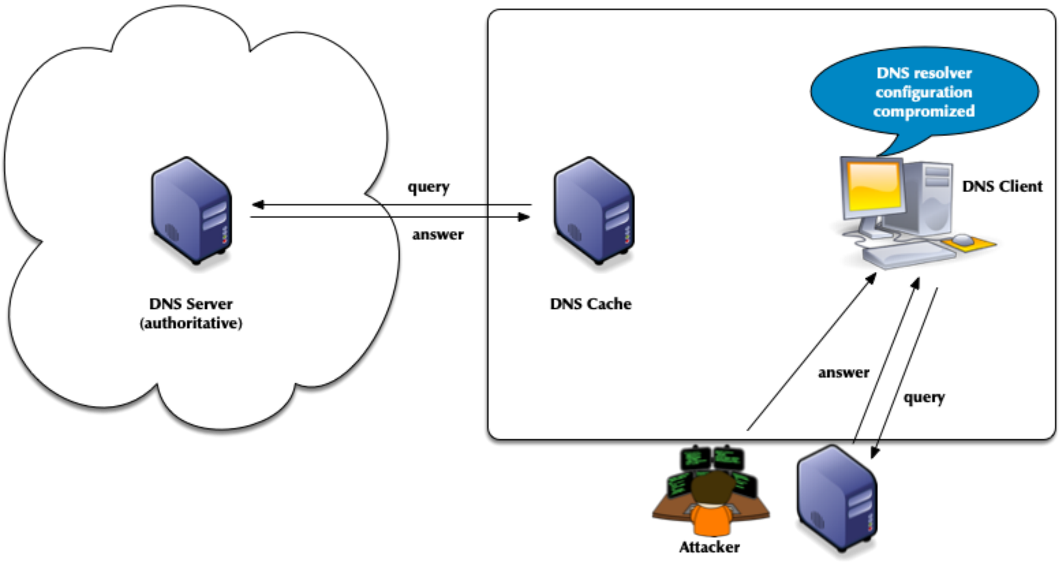
- The classic DNS from 1983 has not been designed with security in mind
- Attack vector: DNS cache poisoning



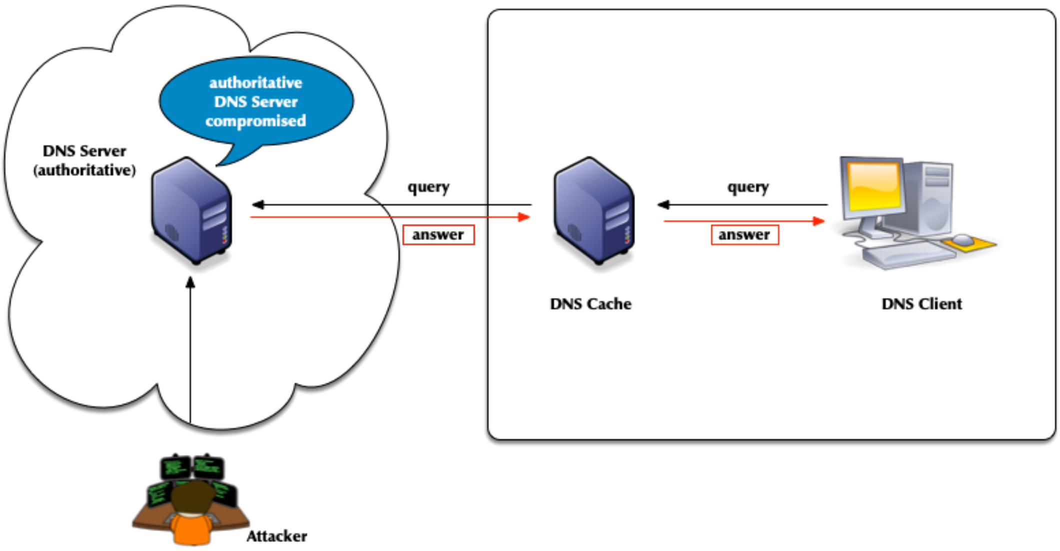
- Attack vector: Men-in-the-Middle data spoofing



- Attack vector: changes to the client DNS resolver configuration


DNS resolver configuration compromized
DNS Client
query
answer
DNS Server (authoritative)
DNS Cache
answer
query
Attacker

- Attacks of authoritative DNS servers


authoritative DNS Server compromised
DNS Server (authoritative)
query
answer
query
answer
DNS Cache
DNS Client
Attacker

- DNSSEC can help

WITH DNSSEC
THIS WOULD
NOT HAVE
HAPPENED

CHAPTER 5. CRYPTOGRAPHY IN DNS (SHORT)

5.1. CRYPTOGRAPHY FOR DNS ADMINS

- Cryptography has four purposes:

  Confidentiality – keeping data secret

  Integrity – Is it "as sent"?

  Authentication – Did it come from the right place?

  Non-Repudiation – Don't tell me you didn't say that.

- DNSSEC implements: Authentication & Integrity
- The IETF has added confidentiality since 2016:

  RFC 7858 "Specification for DNS over Transport Layer Security (TLS)" (DNS-over-TLS) https://tools.ietf.org/html/rfc7858 [https://tools.ietf.org/html/rfc7858]

  RFC 8484 "DNS Queries over HTTPS (DoH)" (DNS-over-HTTPS) https://tools.ietf.org/html/rfc8484 [https://tools.ietf.org/html/rfc8484]

- DNS uses different authenticity and integrity techniques depending on the application.
- Symmetric Cryptography

  Message Digests (aka: hashes, hash values, fingerprints)

  Message Authentication Codes

- Asymmetric Cryptography

  Digital Signatures

5.2. SYMMETRIC CRYPTOGRAPHY

- Symmetric cryptography provides both integrity and authenticity.
- A single key is stored and used by both (all) parties.

  The key encrypts and decrypts.

  The key is a shared secret.

  The system is known as pre-shared key (PSK).

  Securely getting the key to all parties can be a challenge.

- Symmetric cryptography requires mutual trust.

  "My security is good, but what about the other person's?"

  If all sides are administered by one party, trust is a non-issue.

  For DNS, *Pre-Shared-Keys* (PSK) work well:

    between primary and secondary servers (TSIG).

        for dynamic DNS updates (TSIG).

        for controlling BIND 9 with `rndc` (TSIG).

    For DNS, PSKs do not work well:

        between DNS Resolver servers and authoritative DNS servers (DNSSEC).

        between stub resolver & DNS Resolver servers (DNSSEC amongst other).

5.2.1. Confidentiality: Not the Goal

- A Pre-Shared Key (PSK) could be used to encrypt a message.
- If it decrypts, confidentially, integrity, and authenticity are assured.
- However, confidentiality was not a design goal (of DNSSEC).
- Encrypting everything is computationally expensive.
- The alternative is more complicated but less expensive.

5.2.2. Message Digest

- Creating a hash of the message is a light-weight alternative to encrypting everything.

  A hash is also known as a hash value, a message digest (MD) and a fingerprint.
- The sender runs a cryptographic hashing algorithm on the message to produce a fixed-length hash.

  The message and hash are sent over an insecure path.
- As the sender did, the receiver hashes the message.
- If the hashes match, the message was not modified. **Message integrity is proven**.
- However, the receiver does not know if the message and hash were both replaced: **Message authenticity is unknown**.
- **Confidentiality is not provided.**

5.2.3. Message Authentication Codes

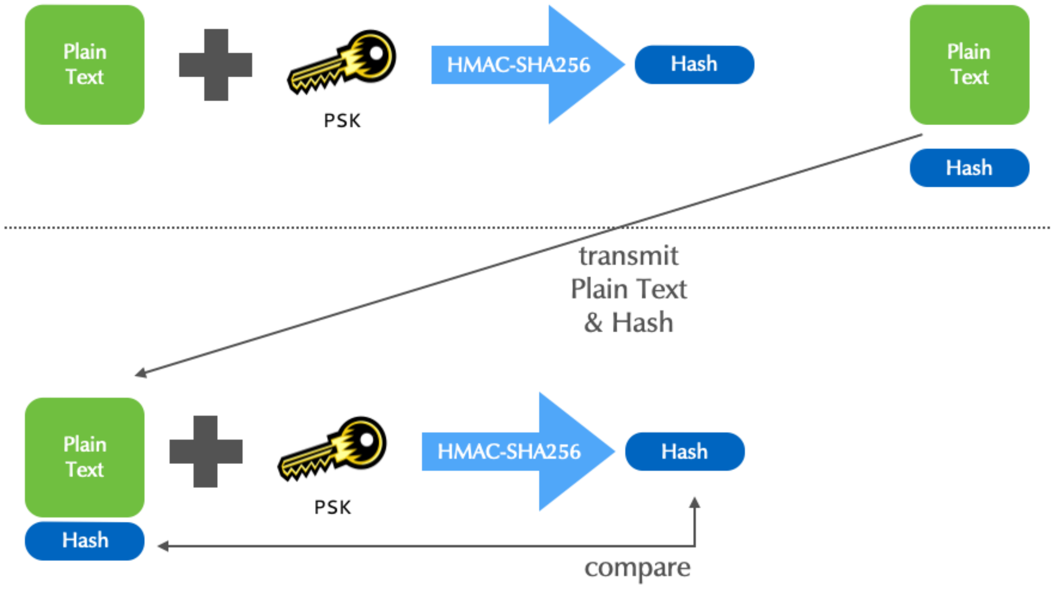
- Hashing, with a symmetric key added to the input, efficiently provides **integrity** and **authenticity**.

  There is no confidentiality.

  A MAC is a fingerprint (MD, hash) created with the message and a PSK as input.

  The MAC described here, is a keyed HMAC (Hash-based message authentication code).

    It is used by DNS.



- Although HMACs efficiently assure both the sender's authenticity, and the message's integrity, not all applications can use (pre)shared keys.

  "Am I really seeing the website `mybank.example`?"

  "Is the email I'm reading really from Edward Snowden?"

  "Is this RRSet actually for `theguardian.com`?"

5.3. ASYMMETRIC CRYPTOGRAPHY

- In DNS, asymmetric cryptography is used for DNSSEC.
- This asymmetric cryptography section is a short overview.
- Asymmetric cryptography uses two keys, a pair.

  Data encrypted with one key can only be decrypted by the other.

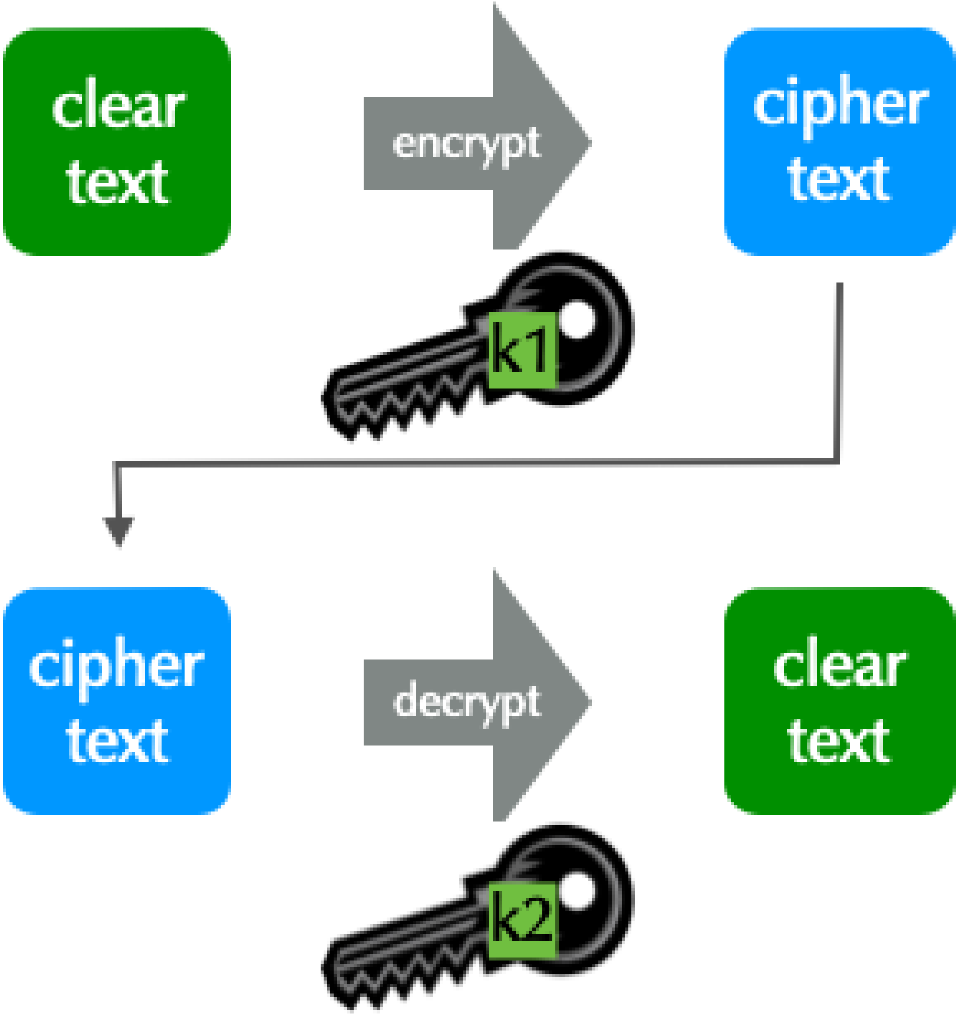
  A key cannot decrypt what it encrypted!

  One key of a pair is declared as public, the other as private.

  The private key is highly sensitive, never shared, and must be well protected.

  The public key is made widely available without concern for who knows it.

  The technique is known as public key encryption.



5.3.1. Two Applications for public key (PK) Encryption

- PK encryption is used for privacy, to assure only the intended receiver can read a message.

  Data integrity is also assured.

  Used in DNS-over-TLS and DNS-over-HTTPS

- PK encryption is used for authenticity, assuring the recipient, that the message came from a specific sender.

  This is known as signing a message.

  Data integrity is also assured.

  Used in DNSSEC, as well in DNS-over-TLS and DNS-over-HTTPS

5.3.2. PK Application 2: Encryption for Authenticity

- To assure authenticity, a sender encrypts a message with her private key.

  Anyone decrypting the message with the public key is assured it came from the holder of the private key.
- The message is encrypted, but there is no privacy.

5.3.3. PK Application 2: Encryption for Authenticity: Digital Signatures

- For efficiency, the message itself is not encrypted.

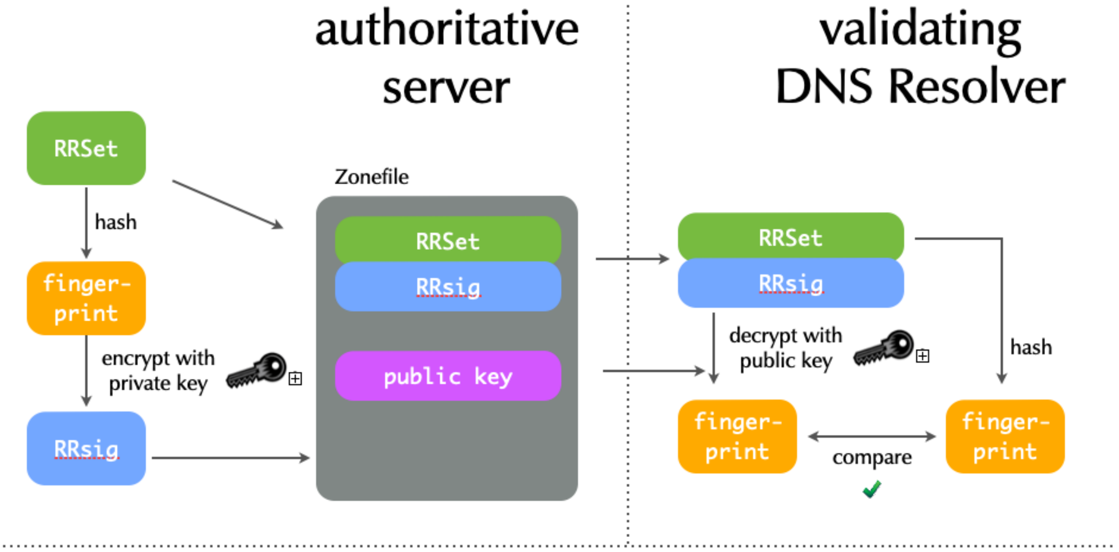
  The message is first hashed to create a fingerprint.

  The fingerprint is encrypted.

  The signed fingerprint is a digital signature (aka encrypted fingerprint and encrypted hash).

A fingerprint is also known as a hash, digest, or message digest. A signature is not the same thing. However, the terms are often used interchangeably.

5.3.4. Digital Signatures in DNSSEC



5.4. DNSSEC

- The DNS Security Extensions, described in RFC 2065 (old DNSSEC), allow a zone administrator to digitally sign zone data
- The base DNSSEC RFCs:

  RFC 4033 to 4035 - Updates DNSSEC to DNSSECbis, published March 2005:

  RFC 4033 - DNS Security Introduction and Requirements

  - RFC 4034 - Resource Records for the DNS Security Extensions
  - RFC 4035 - Protocol Modifications for the DNS Security Extensions

- The DNS security extension DNSSEC secures DNS data by augmenting the data with cryptographic signatures
  - The owner (administrator) creates a pair of private and secret keys for each DNS zone (asymmetric crypto)
  - The owner/administrator signs all DNS data with the private/secret key
  - The recipient of the data (DNS resolver or client operating system or application) will verify (validate) the data
    - That the data has not been changed on the server nor during transit
    - That the data comes from the owner (the owner of the private key)
  - DNSSEC signs data to guarantee authenticity and integrity.
    - It assures a client that a RRSet is from the proper authoritative sever and has not changed.
  - DNSSEC does not encrypt data to provide privacy.
    - Anyone can find out the RRSets you request.
- DNS Servers that support DNSSEC
  - BIND 9.6 and up: Authoritative server and validating resolver
  - NSD from NLnet Labs: Fast authoritative server
  - Unbound from NLnet Labs :Fast and secure validating resolver
  - Windows 2012 DNS Server: Authoritative server and validating resolver
  - PowerDNS authoritative: Authoritative DNS Server with (optional) SQL Database backend
  - PowerDNS Recursor: a resolver with support for DNSSEC validation
  - Knot-DNS: fast authoritative DNS Server from nic.cz
  - Knot-DNS Resolver: recursive server with DNSSEC from nic.cz

5.5. TSIG VS. DNSSEC

- TSIG uses a keyed cryptographic hash algorithm, which requires that both endpoints share a key
- DNSSEC uses public key cryptography, which doesn't require shared keys
  - Zone data is signed by the zone administrator
  - This provides an end-to-end integrity check between producer (DNS Administrator) and consumer (Resolver, Application)

5.6. DNSSEC:DESIGN CHOICES MADE

- DNSSEC signs RRSets, but alternatives were available to the designers:

An entire DNS message could be signed.

Just the answer section of a message could be signed.

Each RR in an answer could be signed.

- DNS messages and answer sections are dynamically generated when a query arrives. = The signatures would have to be dynamically generated.
- RRs and RRSets aren't modified when a query arrives at an authoritative server.

Signatures can be created when the zone is compiled.

- Signing each RRSet is most effectively and what is done.

CHAPTER 6. DNSSEC IN A NUTSHELL

- In DNSSEC, each zone has one or more key pairs.

    The private key of each pair

        Is stored securely (probably on a hidden primary)

        Is used to sign zone data

        Should never be stored in a RR

    The public key of each pair

        Is stored in the zone data, as a DNSKEY record

        Is used to verify zone data

- A private key signs the hashes of each RRSet in a zone.
- The public key is accessible through a standard RR.

    Recursive servers or clients query for the public key RR in order to decrypt a hash.

    The RR is known as the DNSKEY and covered below.

6.1. DNSSEC RECORDS

6.1.1. RRSIG

- The RRSIG records holds the cryptographic signature over the DNS data. The first field of the RRSIG holds the type this RRSIG is for. Together with the domain name of the RRSIG this data is important to match the signature to the data record.
- A zone's private key signs the RRSets in the zone.

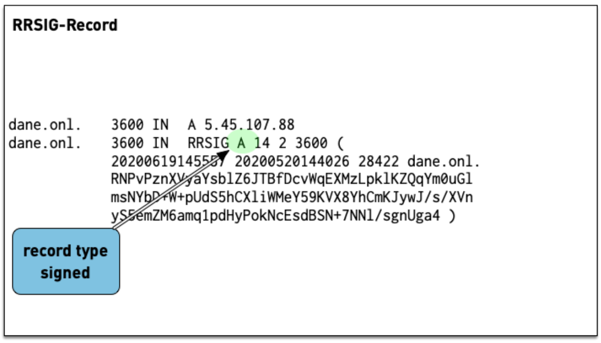
    The signatures are added to the zone as RRSIG RRs.

    If two key pairs are in use, each RRSet is signed twice, and there is double the number of signatures.

- Signatures have start and expiration times (typically a month or more apart).

    They must be replaced before expiring. BIND 9 can automate the signature updates.

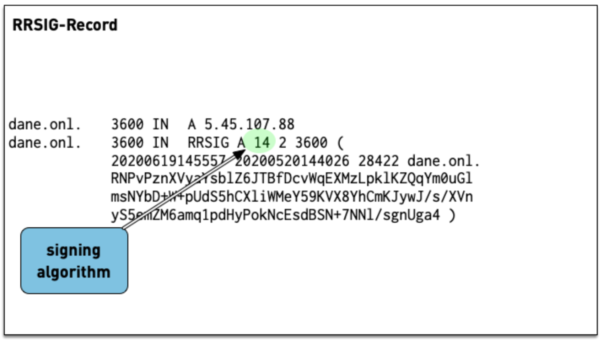
    Keys don't have expiration timers.



RRSIG-Record

```
dane.onl.    3600 IN   A 5.45.107.88
dane.onl.    3600 IN   RRSIG A 14 2 3600 (
             20200619145557 20200520144026 28422 dane.onl.
             RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
             msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
             yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

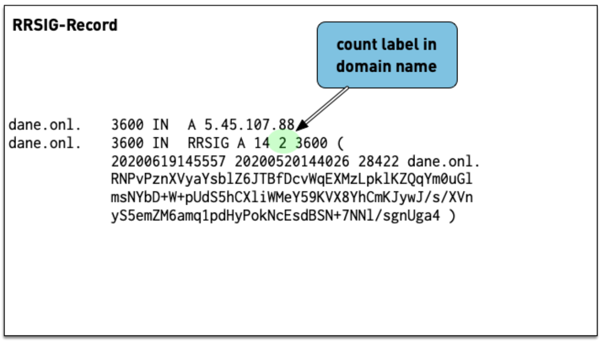
record type signed



RRSIG-Record

```
dane.onl.    3600 IN   A 5.45.107.88
dane.onl.    3600 IN   RRSIG A 14 2 3600 (
             20200619145557 20200520144026 28422 dane.onl.
             RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
             msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
             yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

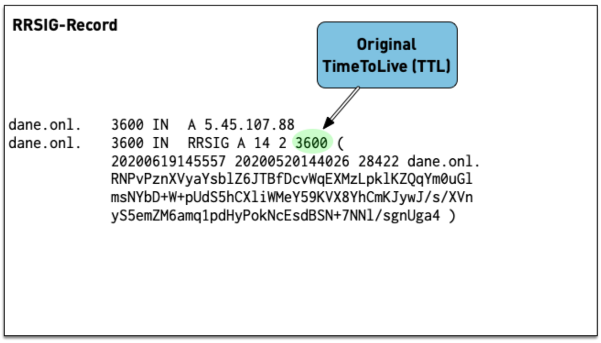
signing algorithm



RRSIG-Record

count label in domain name

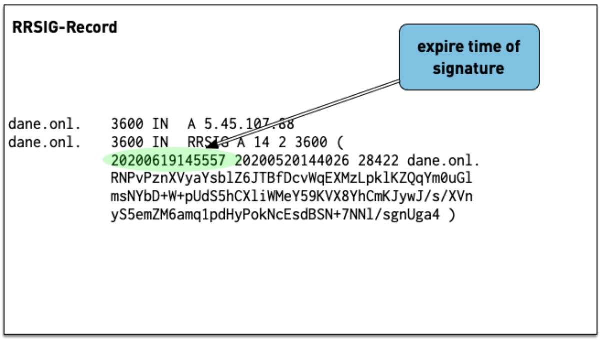
```
dane.onl.    3600 IN   A 5.45.107.88
dane.onl.    3600 IN   RRSIG A 14 2 3600 (
             20200619145557 20200520144026 28422 dane.onl.
             RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
             msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
             yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

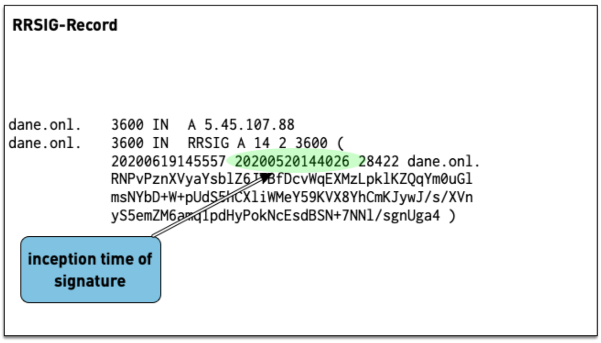
Original
TimeToLive (TTL)



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

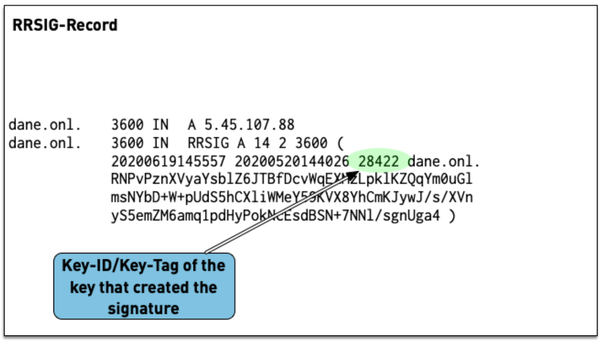
expire time of
signature



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

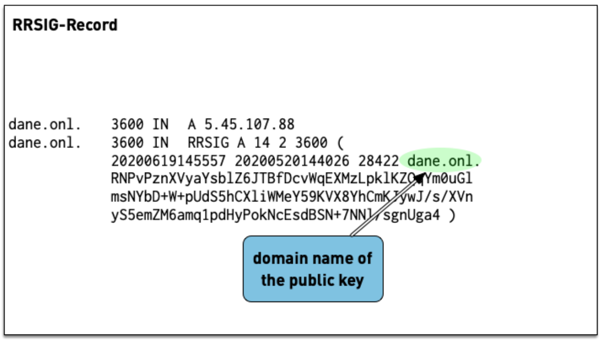
inception time of
signature



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

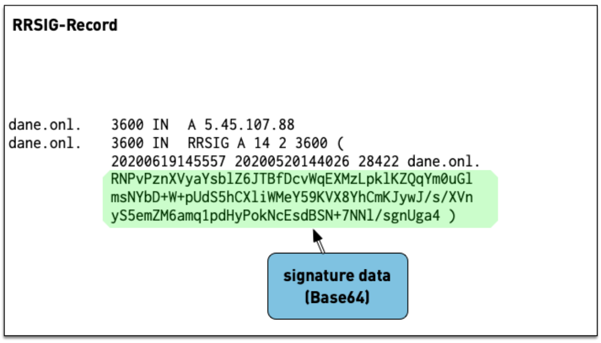
Key-ID/Key-Tag of the key that created the signature



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

domain name of the public key



RRSIG-Record

```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4 )
```

signature data (Base64)

- Work on the DNS resolver machine

- Ask for the DNSSEC signature of `dnssec.works NS`:

```
$ dig dnssec.works NS +dnssec +multi
```

- Answer these questions into the chat

  Which algorithm is this zone using? check the number against IANA Domain Name System Security (DNSSEC) Algorithm Numbers [https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml]

  When does the signature expire?

  When did the signature become valid?

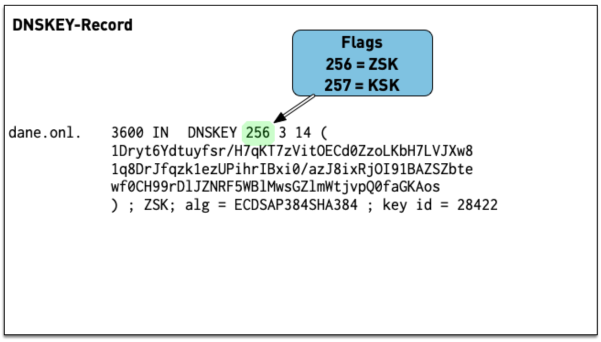
6.1.2. DNSKEY

- The public key of a DNSSEC key pair is stored in an DNSKEY RR.

  The private key is not publicly available or accessible through DNS.

- DNSKEY RRs are stored in the zone they can verify.

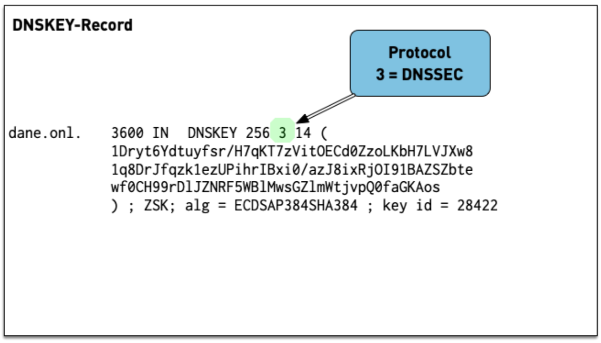
  This conveniently means the zone administrator can sign all the RRSets and create the DNSKEY RRSet.



DNSKEY-Record

```
dane.onl.    3600 IN  DNSKEY 256 3 14 (
             1Dryt6Ydtuyfsr/H7qKT7zVitOECd0ZzoLKbH7LVJXw8
             1q8DrJfqzk1ezUPihrIBxi0/azJ8ixRjOI91BAZSZbte
             wf0CH99rDlJZNRF5WBlMwsGZlmWtjvpQ0faGKAos
             ) ; ZSK; alg = ECDSAP384SHA384 ; key id = 28422
```

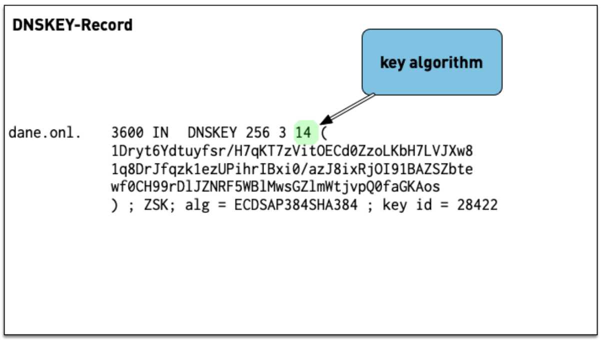
Flags
256 = ZSK
257 = KSK



DNSKEY-Record

Protocol
3 = DNSSEC

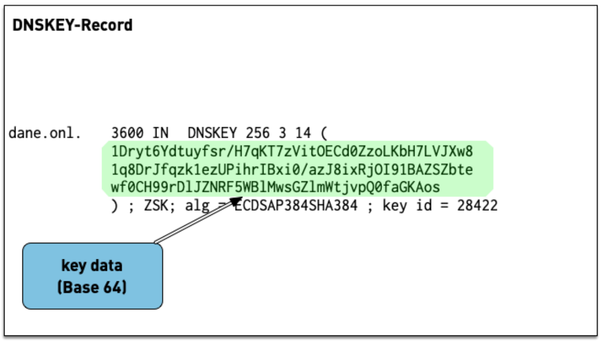
```
dane.onl.     3600 IN   DNSKEY 256 3 14 (
              1Dryt6Ydtuyfsr/H7qKT7zVitOECd0ZzoLKbH7LVJXw8
              1q8DrJfqzk1ezUPihrIBxi0/azJ8ixRjOI91BAZSZbte
              wf0CH99rDlJZNRF5WBlMwsGZlmWtjvpQ0faGKAos
              ) ; ZSK; alg = ECDSAP384SHA384 ; key id = 28422
```



DNSKEY-Record

key algorithm

```
dane.onl.     3600 IN   DNSKEY 256 3 14 (
              1Dryt6Ydtuyfsr/H7qKT7zVitOECd0ZzoLKbH7LVJXw8
              1q8DrJfqzk1ezUPihrIBxi0/azJ8ixRjOI91BAZSZbte
              wf0CH99rDlJZNRF5WBlMwsGZlmWtjvpQ0faGKAos
              ) ; ZSK; alg = ECDSAP384SHA384 ; key id = 28422
```



DNSKEY-Record

```
dane.onl.     3600 IN   DNSKEY 256 3 14 (
              1Dryt6Ydtuyfsr/H7qKT7zVitOECd0ZzoLKbH7LVJXw8
              1q8DrJfqzk1ezUPihrIBxi0/azJ8ixRjOI91BAZSZbte
              wf0CH99rDlJZNRF5WBlMwsGZlmWtjvpQ0faGKAos
              ) ; ZSK; alg = ECDSAP384SHA384 ; key id = 28422
```

key data
(Base 64)

- Work on the DNS resolver machine

- Ask for the DNSSEC keys of `dnsworkshop.cz`:

```
$ dig dnsworkshop.cz DNSKEY +dnssec +multi
```

- Answer these questions into the chat

    Which algorithm is this zone using?

    Compare the size of keys and signature with the zone `dnssec.works`

    Write in the chat the sizes of both DNS answer messages as reported by `dig`

6.2. THE CHAIN OF TRUST IN DNS

- If an attacker breaks into the server with the master zone file, she can change any data, including the DNSKEY RRSet.
- The DNSKEY RRSet is signed by the zone's private key!

    That signature, even when validated, proves nothing.

    It's a circular problem.

- External validation is needed, but we can't look beyond DNS.
- DNSSEC creates a chain of trust between the parent zone and the child zone image::./img/dnssec-chain-of-trust.png[scaledwidth=100%]
- Applications and DNS resolver can follow the chain of trust to a configured trust anchor to validate the DNS data

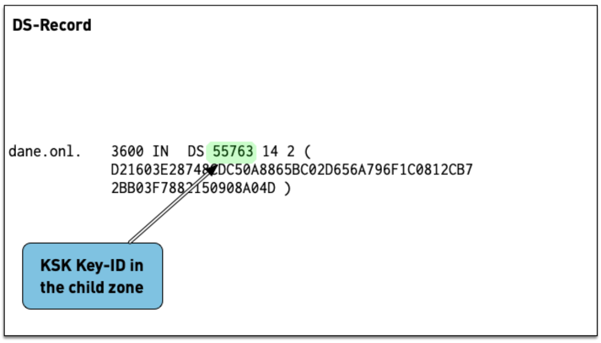
6.2.1. DS Record

- The Delegation Signer (DS) records holds the hash of the Key-Signing-Key of a DNSSEC signed child zone.

    It is used to verify that the KSK has not been replaced without permission

    The DS record is usually submitted to the parent zone operator via a web-based or API system implemented by the domain reseller (registrar)

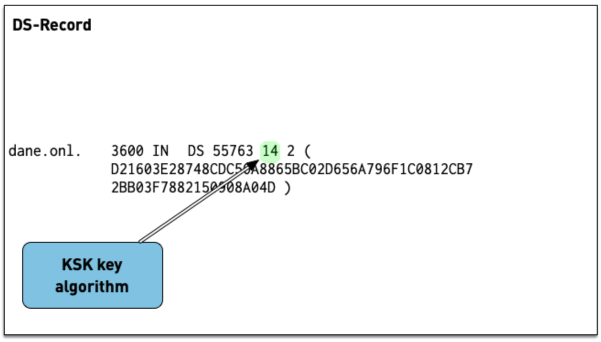
        This interface can become a target for attacker that want to change data in a DNSSEC signed zone and should be protected (two factor signon, registry lock etc).



DS-Record

```
dane.onl.     3600 IN   DS 55763 14 2 (
                   D21603E28748CDC50A8865BC02D656A796F1C0812CB7
                   2BB03F7882150908A04D )
```

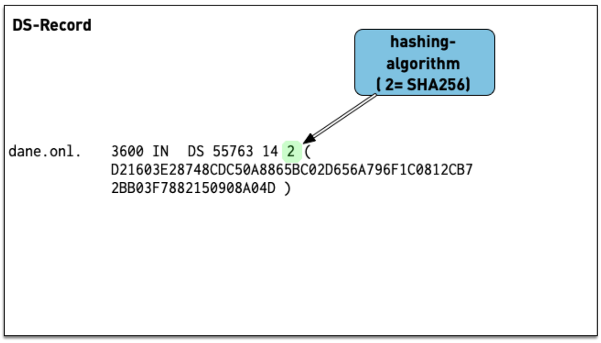
KSK Key-ID in the child zone



DS-Record

```
dane.onl.     3600 IN   DS 55763 14 2 (
                   D21603E28748CDC50A8865BC02D656A796F1C0812CB7
                   2BB03F7882150908A04D )
```

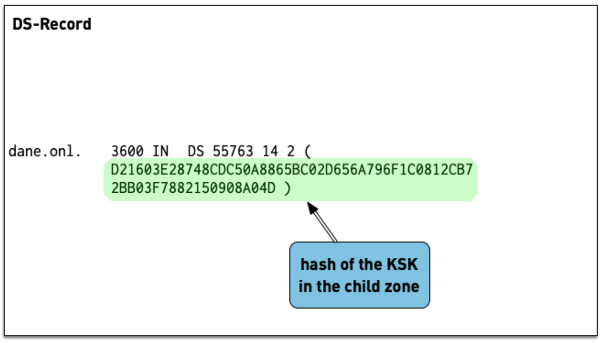
KSK key algorithm



DS-Record

hashing-algorithm (2= SHA256)

```
dane.onl.     3600 IN   DS 55763 14 2 (
                   D21603E28748CDC50A8865BC02D656A796F1C0812CB7
                   2BB03F7882150908A04D )
```



DS-Record

```
dane.onl.    3600 IN  DS 55763 14 2 (
             D21603E28748CDC50A8865BC02D656A796F1C0812CB7
             2BB03F7882150908A04D )
```

- Work on the DNS resolver machine
- Ask for the DNSSEC delegation signer of `isc.org`:

```
$ dig isc.org DS +dnssec +multi
```

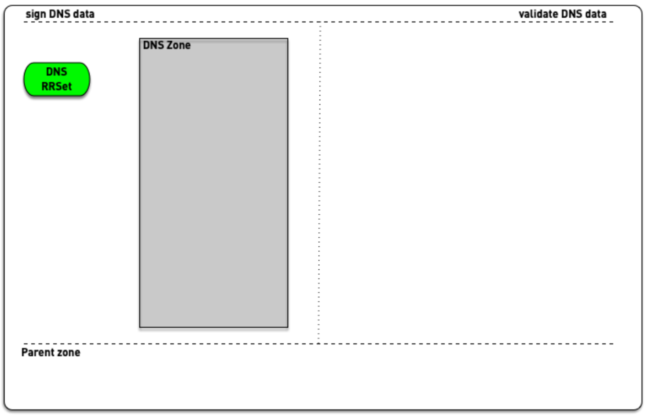
- Answer these questions into the chat

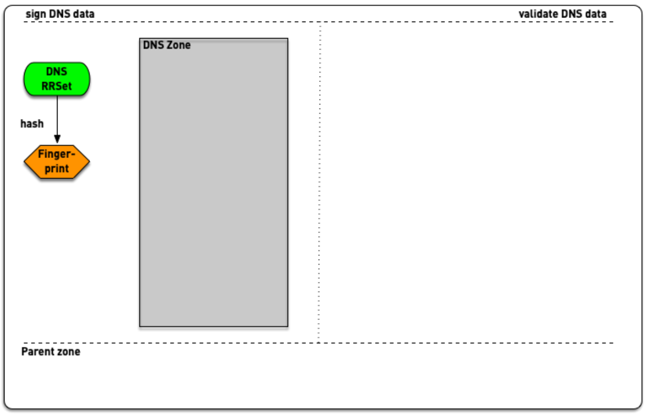
  Which algorithm is this zone using?

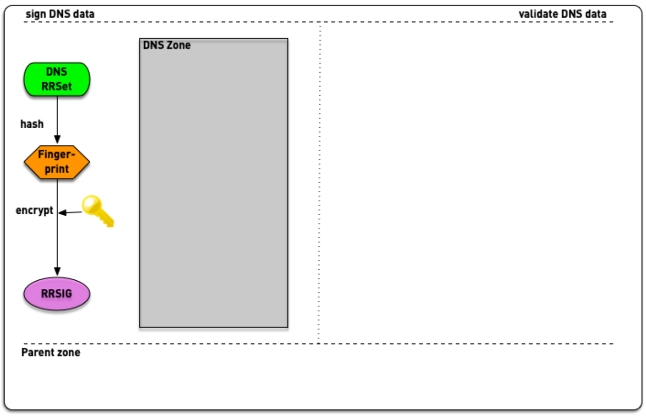
  What is the current key-id of the KSK in the zone `isc.org`?

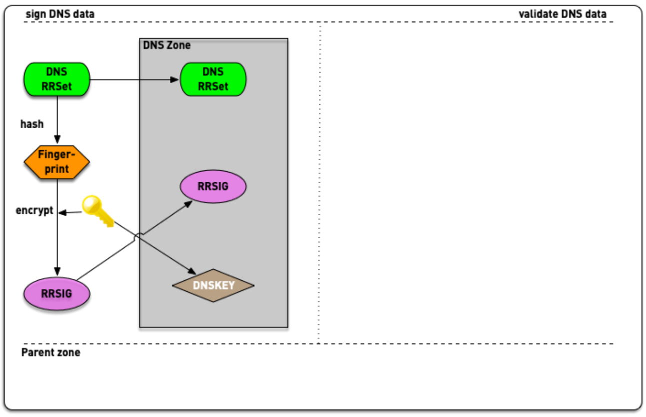
  What hashing algorithm is used for the DS record of `isc.org` check against Delegation Signer (DS) Resource Record (RR) Type Digest Algorithms [https://www.iana.org/assignments/ds-rr-types/ds-rr-types.xhtml]?

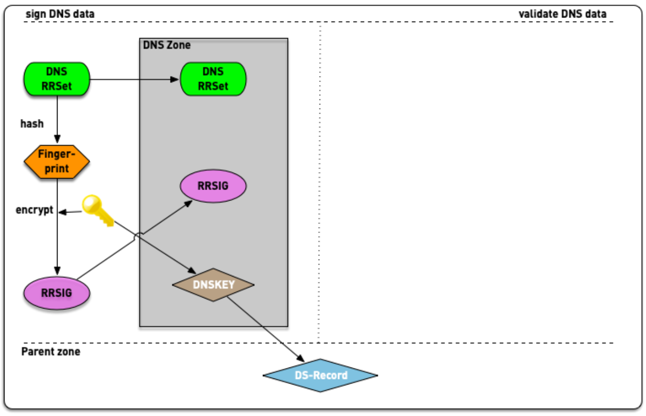
CHAPTER 7. DNSSEC SIGNING AND VALIDATION

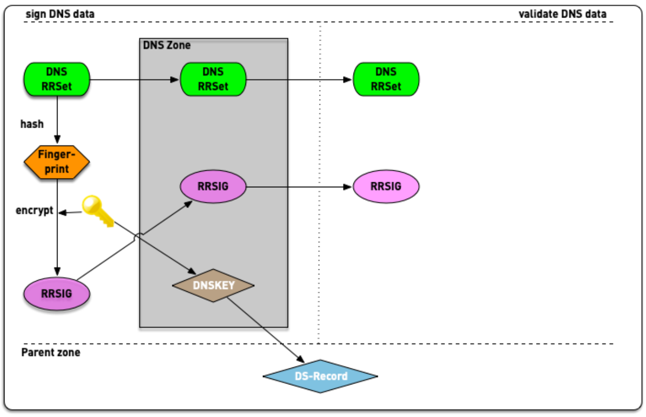


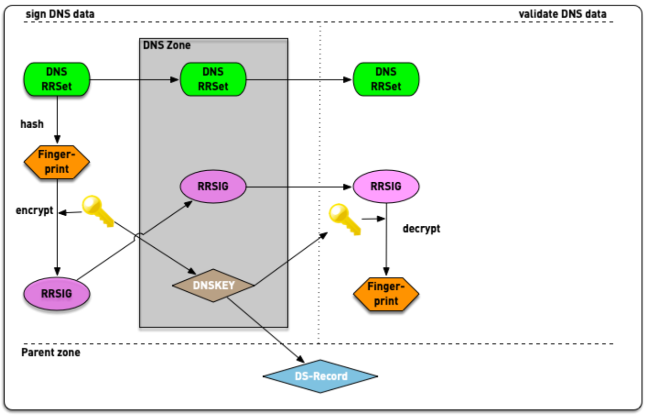


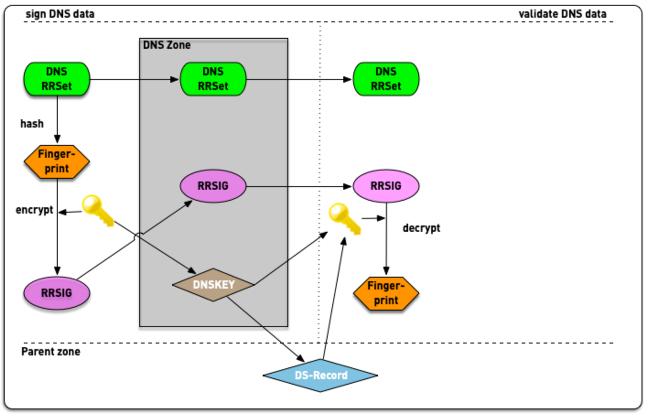


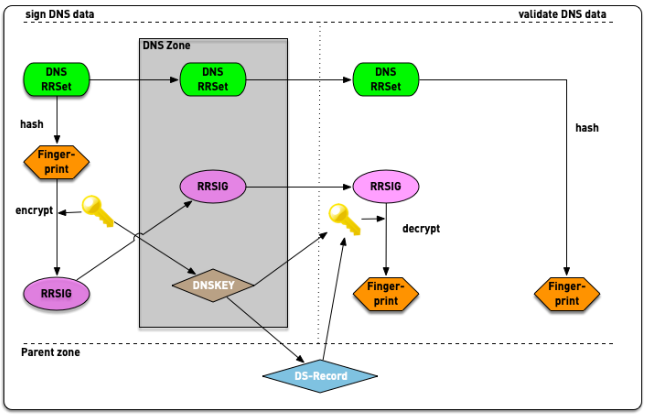

sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
hash
Finger-print
RRSIG
encrypt
RRSIG
DNSKEY
Parent zone

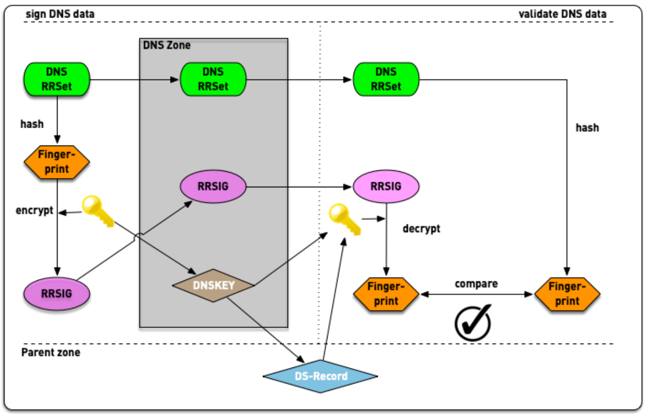

sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
hash
Finger-print
RRSIG
encrypt
RRSIG
DNSKEY
Parent zone
DS-Record


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
Finger-print
RRSIG
RRSIG
encrypt
RRSIG
DNSKEY
Parent zone
DS-Record


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
Finger-print
RRSIG
RRSIG
encrypt
decrypt
RRSIG
DNSKEY
Finger-print
Parent zone
DS-Record


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
Finger-print
RRSIG
RRSIG
encrypt
decrypt
RRSIG
DNSKEY
Finger-print
Parent zone
DS-Record


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
hash
Finger-print
RRSIG
RRSIG
encrypt
decrypt
RRSIG
DNSKEY
Finger-print
Finger-print
Parent zone
DS-Record


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
hash
Finger-print
RRSIG
RRSIG
encrypt
decrypt
RRSIG
DNSKEY
Finger-print
compare
Finger-print
Parent zone
DS-Record

CHAPTER 8. DNSSEC VALIDATION

8.1. DNSSEC IN DNS MESSAGES

- Even if a client does not explicitly request DNSSEC by setting the DO flag in a recursive query, it is protected.

    It benefits from DNSSEC, if its recursive server is configured for DNSSEC.

    The recursive server will return SERVFAIL if the requested RRSet proves unauthentic.

- `DO` Flag: DNSSEC OK

    The client is requesting the authoritative server to return the RRSIG together with the queried RRSet.

        The client is commonly a resolver, but it could be a stub resolver or application.

        With EDNS the client also announces the UDP packet size it will handle.

        If the DO flag is not set, RRSIGs are not returned by authoritative servers.

- `AD` Flag: Authentic Data

    The resolver uses the AD flag to inform a DNSSEC-aware client that it has successfully authenticated the RRSet.

    An unauthentic RRSet will not be sent to the client; the resolver returns SERVFAIL.

        Without DNSSEC, SERVFAIL means an authoritative server isn't responding.

    DO is client (resolver) to authoritative server, and AD is recursive resolver to client.

- `CD` Flag: Checking Disabled

    An application or stub resolver can tell the recursive server that it will validate the DNSSEC information itself.

    This is ideal where the recursive server can't be trusted.

        A recursive server is often run by a third party, e.g. an ISP or Internet cafe, and therefore is not inherently trustworthy.

        Additionally, the connection between the application or stub resolver with even a trusted recursive server, is likely insecure.

    There is no way for a client to force a recursive resolver to use, or not use, DNSSEC.

    CD with DO tells the recursive resolver to return the queried RRSet regardless of its authentication.

        The CD flag is an excellent debugging resource.

        For example, if a client receives SERVFAIL, requerying with dig can indicate if the problem is DNSSEC or not.

```
    $ dig example.com +cdflag
```

+ image::./img/DNSSEC-in-DNS-messages.png[scaledwidth=100%]

8.2. BASICS OF DNSSEC VALIDATION

- When the validator gets an RRSIG in a response, it needs the DNSKEY and DS RR to authenticate.

  If validation fails, the signed RRs are discarded, and SERVFAIL error is returned to the client.

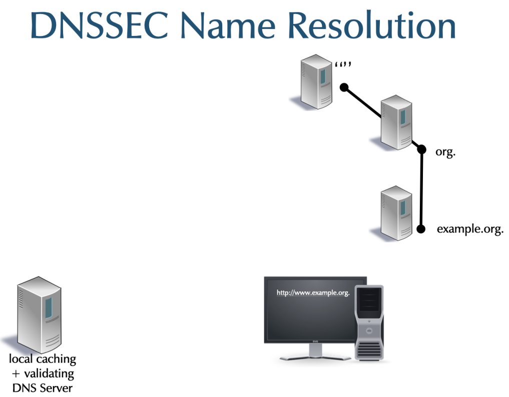
  If no appropriate trust anchor exists, the RRSIG is ignored.

  If the chain of trust is broken the signature is ignored.

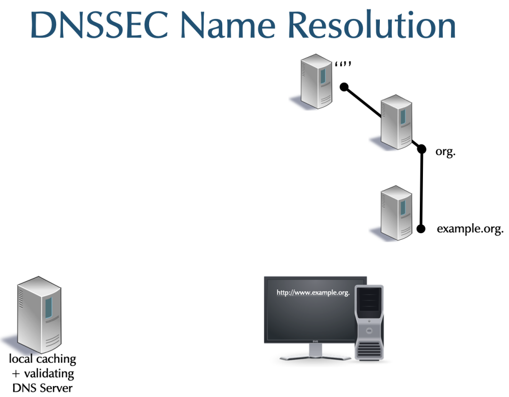
- The steps in the following animation are simplified.

  It only shows validation using one key per zone (SSK/CSK).

  Commonly, a zone has ZSK & KSK, so there are additional steps.

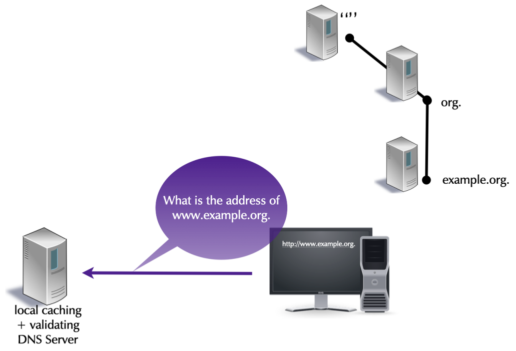


DNSSEC Name Resolution

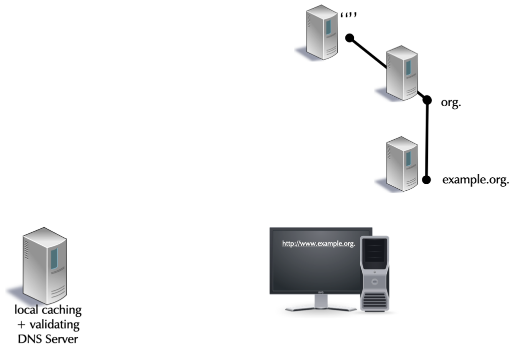


DNSSEC Name Resolution

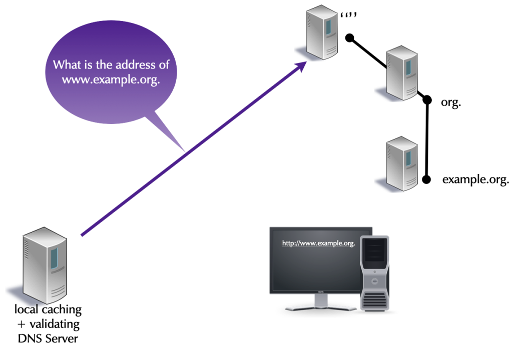
DNSSEC Name Resolution


“.”
org.
example.org.
What is the address of www.example.org.
http://www.example.org.
local caching + validating DNS Server

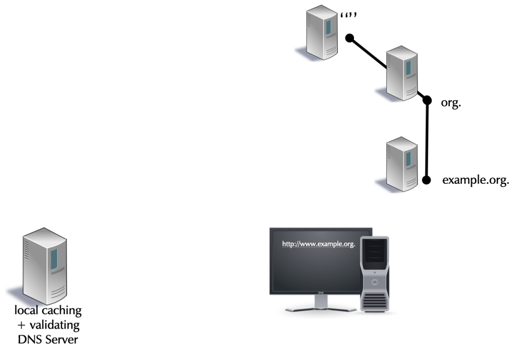
DNSSEC Name Resolution



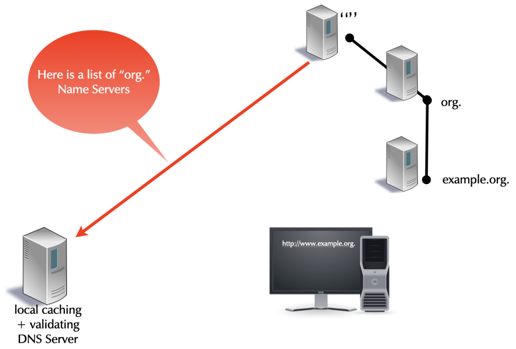
DNSSEC Name Resolution


“.”
What is the address of www.example.org.
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

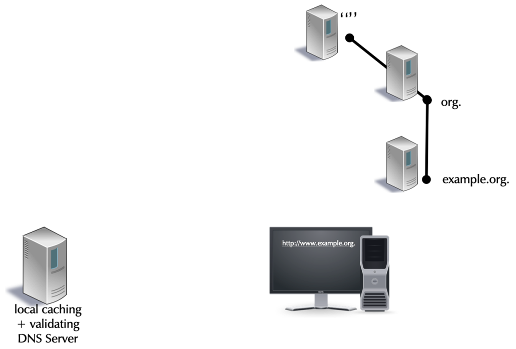
DNSSEC Name Resolution



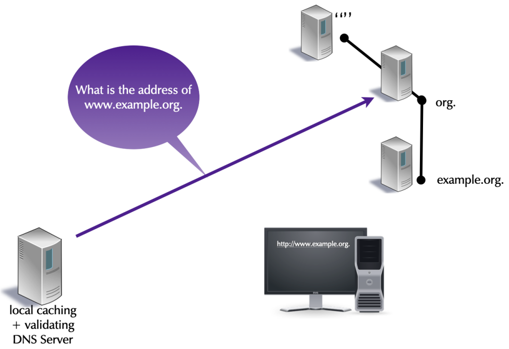
DNSSEC Name Resolution



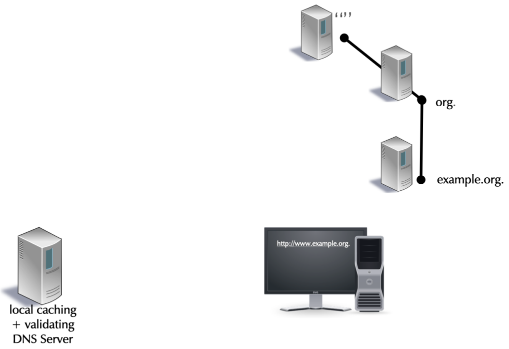
DNSSEC Name Resolution



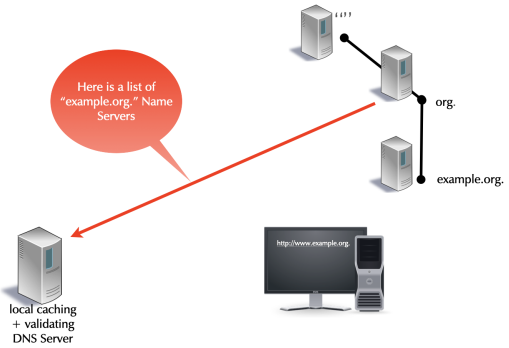
DNSSEC Name Resolution


“”
org.
example.org.
What is the address of www.example.org.
http://www.example.org.
local caching + validating DNS Server

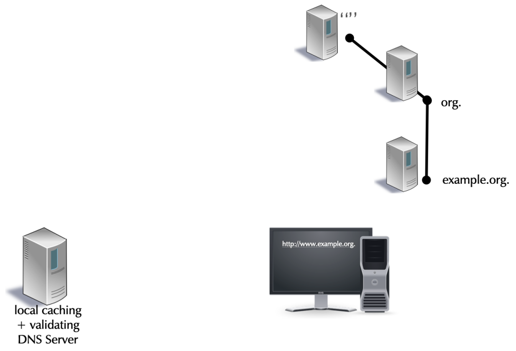
DNSSEC Name Resolution



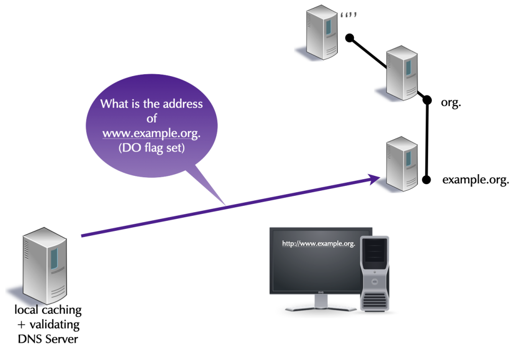
DNSSEC Name Resolution


“”
org.
example.org.
Here is a list of “example.org.” Name Servers
http://www.example.org.
local caching + validating DNS Server

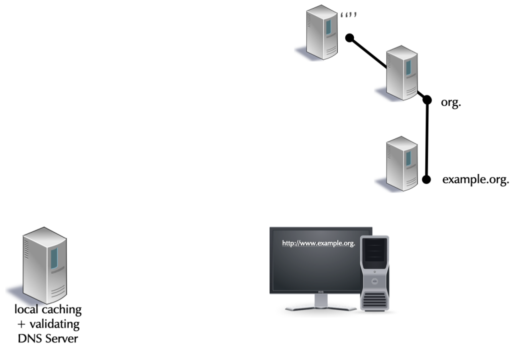
DNSSEC Name Resolution



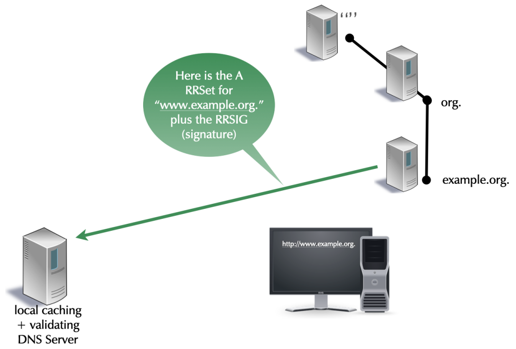
DNSSEC Name Resolution



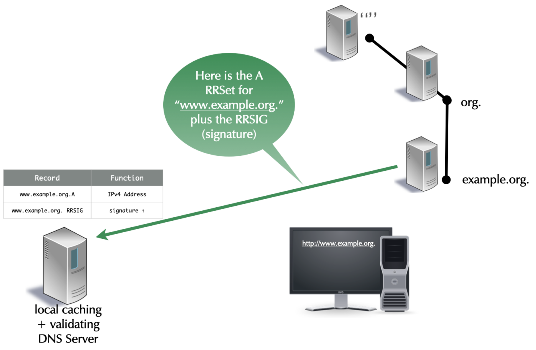
DNSSEC Name Resolution



DNSSEC Name Resolution

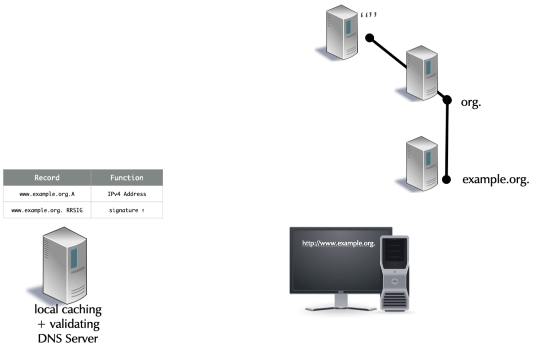

"."
org.
example.org.
Here is the A RRSet for "www.example.org." plus the RRSIG (signature)
http://www.example.org.
local caching + validating DNS Server

DNSSEC Name Resolution


"."
org.
example.org.
Here is the A RRSet for "www.example.org." plus the RRSIG (signature)
http://www.example.org.
local caching + validating DNS Server

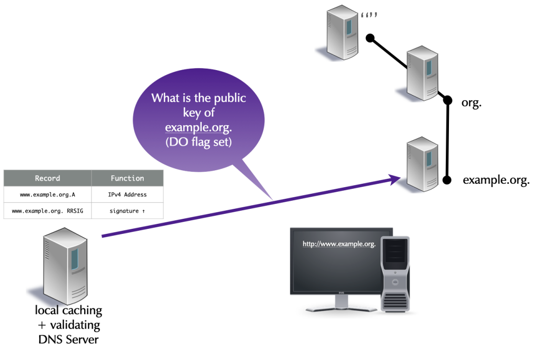
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑

DNSSEC Name Resolution



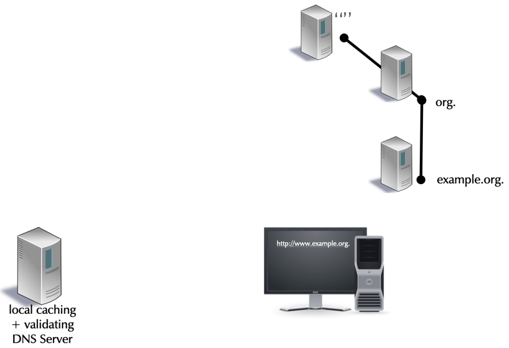
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑

DNSSEC Name Resolution

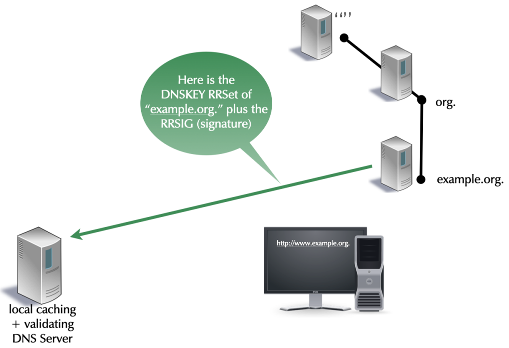

"."
org.
example.org.
What is the public key of example.org. (DO flag set)
http://www.example.org.
local caching + validating DNS Server

Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑

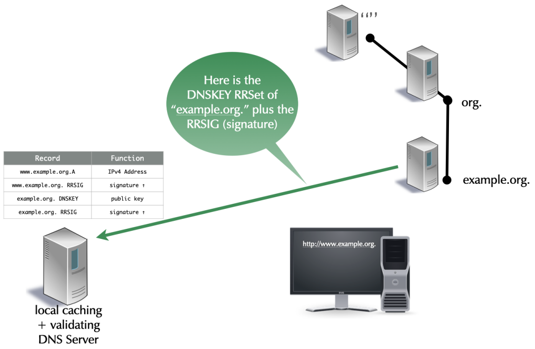
DNSSEC Name Resolution



DNSSEC Name Resolution

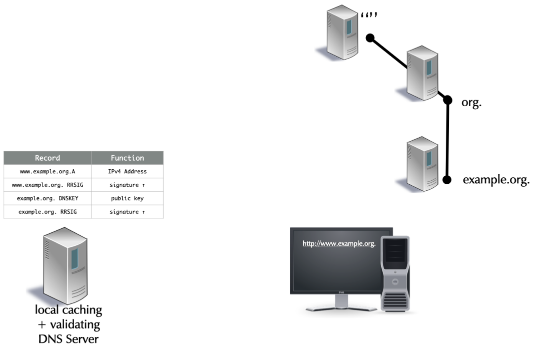

"."
org.
example.org.
Here is the DNSKEY RRSet of "example.org." plus the RRSIG (signature)
http://www.example.org.
local caching + validating DNS Server

DNSSEC Name Resolution


"."
org.
example.org.
Here is the DNSKEY RRSet of "example.org." plus the RRSIG (signature)
http://www.example.org.
local caching + validating DNS Server

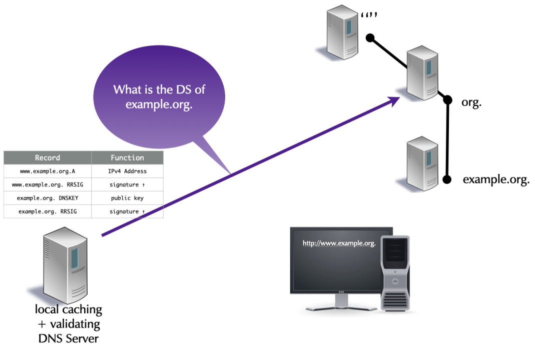
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑

DNSSEC Name Resolution



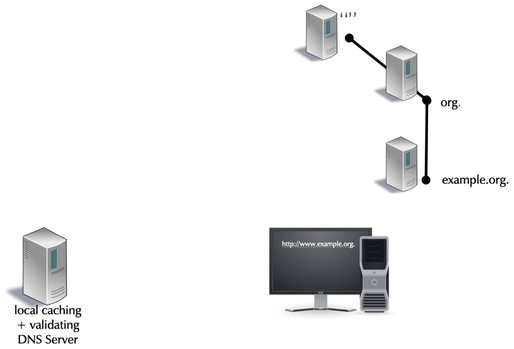
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑

DNSSEC Name Resolution

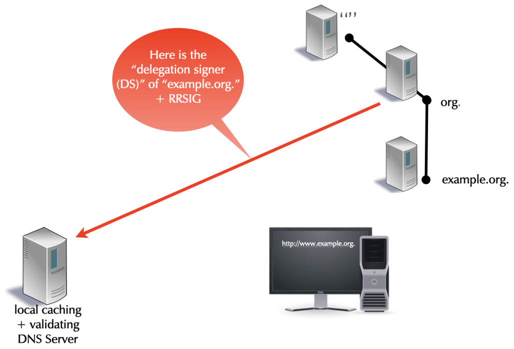

"."
org.
example.org.
What is the DS of example.org.
http://www.example.org.
local caching + validating DNS Server

Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑

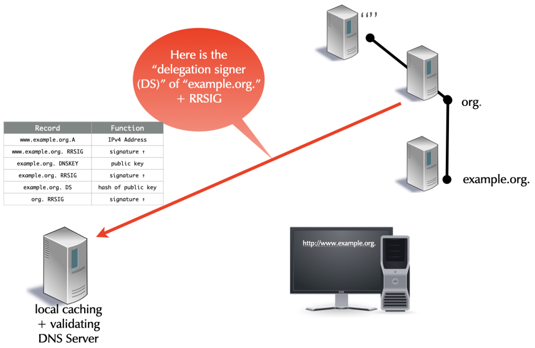
DNSSEC Name Resolution



DNSSEC Name Resolution

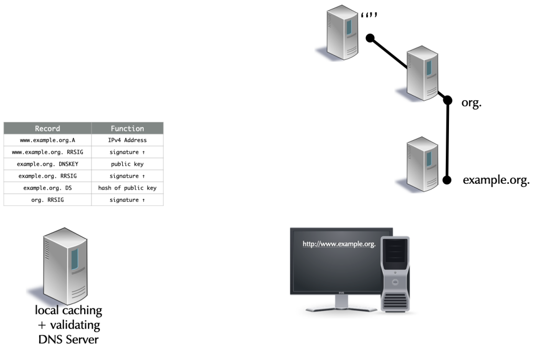


DNSSEC Name Resolution



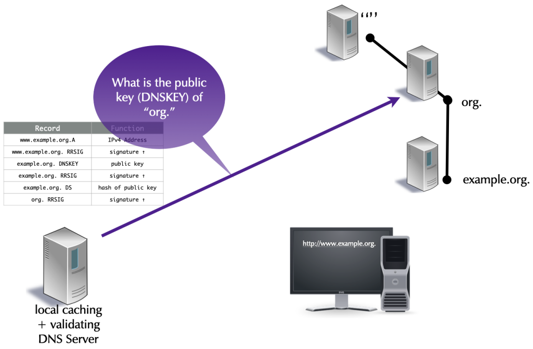
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑

DNSSEC Name Resolution



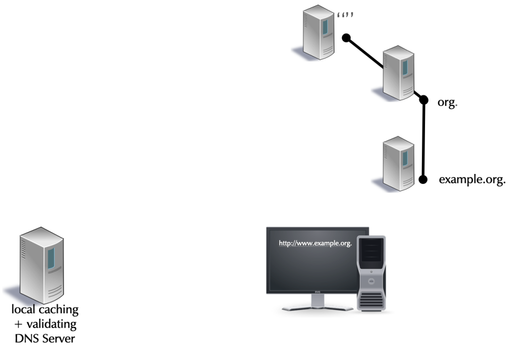
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑

DNSSEC Name Resolution

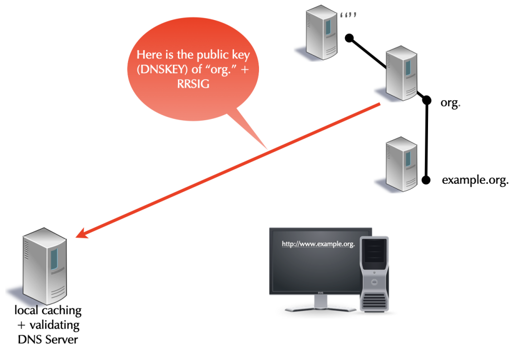


Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑

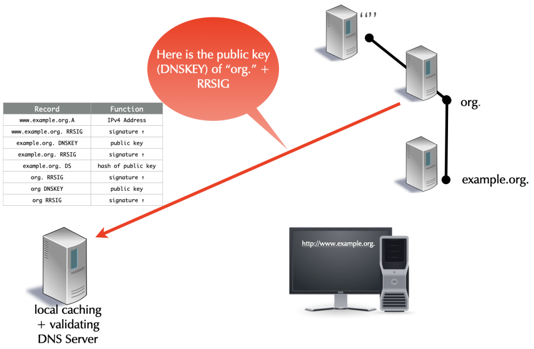
DNSSEC Name Resolution



DNSSEC Name Resolution

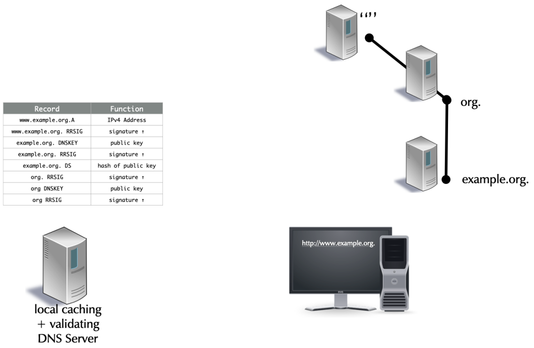

Here is the public key (DNSKEY) of "org." + RRSIG
""
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

DNSSEC Name Resolution


Here is the public key (DNSKEY) of "org." + RRSIG
""
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

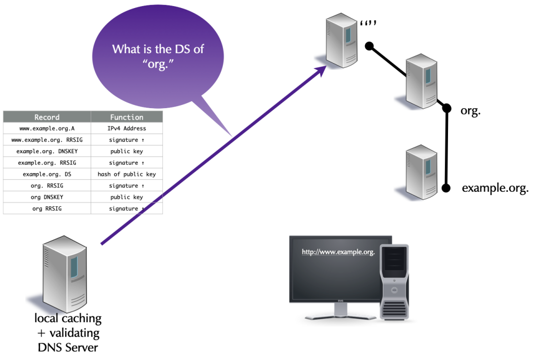
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑

DNSSEC Name Resolution



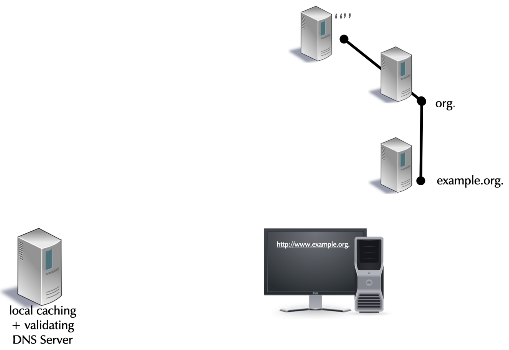
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑

DNSSEC Name Resolution

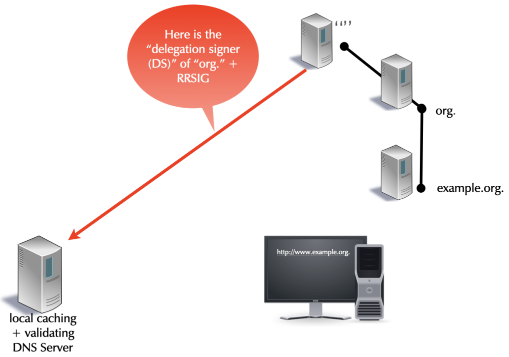


Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑

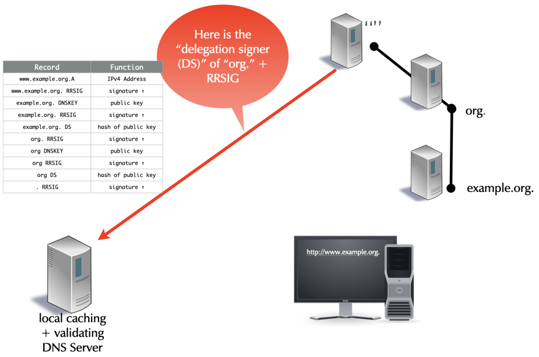
DNSSEC Name Resolution



DNSSEC Name Resolution

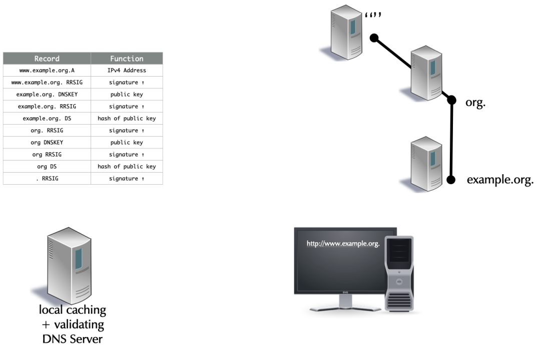


DNSSEC Name Resolution


Here is the “delegation signer (DS)” of “org.” + RRSIG
“.”
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

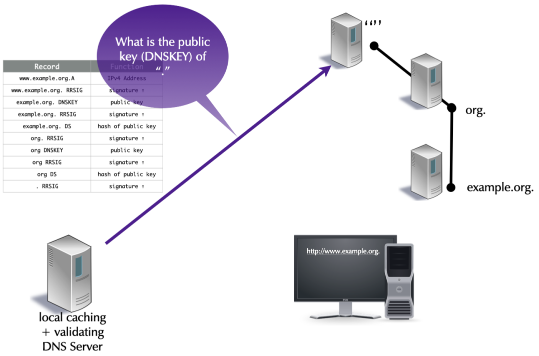
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑

DNSSEC Name Resolution



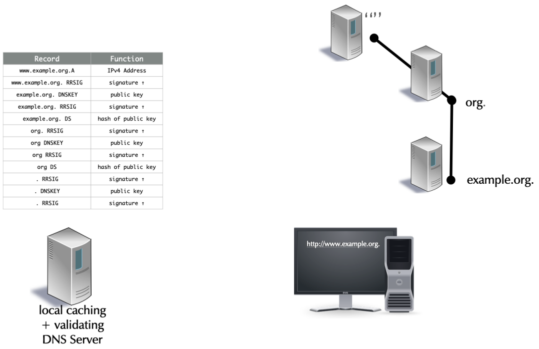
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑

DNSSEC Name Resolution


What is the public key (DNSKEY) of “.”
“.”
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

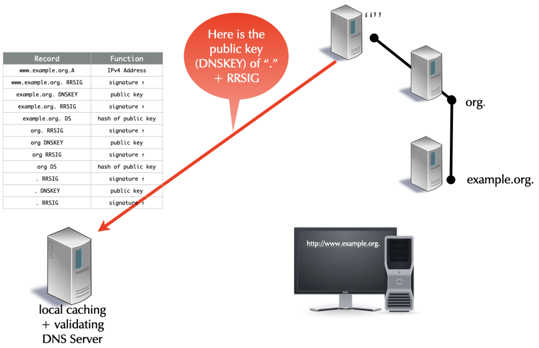
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑

DNSSEC Name Resolution



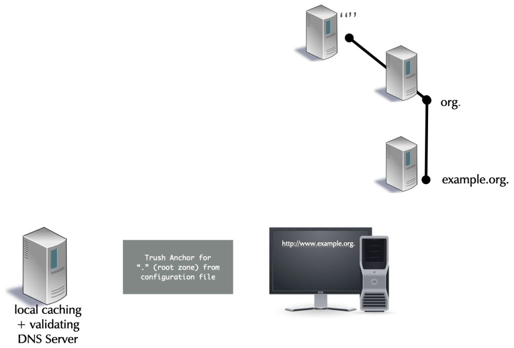
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑
. DNSKEY	public key
. RRSIG	signature ↑

DNSSEC Name Resolution



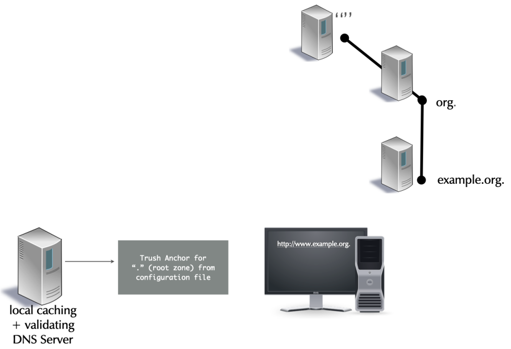
Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑
. DNSKEY	public key
. RRSIG	signature ↑

DNSSEC Name Resolution

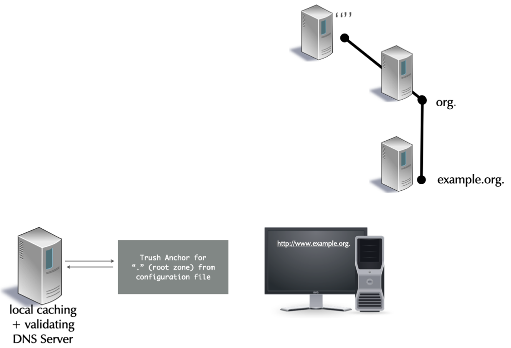


Trush Anchor for
“.” (root zone) from
configuration file

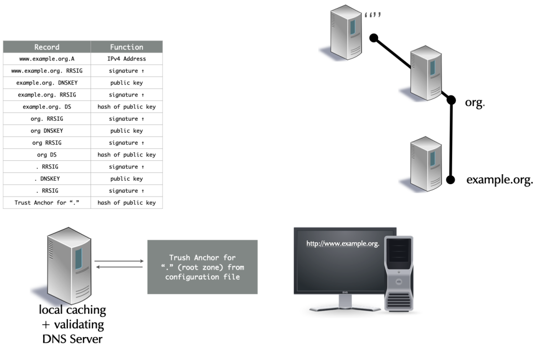
DNSSEC Name Resolution


“.”
org.
example.org.
Trush Anchor for “.” (root zone) from configuration file
http://www.example.org.
local caching + validating DNS Server

DNSSEC Name Resolution

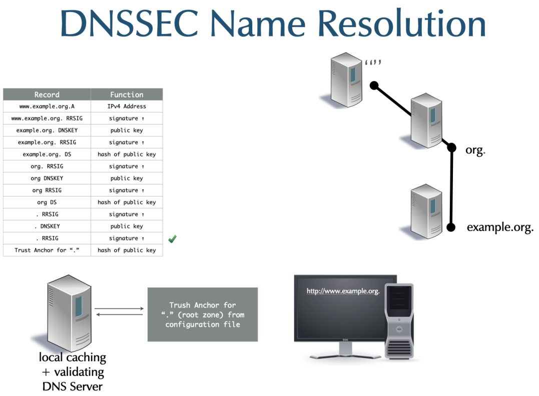

“.”
org.
example.org.
Trush Anchor for “.” (root zone) from configuration file
http://www.example.org.
local caching + validating DNS Server

DNSSEC Name Resolution



Record	Function
www.example.org.A	IPv4 Address
www.example.org. RRSIG	signature ↑
example.org. DNSKEY	public key
example.org. RRSIG	signature ↑
example.org. DS	hash of public key
org. RRSIG	signature ↑
org DNSKEY	public key
org RRSIG	signature ↑
org DS	hash of public key
. RRSIG	signature ↑
. DNSKEY	public key
. RRSIG	signature ↑
Trust Anchor for “.”	hash of public key

“.”
org.
example.org.
Trush Anchor for “.” (root zone) from configuration file
http://www.example.org.
local caching + validating DNS Server

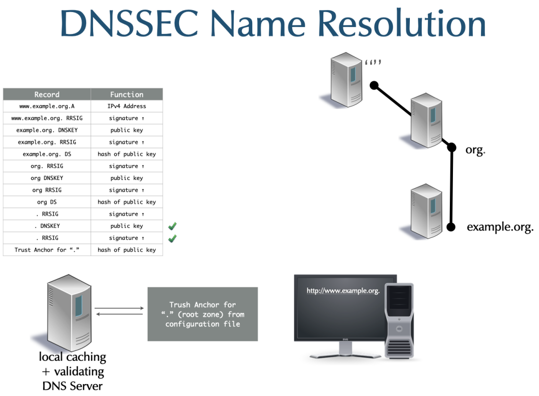


DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	
org RRSIG	signature ↑	
org DS	hash of public key	
. RRSIG	signature ↑	
. DNSKEY	public key	
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

Trush Anchor for "." (root zone) from configuration file

local caching
+ validating
DNS Server

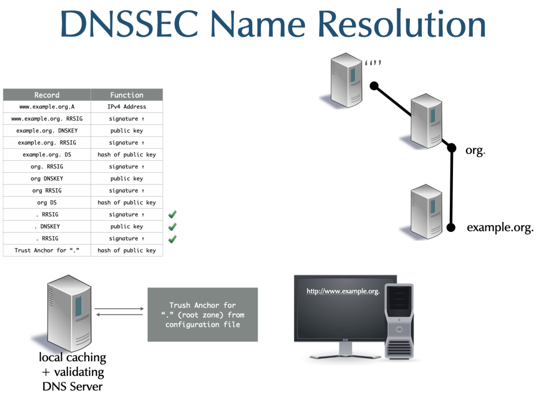


DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	
org RRSIG	signature ↑	
org DS	hash of public key	
. RRSIG	signature ↑	
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

Trush Anchor for "." (root zone) from configuration file

local caching
+ validating
DNS Server

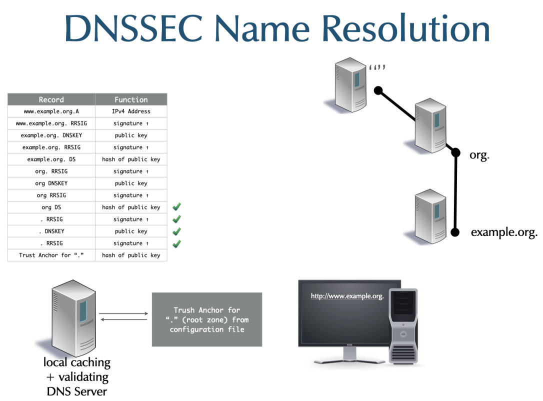


DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	
org RRSIG	signature ↑	
org DS	hash of public key	
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

Trush Anchor for "." (root zone) from configuration file

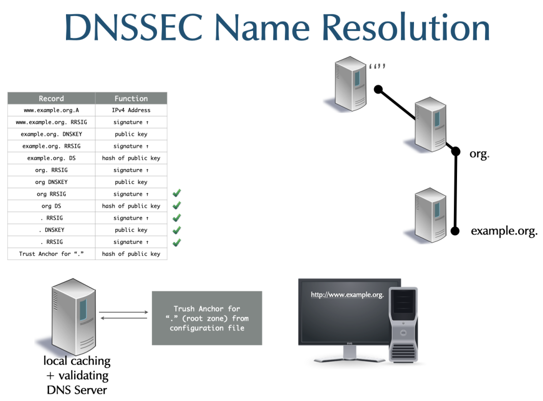
local caching
+ validating
DNS Server



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	
org RRSIG	signature ↑	
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

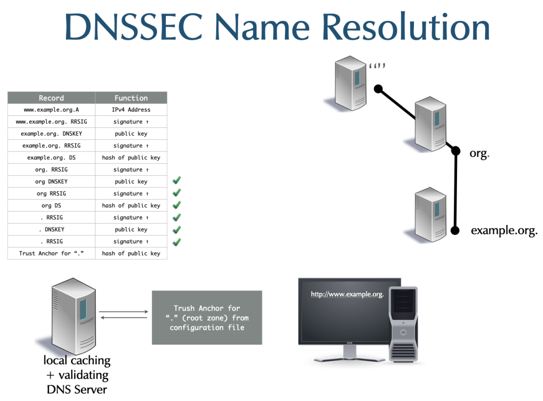
Trush Anchor for "." (root zone) from configuration file



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

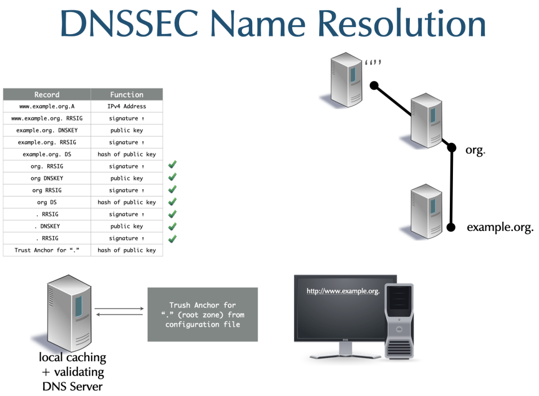
Trush Anchor for "." (root zone) from configuration file



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for "."	hash of public key	

Trush Anchor for "." (root zone) from configuration file



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“.”

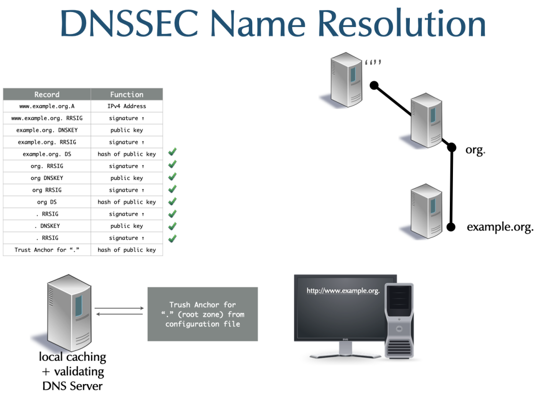
org.

example.org.

Trush Anchor for “.” (root zone) from configuration file

http://www.example.org.

local caching
+ validating
DNS Server



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	
example.org. DS	hash of public key	✓
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“.”

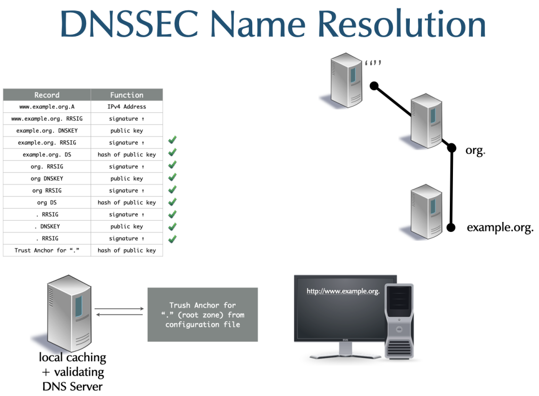
org.

example.org.

Trush Anchor for “.” (root zone) from configuration file

http://www.example.org.

local caching
+ validating
DNS Server



DNSSEC Name Resolution

Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	
example.org. RRSIG	signature ↑	✓
example.org. DS	hash of public key	✓
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“.”

org.

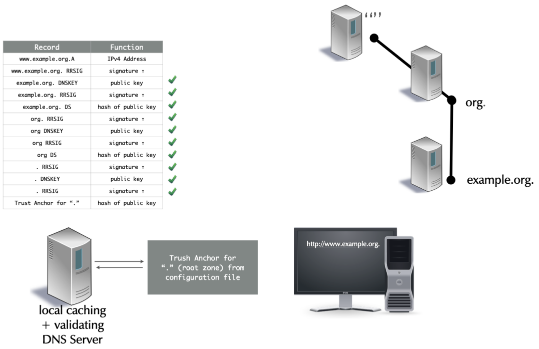
example.org.

Trush Anchor for “.” (root zone) from configuration file

http://www.example.org.

local caching
+ validating
DNS Server

DNSSEC Name Resolution



Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	
example.org. DNSKEY	public key	✓
example.org. RRSIG	signature ↑	✓
example.org. DS	hash of public key	✓
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“”

org.

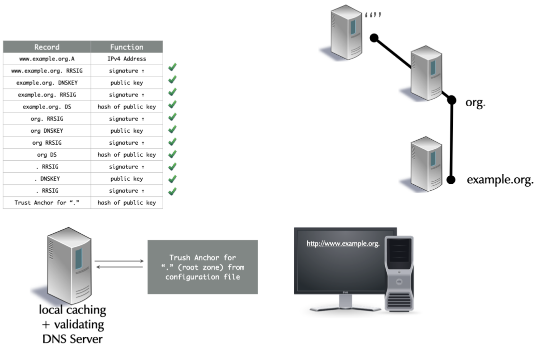
example.org.

local caching
+ validating
DNS Server

Trush Anchor for
“.” (root zone) from
configuration file

http://www.example.org.

DNSSEC Name Resolution



Record	Function	
www.example.org.A	IPv4 Address	
www.example.org. RRSIG	signature ↑	✓
example.org. DNSKEY	public key	✓
example.org. RRSIG	signature ↑	✓
example.org. DS	hash of public key	✓
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“”

org.

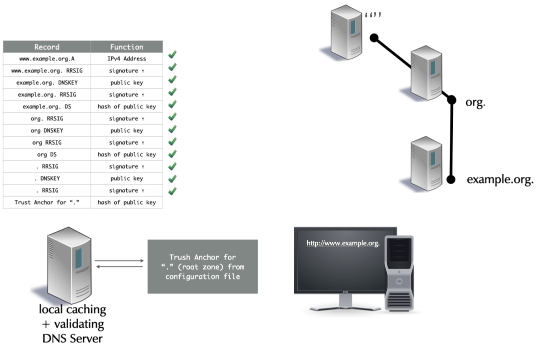
example.org.

local caching
+ validating
DNS Server

Trush Anchor for
“.” (root zone) from
configuration file

http://www.example.org.

DNSSEC Name Resolution



Record	Function	
www.example.org.A	IPv4 Address	✓
www.example.org. RRSIG	signature ↑	✓
example.org. DNSKEY	public key	✓
example.org. RRSIG	signature ↑	✓
example.org. DS	hash of public key	✓
org. RRSIG	signature ↑	✓
org DNSKEY	public key	✓
org RRSIG	signature ↑	✓
org DS	hash of public key	✓
. RRSIG	signature ↑	✓
. DNSKEY	public key	✓
. RRSIG	signature ↑	✓
Trust Anchor for “.”	hash of public key	

“”

org.

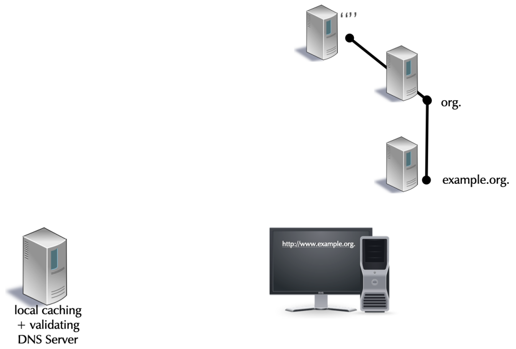
example.org.

local caching
+ validating
DNS Server

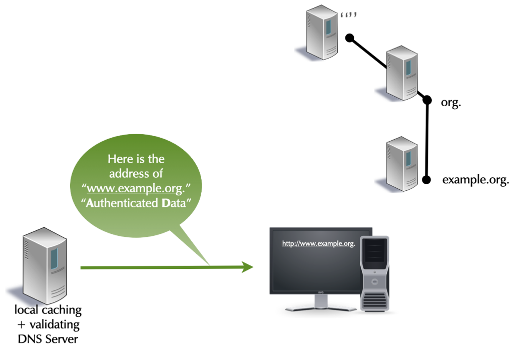
Trush Anchor for
“.” (root zone) from
configuration file

http://www.example.org.

DNSSEC Name Resolution



DNSSEC Name Resolution



8.3. OPTIMIZATIONS TO THE DEFAULT CONFIGURATION OF A VALIDATING DNSSEC RESOLVER

- Work on the `dnsrNNN` virtual machine (DNS resolver server)
- Let's tweak the configuration file `/etc/bind/named.conf.options` for a resolver DNS server. Add the following new lines in the `options` block in the `/etc/bind/named.conf.options` file:

```
options {
        [...]
        dnssec-validation auto;
        server-id none;
        version none;
        hostname none;
        recursive-clients 32768;
        tcp-clients 1024;
        max-clients-per-query 1024;
        fetches-per-zone 2048;
        fetches-per-server 4096;
        edns-udp-size 1232;
        max-udp-size 1232;
        minimal-responses yes;
        querylog no;
        max-cache-size 2147483648;
};
```

- The values above are for an ISP level DNS resolver. They should work for a broad range of DNS resolver machines, from small to very large.

- The new configuration statements:

  `dnssec-validation auto;` enables DNSSEC validation via the built in trust anchor for the Internet Root-Zone. If DNSSEC is used in an closed private DNS system (not connected to the Internet), dedicated DNSSEC trust anchor must be configured. This setting is the default in

  server-id none: disable returning the server's hostname on the query `dig @ip-of-server ch TXT hostname.bind`

  version none: disable returning the BIND 9 version number on the query `dig @ip-of-server ch TXT version.bind`

  recursive-clients: number of concurrent DNS queries over UDP that are allowed from clients

  tcp-clients: number of concurrent DNS queries over TCP that are allowed from clients

  max-clients-per-query 1024: rate-limiting - number of concurrent clients that can request the same query

  fetches-per-zone: rate-limiting - maximum number of concurrent DNS queries inside one zone

  fetches-per-server: rate-limiting - maximum number of concurrent DNS queries towards one authoritative DNS server

  edns-udp-size: maximum UDP answer size requested from authoritative servers

  max-udp-size: maximum UDP answer size when sending answers to clients

  minimal-responses yes: only fill the authority and additional sections when necessary by the DNS protocol

  querylog no: disable query logging on start/restart even if configured. Query logging slows down the DNS resolver

  max-cache-size 2147483648: use a maximum of 2GB for the DNS cache. A larger cache often has a negative impact on the DNS resolvers performance. If unsure, test with a performance benchmark.

- Check the configuration with `named-checkconf` and reload the BIND 9 configuration with `rndc reconfig`
- Make sure that the DNS resolver still does resolve DNS names in the Internet

CHAPTER 9. DNSSEC SIGNING WITH BIND 9

9.1. MANUAL ZONE SIGNING

- Since BIND 9.6 zones can be signed manually (BIND 9.6 was the first version to support the new DNSSEC version)
- Benefits:

  The signing can be done *offline* on a machine not connected to any network, BIND 9 does not need to be running on the signing machine

  The DNSSEC private keys can be stored on security devices such as Hardware Security Modules (HSM) and can be encrypted

  The generated DNSSEC signed zone files are universal and can be used in any DNS server that supports DNSSEC signed zones

  The parameters of the signing process can be tuned

- Drawbacks:

  Manually signed zones need to be refreshed (re-signed) periodically so that the signatures do not expire

  Any automation must be done through custom scripts

9.2. MANUAL SIGNING WITH BIND 9

- These are the steps to manually sign a zone for BIND 9
- For older BIND 9 versions, enable DNSSEC in the `named.conf` configuration file

```
        options {
           directory "/var/named";
           dnssec-enable yes;
        };
```

- Create a Zone-Signing-Key (ZSK)

  This requires good real entropy (randomness) in the operating system. If the key creation process takes too long (more than a minute for a key), there might not be enough entropy in the system. In such case a hardware random number generator or a software entropy gathering daemon (such as haveged [http://issihosts.com/haveged/]) can help

```
     % dnssec-keygen -a RSASHA256 -b 1536 zoneNNN.dnslab.org
     Generating key pair.........++++ ...........................
     KzoneNNN.dnslab.org.+008+16239
+
     % more KzoneNNN.dnslab.org.+008+16239.key
     ; This is a zone-signing key, keyid 16239, for zoneNNN.dnslab.org.
     ; Created: 20160202121320 (Tue Feb  2 13:13:20 2016)
     ; Publish: 20160202121320 (Tue Feb  2 13:13:20 2016)
     ; Activate: 20160202121320 (Tue Feb  2 13:13:20 2016)
     zoneNNN.dnslab.org. IN DNSKEY 256 3 8 AwEAAc1xFtt40wPEx4TVB7h8Ac7HvMZuF1LIqESU/0HUUzDT2rkujMdL
     z0fgJJQVStYIbb1fXN0/PmKayEpj5ScbT7WU9Bef6b49uG1PwhsaftRr
     /udr3DEA6MTEdRqkl8K+E3P9hFj4XKxus45MYVSPaXZg3TcIQK3xpXC8
     sKISny43cQaJpm12oBtKsANlA25KRJC8soP1s/GqLSnArWDMN/YGqvs0
```

```
    QECulpm2Nh1uULZfzwga8515xizyx5yAl/sgWQ==
```

- Generate a Key-Signing-Key with the same DNSSEC algorithm used for the ZSK

```
        % dnssec-keygen -a RSASHA256 -b 2048 -f KSK zoneNNN.dnslab.org
        Generating key pair...............................................
        KzoneNNN.dnslab.org.+008+04351
+
        % more KzoneNNN.dnslab.org.+008+04351.key
        ; This is a key-signing key, keyid 4351, for zoneNNN.dnslab.org.
        ; Created: 20160202121714 (Tue Feb  2 13:17:14 2016)
        ; Publish: 20160202121714 (Tue Feb  2 13:17:14 2016)
        ; Activate: 20160202121714 (Tue Feb  2 13:17:14 2016)
        zoneNNN.dnslab.org. IN DNSKEY 257 3 8
AwEAAcmn/QkiCne922gBBBuJJOnq9jnG2yYbB10zBS2SgUCUxlZfM2ja
        PAyubB2V+QhFsKf0VKUsVGl28JWAMcG1NGitj+nna4sGwvmeumj70DbG
        ZzynwcFknEZG1Swn2bM/OFmlMS2WV3luzDYKnLeZgvN5geB6ZetONlpP
        H9am3MRmExNIxoFb5NEcUlCzxSUI5GzjPZtGmCtDoNKrGE5nsssCgrjw
        ec6hbeXLOjP9JiQ3egF3+PJHLUOjuqXKwSofHw4jV4Rqc3eP+uAHk5Wp
        iH/BNW7c7lJ9IP+jZYZ3dp3SkO2qU8BOVV4fcm1L+IVcA9jwuuPaOV53
        j9L8fCTL/Uk=
```

- This is the content of the unsigned DNS zone

```
        $TTL 3600
        $ORIGIN zoneNNN.dnslab.org.
        @             IN SOA serverNNN.dnslab.org.  hostmaster.zoneNNN.dnslab.org. (
                             1001 ; serial
                             1d   ; refreh
                             2h   ; retry
                             4w   ; expire
                             30m  ; negTTL
                          )
                      IN NS serverNNN.dnslab.org.
                      IN MX 10 mail.zoneNNN.dnslab.org.
        www           IN A  192.168.53.199
        mail          IN A  192.168.53.199
```

- Add the public part of the ZSK and KSK to the zone (be careful with the shell redirection!) or use master file `$INCLUDE` syntax.

```
        % cat KzoneNNN.dnslab.org.+008+*.key >> zoneNNN.dnslab.org
```

- Sign the zone file

```
        % dnssec-signzone -o zoneNNN.dnslab.org \
            -k KzoneNNN.dnslab.org.+008+04351.private \
                zoneNNN.dnslab.org \
        KzoneNNN.dnslab.org.+008+16239.private
        Verifying the zone using the following algorithms: RSASHA256.
        Zone fully signed:
        Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                              ZSKs: 1 active, 0 stand-by, 0 revoked
        zoneNNN.dnslab.org.signed
```

- Zone signing tool syntax:

```
        dnssec-signzone -o <origin> -k <KSK-private> <zone-file> <ZSK-private>
```

- If the error message `dnssec-signzone: fatal: SOA is not signed (keys offline or inactive?)` is displayed, then the KSK and ZSK have been given in the wrong order

- Additional options for `dnssec-signzone`

  `-j sec` Jitter, variation in seconds of the signature validity

  `-M maxttl` - specifies the maximum allowed TTL in the signed zone. Higher TTL values will be capped to this number.

  `-s starttime` - start of validity of the signatures

  `-e endtime` - expire time of the signatures

  `-N SOA-format` - format of the SOA serial number

    `keep` do not modify the current SOA serial number

    `increment` increment the current SOA by one

    `date` set the SOA serial number to today's date in `YYYYMMDDnn` format

    `unixtime` use the Unixtime (seconds since 1.1.1970) as the SOA serial

  `-x` only create one signature for the DNSKEY record set (using the KSK)

  `-n numcpus` use this amount of CPU cores for signing

  `-t` print performance statistics after signing

    The signed zone is stored in a new file with the name of the original zone file and the extension `.signed`

```
     zoneNNN.dnslab.org.      3600    IN SOA  serverNNN.dnslab.org. hostmaster.zoneNNN.dnslab.org. (
                                              1001       ; serial
                                              86400      ; refresh (1 day)
                                              7200       ; retry (2 hours)
                                              2419200    ; expire (4 weeks)
                                              1800       ; minimum (30 minutes)
                                              )
                              3600    RRSIG   SOA 8 3 3600 (
                                              20160303114542 20160202114542 16239
zoneNNN.dnslab.org.

                                              jT+M9IhjViHkoLnC5/y+MaHYoxpcoyHvif7H
                                              sFESQfk7JBx654zb7OZ2LVxsmMnGtiqxkLlD
                                              l5nJtBJuWXMufPLks+qQb42YjdGgU6vf5WOI
                                              GkTyTMf0ZcNc3ULZZCMG/Vkf3Pak4O6He+cB
                                              xvdbDzOOaLcQqlKH8xY/ylmHawJTm8Mmcbb/
                                              1tL3B/Iv0SI9Lv+F/g+ajaA7fd3bcr0Vueol
                                              gOTJ3OZkIEoQFROrn5UMQ/fvbY7Go2TjDT7I
                                              GYMu )
                              3600    NS      serverNNN.dnslab.org.
                              3600    RRSIG   NS 8 3 3600 (
                                              20160303114542 20160202114542 16239
zoneNNN.dnslab.org.

                                              ColPExa9EdSA1Nt1DsEtX5qjYzNWA8vUl8ef
                                              oJG409V6BX4JJsK7RJGCGlmqDIA7HO1IQKug
                                              64N+fD/IuVjqHsPxc/YtP1sdnnLjYK0rHgG7
                                              kMMkoqU7Ta/l/laKKd3jcI/kh66dOsTwYrEC
                                              aykvfzYCnVj/rxwEomu2aTbPh/Nt7V2OEXDT
                                              EqlmpS34yQ3LvYfnSTTipLYZNS/2kL7NRuEo
                                              1gId5PD/IMAKSpTqfilW1bQUf+CYJF/CGgF5
                                              KRIx )
```

- Compare the file sizes. The signed zone is much larger

```
         % wc zoneNNN.dnslab.org*
              23     133    1554 zoneNNN.dnslab.org
```

```
          130     314    5178 zoneNNN.dnslab.org.signed
          153     447    6732 total
```

- Adjust the zone definition in `named.conf` to load the new signed zone

```
      zone "zoneNNN.dnslab.org" IN {
          type primary;
          file "zoneNNN.dnslab.org.signed";
      };
```

- Check the new BIND 9 configuration. The output should indicate that the zone is now DNSSEC signed

```
      % named-checkconf -z
      zone zoneNNN.dnslab.org/IN: loaded serial 1001 (DNSSEC signed)
```

- Load the DNSSEC signed zone into the BIND 9 DNS server

```
      % rndc reload zoneNNN.dnslab.org
```

- Send the DS record for the KSK to the operator of the parent zone

```
      % cat dsset-zoneNNN.dnslab.org.
  zoneNNN.dnslab.org.  IN DS 16239 8 1 C0213 ... 0373A
      zoneNNN.dnslab.org.  IN DS 16239 8 2 EAC59 ... E694690565680E3F6D201 E2650EAE
```

9.3. EXERCISE: AN AUTHORITATIVE DNS SERVER

- Use your virtual lab machine, `authXX.dane.onl` (primary authoritative)
- Obtain *root* privileges

```
$ sudo -s
```

- BIND 9 is already installed and started

```
% systemctl status named
```

- Check the version and the compile time configuration of the installed BIND 9 server

```
% named -V
```

- Make the BIND 9 configuration file writeable

```
% chmod +w /etc/bind/named.conf
```

- Open the BIND 9 configuration file `/etc/bind/named.conf` with a text editor (`vim`, `nano`, `emacs` …)

```
% $EDITOR /etc/bind/named.conf
```

- Disable recursion `recursion no;`
- Adjust the `listen` directives to listen to `any` interface
- Add a sensible logging configuration for an authoritative BIND 9 server (here's a template you can use [https://dnslab.org/dns/logging.conf]).
- A primary zone definition for the zone `zXX.dane.onl` is already configured. Test if you can resolver this zone from the DNS resolver machine.

- Get the IPv6 Address of your authoritative server with `hostname -I` or `ip -6 address show dev eth0`. Add the (public) IPv6 address as an AAAA record to APEX (base domain name) the zonefile (filename `/etc/bind/zonefile.db`).
- Check the BIND 9 configuration file and the zone file for errors

```
% named-checkconf -z
```

- If no errors are reported, reload the new zone into the BIND 9 server (via `rndc`)

```
% rndc reload zXX.dane.onl
```

- Test of your DNS server will respond to queries for this zone

```
$ dig @127.0.0.1 zXX.dane.onl SOA
```

- Use an external DNS resolver (Quad9 in this example) to resolve data from the new DNS zone

```
$ dig @9.9.9.9 AAAA auth.zXX.dane.onl
```

9.4. EXERCISE: SIGNING BIND 9.6 STYLE (MANUAL SIGNING)

- Create a Zone Signing Key (ZSK), RSASHA256 Algorithm and 1024 bit length inside the directory `/etc/bind/manual-keys`

```
% mkdir /etc/bind/manual-keys
% cd /etc/bind/manual-keys
% dnssec-keygen -a RSASHA256 -b 1024 zXX.dane.onl
```

- Create a Key Signing Key (KSK), RSASHA256 Algorithm and 2048 bit length

```
% dnssec-keygen -a RSASHA256 -b 2048 -f KSK zXX.dane.onl
```

- Take a look at the generated key files, these are text files

```
ls -ltr /etc/bind/manual-keys
```

- Add the public part of the ZSK and KSK to the zone file and manually increment the SOA serial number (in a text editor)

```
% cat KzXX.dane.onl.+008+*.key >> /etc/bind/zonefile.db
% $EDITOR /etc/bind/zonefile.db
```

- DNSSEC sign the zone

```
% dnssec-signzone -o zXX.dane.onl \
       -k KzXX.dane.onl.+008+(KSK).private \
      /etc/bind/zonefile.db \
      KzXX.dane.onl.+008+(ZSK).private
```

- Adjust the zone definition in the Bind 9 configuration file to now load the signed version of the zone file

```
zone "zXX.dane.onl" {
      type primary;
      file "zonefile.db.signed";
};
```

- Test the BIND 9 configuration, and if no errors are shown, reload the signed zone into BIND 9

```
% named-checkconf -z
% rndc reconfig
```

- Check locally if DNSSEC records are returned from our DNS server

```
$ dig @localhost zXX.dane.onl SOA +dnssec +multi
```

- The DS record for the zone can be found in the file `dsset-zXX.dane.onl` in the directory where the zone has been signed. Copy and paste the DS record to the machine of the trainer (`scp dsset-zXX.dane.onl. user@auth.z15.dane.onl:.`). The trainer is operating the parent zone of your zone and will add the DS record there to close the chain of trust.

```
% cat dsset-zXX.dane.onl.
% scp dsset-zXX.dane.onl. user@auth.z15.dane.onl:.
```

- Wait for the DS record to be published in the parent zone
- Now test DNSSEC validation against your DNS resolver machine

```
$ dig @<IP-of-resolver> zXX.dane.onl SOA +dnssec +multi
```

- Query the DS record from the parent

```
$ dig @<IP-of-resolver> zXX.dane.onl DS +dnssec
```

- Make a small change to your zonefile (add a TXT record for `test.zXX.dane.onl`), increment the SOA serial (or use the command `dnssec-signzone` with the parameter `-N unixtime`)
- Test the zone file

```
% named-checkzone zXX.dane.onl /etc/bind/zonefile.db
```

- Re-Sign the zone

```
% dnssec-signzone -o <zonenname> -k <KSK-private-file> <zonefile> \
       <ZSK-private-file>
```

- Load the newly signed zone in BIND 9

```
% rndc reload zXX.dane.onl
```

- Check for the changes you made, are they visible? (Do you still see the AD-Flag?). Adjust the query below to match your changes to the zone file:

```
$ dig TXT text.zXX.dane.onl +dnssec +multi
```

CHAPTER 10. DNSSEC KEYS

10.1. ALGORITHMS FOR DNSSEC

DNSSEC keys can be generated with different crypto algorithms. Some of these algorithms are obsolete and deprecated, others are not (yet) widely supported by deployed DNS software to be useful

Algorithm	No.	Note
~~RSAMD5~~	1	deprecated, not implemented
~~RSASHA1~~	5	not recommend, deprecated for DNSSEC signing, not supported in Red Hat Enterprise Linux 9 (and up)
RSASHA256	8	recommended
RSASHA512	10	large keys, large signatures, risk of UDP fragmentation or TCP fallback
~~DSA~~	3	deprecated, slow validation, no extra security
~~ECC-GOST~~	12	deprecated
ECDSA	13/14	small signatures, read RSA vs ECDSA for DNSSEC [https://blog.apnic.net/2021/11/10/rsa-vs-ecdsa-for-dnssec/]

Algorithm	No.	Note
ED448/ED25519	16/15	not supported by legacy resolver RFC 8080 [https://tools.ietf.org/html/rfc8080] / RFC 8032 Edwards-Curve Digital Signature Algorithm (EdDSA) [https://tools.ietf.org/html/rfc8032] / Assessing DNSSEC with EdDSA [https://blog.apnic.net/2021/06/18/dnssec-with-eddsa/]

10.2. DNSSEC KEY SIZES FOR RSA ALGORITHM

- Every additional bit increases the strength of a DNSSEC against *brute-force* attacks to break the key
- Double the RSA key size results in

    Generating signatures is up to eight times more work

    Validation of DNSSEC signatures inside an DNS resolver up to 4 times more work

    Size of key and signature records increases and can create operational issues

10.2.1. Problems with large DNS response messages

- Fragmentation of IPv4 and IPv6 UDP messages

    Hurts performance

    Fragmented IPv6 packets (containing a fragment header) could be blocked on the Internet backbone

    Enables UDP fragmentation attacks against non-validating DNS resolver

    DNS amplification attacks

The goal should be that the DNSKEY RRSet during an KSK rollover (including perhaps an emergency KSK) stays below 1232 byte

10.3. KSK AND ZSK

- For organizational reasons, DNSSEC splits the chain of trust inside a DNS zone onto two different keys: the Key-Signing-Key (KSK) and the Zone-Signing-Key (ZSK)

10.3.1. KSK:

- The KSK generates only one signature: the signature over the DNSKEY record (RRset)

- The hash of the KSK is stored in the parent zone as the DS record. This hash is used to close the chain of trust from the parent zone to the DNSSEC signed zone. Each time the KSK in the DNSSEC signed zone is changed, the DS record in the parent zone must be replaced. The KSK has therefore a dependency on the parent zone.
- Because of this dependency with the parent zone, the KSK is usually a stable and strong key and not rolled often

10.3.2. ZSK:

- The Zone-Signing-Key does not have dependencies to external resources and can be rolled at any time
- Usually the ZSK is rolled often (and automatically), so the key does not need to be particularly strong

10.4. BIND 9: KSK/ZSK SIGNATURE OVER THE DNSKEY RECORD SET

- For historical reasons, BIND 9 generates two signatures over the DNSKEY record set:

  one signature generated with the KSK (required)

  one signature generated with every active ZSK (optional)

- This can generate larger than good DNSKEY record answer messages
- BIND 9 can be configured to only generate a signature using the KSK

```
options {
  [...]
  dnssec-dnskey-kskonly yes;
};
```

10.5. LIFETIMES OF DNSSEC KEYS

- DNSSEC keys have an organisational lifetime (there is no technical limit on the lifetime, unlike with X.509 TLS certificates)

  the administrator responsible for a DNSSEC-signed zone can decide when to change the DNSSEC keys

- Renewing the DNSSEC key material of a zone is called a *key rollover*
- Keys with weak algorithms or short key lengths should be changed (rolled) in shorter intervals

  1024bit RSASHA256 → 30 days (ZSK)

  1536bit RSASHA256 → 120 days (ZSK)

  2048bit RSASHA256 → 360 days (KSK)

  2560bit RSASHA256 → 720 days+ / 2 years+ (KSK)

10.6. DNSSEC SIGNER "BEST-PRACTICES"

- Zones with high security requirements should keep the DNSSEC keys "offline"

  this requires manual DNSSEC signing

- Keys can be stored securely in Hardware Security Modules (HSM)
- HSM selection criteria for DNSSEC signing

    Number of key storage slots

    Timely support for new operating system releases (Linux distribution)

    Timely support for new OpenSSL releases

    Stability of the HSM-Driver

- Zones with medium to low security requirements should use a *hidden-primary* DNSSEC signer configuration

    The DNSSEC signing authoritative server is not exposed to the Internet, it will not receive DNS queries

    DNS secondary authoritative servers in the Internet will receive the DNSSEC signed zone from the hidden DNSSEC signer through DNS zone transfer

    The DNSSEC key material is secured with operating system level permissions

    With full automation of DNSSEC key-rollovers, the DNS server administrators don't need access to the DNSSEC keys

    Login and access to the DNSSEC key files should be audited (Logging online and towards a remote log server)

- Zone content can be split across multiple zones or DNS server with DNS delegation

    Every zone can have a different DNSSEC security level

    Example: main zone `example.com` signed via a *hidden-signer* (keys not exposed to the Internet) has a sub-zone with e-mail address hashes (OPENPGPKEY and SMIMEA) in the zone `securemail.example.com`) which is secured with *NSEC3-NARROW* scheme and keys that are online on every authoritative server

CHAPTER 11. DNSSEC WITH DYNAMIC ZONES

- Since BIND 9.7.4 dynamic DNS-Zones (zones that are managed through dynamic DNS updates) can be DNSSEC signed
- Benefits of DNSSEC with dynamic zones:

  BIND 9 takes care of updating the signatures (RRSIG records) whenever they expire

  The DNSSEC keys can be automatically imported into the zone and also they can be automatically removed once they are no longer in use

  The lifetime of a DNSSEC key can be specified in the meta-data in the key files

  Using the meta-data the key rollover can be automated

  Changes to the zone are automatically DNSSEC signed

  The SOA serial number will be automatically incremented for each change

  The parser of the `nsupdate` tool used to send dynamic updates prevents syntax errors

  The zone journal collects all changes to the zone
- Drawbacks:

  The primary authoritative DNS server with the dynamic zone must have access to the private DNSSEC key material in order to create the signatures. The primary server should not be exposed to the Internet but should be a "Hidden Primary"

  The zone file may not be modified manually. The use of text editors and DNS management scripts on the zone file is not possible anymore

11.1. DNSSEC WITH A DYNAMIC ZONE

11.1.1. DNSSEC key creation with timing events

- Create a ZSK and KSK key pair for the zone

  If no algorithm is given to the `dnssec-keygen` command, the default is *RSAHSA1* which is not recommended anymore

  If no key sizes are given, the command creates (for RSA algorithms) a ZSK with 1024 bit and a KSK with 2048 bit

  The parameter `-K` specifies the directory where the keys are being created

```
      % dnssec-keygen -a RSASHA256 -b 1536 -K /etc/bind/keys dynamic.zXX.dane.onl
      % dnssec-keygen -a RSASHA256 -b 2048 -K /etc/bind/keys -f KSK dynamic.zXX.dane.onl
```

- Key files must be readable by the `named` process; adjust the ownership of the files accordingly

```
  % chown bind:bind /etc/bind/keys/K*
```

- When creating DNSSEC keys it is possible to store the lifecycle events of the keys in the key file as meta data

- Dates can be specified in the format YYYYMMDD or YYYYMMDDHHMMSS, or as an offset to the current time and day (now). The offset is specified with +/-, a number and the scale (y=Year, mo=Months, d=Days, h=Hours, mi=Minutes)

  Months always have 30 days

  Years always have 365 days

- These meta data timing events can be printed or changed on existing keys with the `dnssec-settime` command

  `-P` Time the key should be visible in the zone (publish)

  `-A` Time the key should be used to create signatures in the zone (active)

  `-I` Time the key should not longer be used to create signatures in the zone (in-active)

  `-D` Time when the key should be removed from the zone (delete)

  `-R` Time the key has been revoked

11.1.2. revoking DNSSEC keys

- Published keys can be revoked with the `dnssec-revoke` command. Example:

```
     % dnssec-revoke KzXX.dane.onl.+008+23689.private
     KzXX.dane.onl.+008+23817
+
     % more KzXX.dane.onl.+008+23817.key
     ; This is a revoked key-signing key, keyid 23817, for zXX.dane.onl.
     ; Revoke: 20160202164447 (Tue Feb  2 17:44:47 2016)
     zXX.dane.onl. IN DNSKEY 385 3 8 AwEAAeHhGKk8bOlK2sI8dysod64WOBpkudNx/SNNsAcy8PWddOGau8Iq
     F7a+YZH2JAOPFshfF9GLR3yt0kWTDjUOs0TCkyFoB4uYJftkeP5o/VO1
     BeDapl5O87Qij3sq+DC8AmPfxYIIT/Kl0BSl0bEhR0AxnGoEpPzsaoNH
     MSgkYp3wUZjNxZrXfOslekfN2VcCdwtzXfjW9FJxw61tg4bc2HydDUKw
     6YS8YntWcdkbDdTWHImcaBk2UqBcfzluL9BShedDZ7psnIqh9EmNu+BR
     jaMuE64xAbuk5py2cKKY3sg9LEpT5CLEuN0HSoH+iNY/E1QV1AHMGWlj
     pdnw9il5Wq0=
```

- A revoked KSK has the flag value *385*
- It is only possible to revoke KSK keys, ZSK keys should be rolled
- The key ID or key tag of the key changes when it is revoked

11.1.3. Zone definition for dynamic zone with DNSSEC

- Create a new zone file for the zone `dynamic.zXX.dane.onl` in the directory `/var/named`. It is important to use low Time-To-Live values in our lab zones (recommended: 60 seconds)

```
$ORIGIN dynamic.zXX.dane.onl.
$TTL 60
@                IN SOA authXX.dane.onl. hostmaster 1001 1h 30m 41d 60
                 IN NS  authXX.dane.onl.
```

- Create the zone definition in the BIND 9 configuration file `named.conf`

  `update-policy local;` permits dynamic updates with the BIND 9 session key from `/run/named/session.key`. This key is being used to authenticate against BIND when `nsupdate` is started with the parameter `-l`

`auto-dnssec maintain` enables the automatic import of DNSSEC keys and the automatic refresh of DNSSEC signatures

```
zone "dynamic.zXX.dane.onl" IN {
          type primary;
          file "dynamic.zXX.dane.onl";
          update-policy local;
          auto-dnssec maintain;
};
```

- Check the configuration and load the new configuration into BIND 9

```
      % named-checkconf -z
      % rndc reconfig
```

- From this time on the zone file should not be touched by a text editor!
- BIND 9 will sign the zone at the next DNSSEC-signing-interval (max. 60 minutes)
- The DNSSEC signing of the zone can be triggered with `rndc sign <zone-name>`. The command `rndc sign` does not return an error when the signing process fails. Always check the log files!

```
      % rndc sign dynamic.zXX.dane.onl
      % dig AXFR @localhost dynamic.zXX.dane.onl +multi
```

11.1.4. Adding a new record to a dynamic zone

- When a new DNS record is added to the zone, it will automatically be signed

```
   % nsupdate -l
   > update add test.dynamic.zXX.dane.onl. 60 IN TXT "a new record"
   > send
   $ dig @localhost test.dynamic.zXX.dane.onl TXT +dnssec +multi
   [..]
   ;; ANSWER SECTION:
   test.dynamic.zXX.dane.onl. 60 IN TXT "a new record"
   test.dynamic.zXX.dane.onl. 60 IN RRSIG TXT 8 4 60 (
          20210205142801 20210117081014 32184 dynamic.zXX.dane.onl.
          LbtggFHnWsLV7gEyI5PVdtDJ+LVihuLVHT3Ss1KtsB8a
          ...
          yVHm3Tb+qeY+Flj5073NfVPA+oUZa8O4DaPP )
```

11.1.5. Removing a DNS record from a dynamic zone

- When removing a record from a dynamic zone the record and its signature record will be removed, also the SOA serial number is incremented and a new signature for the SOA record is created

```
   % nsupdate -l
   > update del test.dynamic.zXX.dane.onl. IN TXT
   > send
```

11.1.6. DS records for dynamic DNSSEC zones

- The DS records for dynamic zones are created with the command `dnssec-dsfromkey`. This command takes the file containing the public part of the active KSK of the zone:

```
   % dnssec-dsfromkey Kdynamic.zXX.dane.onl.+008.+12345.key
```

```
    dynamic.zXX.dane.onl. IN DS 12345 8 1 B7BE432A2C4A25A0C9F1DA6DD4C289C99C25B2CC
    dynamic.zXX.dane.onl. IN DS 12345 8 2 DCA5AFBD131982707B55A19CD33048674ADE5FDD9
                          FFCB008A2F54744 B378B689
```

- By default, the command prints the SHA1- and SHA256-Hash versions of the DS record. This output can be sent to the operator of the parent zone.
- Some operators might require the *public* DNSKEY instead as they can generate the hashes themselves

11.2. EXERCISE: DNSSEC SIGNING A DYNAMIC DNS ZONE WITH BIND 9

- Create a new directory for the zone and change its ownership

```
% mkdir /etc/bind/myzones
% chown bind:bind /etc/bind/myzones
```

- Create a new zonefile as a child to the existing zone `zXX.dane.onl` with the name `dynamic.zXX.dane.onl`. It should contain one NS record and one SOA record (a bare minimum zone file). Save the zone file into `/etc/bind/myzones/dynamic.zXX.dane.onl`.

```
$TTL 30
@    SOA  authXX.dane.onl.  hostmaster 1001 1h 30m 41d 30s
@    NS   authXX.dane.onl.
```

- Ensure *named* can write to the zone file

```
% chown bind:bind /etc/bind/myzones/dynamic.zXX.dane.onl
```

- Add a new zone definition to the BIND 9 configuration file `/etc/named.conf`. The Key-Directory is relative to the BIND 9 home directory `/var/named`

```
zone "dynamic.zXX.dane.onl" {
  type master;
  key-directory "myzones/keys";
  update-policy local;
  auto-dnssec maintain;
  file "myzones/dynamic.zXX.dane.onl";
};
```

- Create the directory for the keys and adjust the permissions

```
% mkdir -p /etc/bind/myzones/keys
% chown bind:bind /etc/bind/myzones/keys
```

- Create a ZSK for the zone. Here we use the ECDSA algorithm with SHA256. With ECDSA, the key length is fixed.

```
% dnssec-keygen -a ECDSAP256SHA256 \
     -K /etc/bind/myzones/keys/ \
     dynamic.zXX.dane.onl
```

- Create the KSK for the zone. We also use ECDSA with SHA256:

```
% dnssec-keygen -a ECDSAP256SHA256 \
        -K /etc/bind/myzones/keys/ \
        -f KSK dynamic.zXX.dane.onl
```

- Adjust the permissions of the key files

```
% chown bind:bind /etc/bind/myzones/keys/K*
```

- Test the configuration and reload the BIND 9 server

```
% named-checkconf -z
% rndc reconfig
```

- Sign the zone with DNSSEC, check for error messages from BIND 9

```
% rndc sign dynamic.zXX.dane.onl
% journalctl -u named   # or check your logs
```

- Test: you should see the DNSSEC RRSIG record(s)

```
$ dig @127.0.0.1 dynamic.zXX.dane.onl SOA +dnssec +multi
```

- Add a NS record for the delegation to the **parent** zone

```
% echo "dynamic.zXX.dane.onl. IN NS authXX.dane.onl." >> \
        /etc/bind/zonefile.db
```

- Create the DS record and add it to the parent zone to close the chain of trust

```
% dnssec-dsfromkey \
   /etc/bind/myzones/keys/Kdynamic.zXX.dane.onl.+013+(KSK) \
      >> /etc/bind/zonefile.db
```

- Increment the SOA serial of the parent zone `zXX.dane.onl` and re-sign that parent zone (remember, this zone is manually signed)

```
% dnssec-signzone -o <zone-name> -k <KSK-private-file> \
          <zone-file> \
          <ZSK-private-file>
```

- Re-Load the parent zone

```
% rndc reload zXX.dane.onl
```

- You should now see a AD-Flag when querying data of the new dynamic zone from a DNSSEC validating resolver

```
$ dig @9.9.9.9 dynamic.zXX.dane.onl SOA +dnssec +multi
```

- Add a new IPv4 A record to the zone (for testing purposes)

```
% nsupdate -l
> ttl 30
> add www.dynamic.zXX.dane.onl. IN A 192.0.2.80
> send
> quit
```

- Query the new entry. Check for the AD-Flag.

```
$ dig www.dynamic.zXX.dane.onl +dnssec
```

- Change the IPv4 address of the A record (replace `xxx.xxx.xxx.xxx` with the public IPv4 address of your primary authoritative server)

```
% nsupdate -l
> ttl 30
> del www.dynamic.zXX.dane.onl IN A
> add www.dynamic.zXX.dane.onl IN A xxx.xxx.xxx.xxx
> send
> quit
```

- Test if the new entry can be queried and shows the AD-Flag.
- Delete the www-Entry from the zone

```
% nsupdate -l
> del www.dynamic.zXX.dane.onl IN A
> send
> quit
```

- Force the zone file to be written from memory into a file and display the content

```
% rndc sync dynamic.zXX.dane.onl
% less /etc/bind/myzones/dynamic.zXX.dane.onl
```

- All changes to the dynamic zone are written to the BIND 9 journal file for this zone. The journal file has the same name as the zone file, but with the extension `.jnl`
- The journal file is a binary file, this file can be converted to readable format with the command `named-journalprint`:

```
% named-journalprint /etc/bind/myzones/dynamic.zXX.dane.onl.jnl
```

CHAPTER 12. DNSSEC 'INLINE'-SIGNING

- Starting with BIND 9.9, a BIND 9 authoritative (primary or secondary) server can sign a DNS zone while

    Loading a zone from a file

    Loading via zone transfer from another authoritative DNS server

- BIND 9 will automatically update the DNSSEC signature records (RRSIG)

    If the zone is loaded from a file and the file contains a zone with a higher SOA serial than is present in memory

    On loading the zone via zone transfer (IXFR or AXFR)

- Benefits of *inline-signing*

    Administrative procedures do not need to change with the adoption of DNSSEC

    DNS management tools without support for DNSSEC can still be used

    Existing DNS infrastructure can be made DNSSEC enabled by using BIND 9 as a *bump-in-the-wire* DNSSEC signer, without the need to touch the authoritative server

    BIND 9 automates the key handling (loading the DNSSEC keys) and refreshes the RRSIG DNSSEC signature records

    The life cycle of DNSSEC keys can be configured used meta data inside the key files

    Using the key life cycle metadata, DNSSEC key rollover can be automated

- Downsides of *inline-signing*

    The SOA serial in the zone on the authoritative DNS servers will be *out-of-sync* (higher) than the number in the zone files

    The administrator still must take care that the SOA serial is incremented on every change to the zone file

12.1. DNSSEC WITH 'INLINE-SIGNING'

- To use 'inline-signing', BIND 9 needs to be able to find the DNSSEC key files. A key-directory should be specified globally (for all zones) or in each DNSSEC zone configuration

```
options {
    directory "/etc/bind";
    key-directory "/etc/bind/keys";
    dnssec-enable yes;
};
```

- The new configuration should be checked and then loaded into the BIND 9 DNS server

```
% named-checkconf -z
% rndc reconfig
```

- The next step is to create the ZSK and KSK for the zone

    the parameter `-K` specifies the directory in which the keys will be created

```
% dnssec-keygen -a RSASHA256 -b 1536 -K /etc/bind/keys inline.zXX.dane.onl
% dnssec-keygen -a RSASHA256 -b 2048 -K /etc/bind/keys -f KSK inline.zXX.dane.onl
```

- Inline signing is activated in the zone definition with `inline-signing yes;`

```
zone "inline.zXX.dane.onl" IN {
    type primary;
    file "inline.zXX.dane.onl";
    inline-signing yes;
    auto-dnssec maintain;
};
```

- Check the configuration and re-load BIND 9, then sign the zone

```
% named-checkconf -z
% rndc reconfig
% rndc sign inline.zXX.dane.onl
```

- As `rndc sign` does not report errors, the log files should always be checked for error messages (missing keys, permission issues)

```
     % tail /etc/bind/named.log
     31-Jan-2016 21:58:37.945 zone inline.zXX.dane.onl/IN (unsigned): loaded serial 1002
     31-Jan-2016 21:58:37.946 zone inline.zXX.dane.onl/IN (signed): loaded serial 1003 (DNSSEC
signed)
```

- The content of the signed zone will be written into a file with the same name as the original zone file but with the extension `.signed`. The file content is in binary "raw" format
- In addition to the signed file BIND 9 will create a journal file with the extension `.jnl` (for IXFR zone transfers) and a backup of the journal with the extension `.jbk`
- The original zone file will **not** be changed!
- The DNSSEC signing will take place in memory and the signed zone will be written to disk no later than 15 minutes after the signing has taken place
- With `rndc sync` the new signed zone data can be forced to disk immediately
- The signed zone file in "raw" format can be converted to readable text format with `named-compilezone`

```
% rndc sync inline.zXX.dane.onl
% named-compilezone -f RAW \
    -o inline.zXX.dane.onl.txt \
       inline.zXX.dane.onl inline.zXX.dane.onl.signed
```

- Changes to the zone file can now be done as normal to the unsigned zone file.

    After each change the SOA serial needs to be incremented (also as normal)

    When signing the zone, BIND 9 will also increment the SOA serial. For inline signed zones, it is normal that the SOA serial returned by the DNS server is higher than the SOA serial in the zone file

12.2. EXERCISE: DNSSEC WITH 'INLINE-SIGNING'

- Use your virtual lab machine, `authXX.dane.onl` (primary authoritative)
- Enable DNSSEC in the `named.conf` file (default in) and provide a directory for the DNSSEC keys

```
    options {
        directory "/etc/bind";
        key-directory "/etc/bind/keys";
    };
```

- Test the configuration file and reload the BIND 9 DNS server

```
    % named-checkconf -z
    % rndc reconfig
```

- Create the key directory `/etc/bind/keys`
- Create the DNSSEC keys (ZSK and KSK) for the zone

  The parameter `-K` specifies the directory in which the keys will be created

```
    % dnssec-keygen -a RSASHA256 -b 1536 -K /etc/bind/keys inline.zXX.dane.onl
    % dnssec-keygen -a RSASHA256 -b 2048 -K /etc/bind/keys -f KSK inline.zXX.dane.onl
```

- Ensure correct ownership of key files

```
    % chown -R bind:bind /etc/bind/keys
```

- Create a new zone file for the domain `inline.zXX.dane.onl` on your authoritative primary server. The zone should contain one SOA, 2 NS records (for the two authoritative servers), two IPv4 and IPv6 addresses and one TXT record. You can use the zone `zXX.dane.onl` as a template.
- Activate inline-signing for this zone

```
    zone "inline.zXX.dane.onl" IN {
        type master;
        file "inline.zXX.dane.onl";
        inline-signing yes;
        auto-dnssec maintain;
    };
```

- Test the configuration, reload the configuration and then sign the zone

```
    % named-checkconf -z
    % rndc reconfig
    % rndc sign inline.zXX.dane.onl
```

- Check the log files for issues (`rndc sign` does **not** report errors!)

```
    % tail /etc/bind/named.log
    31-Jan-2016 21:58:37.945 zone inline.zXX.dane.onl/IN (unsigned): loaded serial 1002
    31-Jan-2016 21:58:37.946 zone inline.zXX.dane.onl/IN (signed): loaded serial 1003 (DNSSEC
signed)
```

- The new zone file with the signed zone will be written in a binary format (`RAW` format).

  The new zone file has the suffix `.signed`

  In addition to that file, BIND 9 also creates a journal file with the suffix `.jnl` containing the changes (for IXFR style zone transfers) and a backup for the journal (suffix `.jbk`)
- The original zone file will **not** be touched!
- All changes due to inline-signing will be written to storage after approx. 15 minutes

This can be forced with the command `rndc sync`

- The command `named-compilezone` can be used to convert the `RAW` format zone file into a file with the RFC 1035 master file format

```
  % rndc sync inline.zXX.dane.onl
  % named-compilezone -f RAW \
      -o inline.zXX.dane.onl.txt \
         inline.zXX.dane.onl inline.zXX.dane.onl.signed
```

- All changes to the zone are now done to the text zone file as usual

  The SOA serial number must be incremented on each change

  The SOA serial in the file does not need to be higher than the SOA serial seen live on the DNS servers, it only needs to be higher than the last loaded SOA serial

- Add a record and bump the serial number in the zone file

```
  % $EDITOR inline.zXX.dane.onl
```

- Reload the zone and observe the logs and perform an SOA query for the zone to determine whether the SOA serial has changed

```
% rndc reload inline.zXX.dane.onl
$ dig @::1 inline.zXX.dane.onl SOA +multi
```

12.3. WHERE IS THE AD-FLAG?

- There is no AD Flag on queries for `inline.zXX.dane.onl`. There are no answers as well!

  Delegation NS records and the DS record is missing from the parent zone `zXX.dane.onl`

- Edit the file `zonefile.db` and add a delegation NS record for `inline.zXX.dane.onl` and the DS record for `inline.zXX.dane.onl`

```
inline.zXX.dane.onl.   IN NS authXX.dane.onl.
```

- Sign the parent zone `zXX.dane.onl` (manually signed) with `dnssec-signzone` (same command as many times before)

- Reload the new signed parent zone into the BIND 9 name-server, check for the `AD` flag on queries for `inline.zXX.dane.onl`

CHAPTER 13. NSEC VS. NSEC3

13.1. AUTHENTICATED DENIAL OF EXISTENCE - NSEC VS. NSEC3

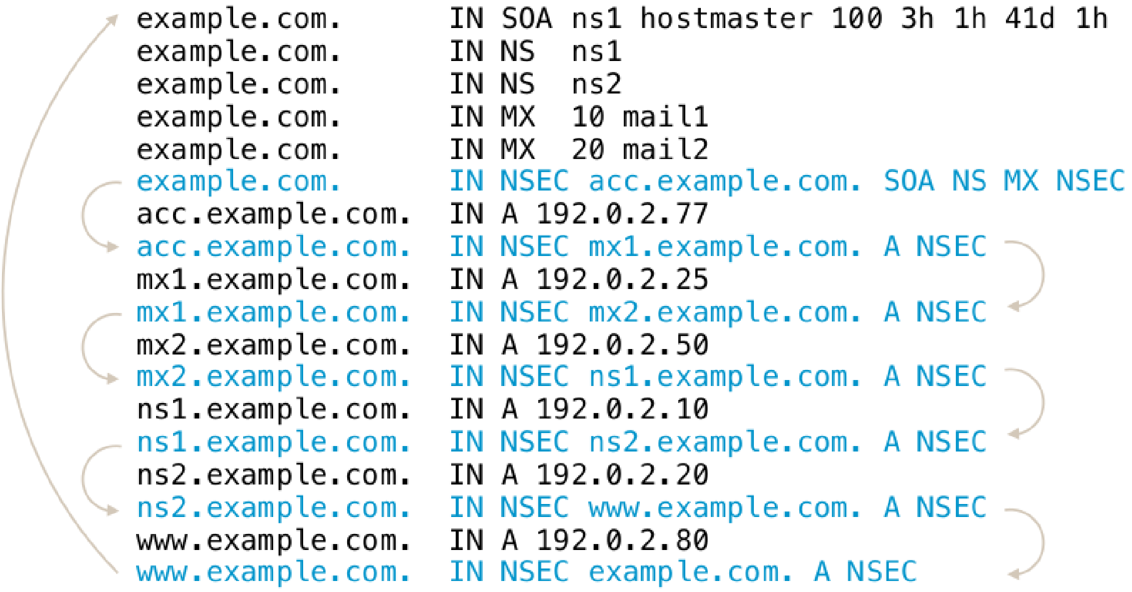
- The RRSIG records in DNSSEC secure the positive answers from DNS (answers to queries where DNS records exist)
- But also negative answers need to be protected

  without protection for negative answers, attackers could spoof negative answers to launch denial-of-service attacks. The attacker can make DNS clients believe that a DNS resource does not exist
- DNS knows two kinds of negative answers: NXDOMAIN and NODATA/NXRRSET

  NXDOMAIN = the requested domain name does not exist

  NODATA/NXRRSET = the requested domain name does exist, but the requested record type does not exist for this name
- DNS error messages such as SERVFAIL, FORMERR or REFUSED cannot be secured with DNSSEC (these errors indicate errors in the protocol or in the DNS server infrastructure). If the protocol is broken, DNSSEC can't work either.
- The solution: the NSEC records in the DNSSEC signed zone create a list of all existing data in the zone (domain names and records types for the domain names)

  When returning a negative answer, the DNS server returns the negative answer containing the NSEC record (plus a signature for the NSEC record) which proves that the requested data does not exist in DNS



```
example.com.        IN SOA ns1 hostmaster 100 3h 1h 41d 1h
example.com.        IN NS  ns1
example.com.        IN NS  ns2
example.com.        IN MX  10 mail1
example.com.        IN MX  20 mail2
example.com.        IN NSEC acc.example.com. SOA NS MX NSEC
acc.example.com.    IN A 192.0.2.77
acc.example.com.    IN NSEC mx1.example.com. A NSEC
mx1.example.com.    IN A 192.0.2.25
mx1.example.com.    IN NSEC mx2.example.com. A NSEC
mx2.example.com.    IN A 192.0.2.50
mx2.example.com.    IN NSEC ns1.example.com. A NSEC
ns1.example.com.    IN A 192.0.2.10
ns1.example.com.    IN NSEC ns2.example.com. A NSEC
ns2.example.com.    IN A 192.0.2.20
ns2.example.com.    IN NSEC www.example.com. A NSEC
www.example.com.    IN A 192.0.2.80
www.example.com.    IN NSEC example.com. A NSEC
```

13.2. ISSUES WITH NSEC

- The NSEC record is an elegant solution to the problem, but it has drawbacks:

It is now possible for outsiders to list the complete zone contents by following the NSEC record chain. This is called *zone walking*

For most zones this is not a big problem, as the DNS zone content is public anyway (SOA, NS records, WWW-A, MX records)

But it can be an issue for zones with sensitive content (email addresses, hostnames of critical infrastructure, new product names that should no go public yet)

Operators of TLDs zones stay away from DNSSEC with NSEC, as it allows outsiders to record all changes to the TLD zone (new delegations and delegation removals)

- Example of zone walking:

```
$ ldns-walk isc.org
```

13.3. THE NSEC3 RECORD

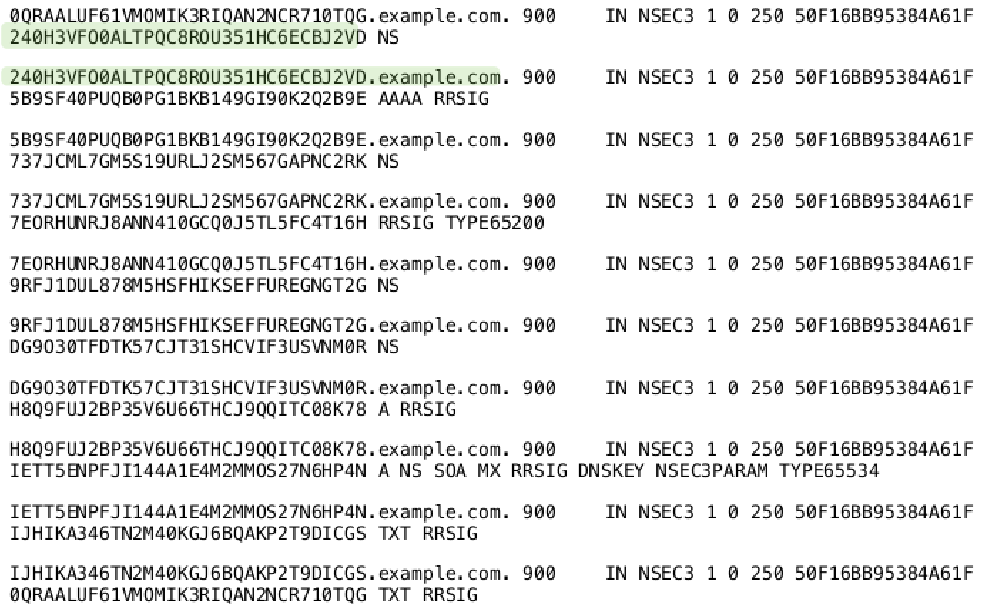
- RFC 5155 [https://tools.ietf.org/html/rfc5155] (2008) defines the NSEC3 record as an alternative to NSEC

All modern DNS servers support NSEC3

The NSEC3 record works similarly to NSEC, with the difference that the link between the owner names is done using SHA1 hashes of the domain names instead of the clear text names

NSEC3 makes it harder, but not impossible, to list the zone content

13.4. NSEC3 CHAIN



```
0QRAALUF61VMOMIK3RIQAN2NCR710TQG.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
240H3VFO0ALTPQC8ROU351HC6ECBJ2VD NS

240H3VFO0ALTPQC8ROU351HC6ECBJ2VD.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
5B9SF40PUQB0PG1BKB149GI90K2Q2B9E AAAA RRSIG

5B9SF40PUQB0PG1BKB149GI90K2Q2B9E.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
737JCML7GM5S19URLJ2SM567GAPNC2RK NS

737JCML7GM5S19URLJ2SM567GAPNC2RK.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
7EORHUNRJ8ANN410GCQ0J5TL5FC4T16H RRSIG TYPE65200

7EORHUNRJ8ANN410GCQ0J5TL5FC4T16H.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
9RFJ1DUL878M5HSFHIKSEFFUREGNGT2G NS

9RFJ1DUL878M5HSFHIKSEFFUREGNGT2G.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
DG9O30TFDTK57CJT31SHCVIF3USVNM0R NS

DG9O30TFDTK57CJT31SHCVIF3USVNM0R.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
H8Q9FUJ2BP35V6U66THCJ9QQITC08K78 A RRSIG

H8Q9FUJ2BP35V6U66THCJ9QQITC08K78.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
IETT5ENPFJI144A1E4M2MMOS27N6HP4N A NS SOA MX RRSIG DNSKEY NSEC3PARAM TYPE65534

IETT5ENPFJI144A1E4M2MMOS27N6HP4N.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
IJHIKA346TN2M40KGJ6BQAKP2T9DICGS TXT RRSIG

IJHIKA346TN2M40KGJ6BQAKP2T9DICGS.example.com. 900    IN NSEC3 1 0 250 50F16BB95384A61F
0QRAALUF61VMOMIK3RIQAN2NCR710TQG TXT RRSIG
```

13.5. NSEC3-PARAMETER

- The NSEC3 record contains a number of parameters used for the NSEC3 operation

The hashing algorithm used (currently only SHA1 is defined)

Flags: allows for DNSSEC opt-out in delegations.

Number of hash iterations

A salt value

13.6. NSEC3-PARAMETER RECORD

- Every zone with NSEC3 records contains an `NSEC3PARAM` record. This record holds information needed by authoritative DNS servers to generate NSEC3 records for negative answers
- Example: (SHA1, no flags, 20 iterations, salt value "ABBACAFE")

```
nsec3.dnslab.org.    0   IN  NSEC3PARAM 1 0 20 ABBACAFE
```

13.7. NSEC3 ITERATIONS AND SALT

- The idea of the hash iterations in the NSEC3PARAM record was to allow the operator of the zone to fine tune the work required for calculating the hash

  A higher number of iterations should make it harder for attackers to brute force break an NSEC3 domain name

  A higher number also creates more CPU load for DNS resolvers that validate the NSEC3 records, making DNS name resolution slower

  The iteration parameter of an NSEC3 signed zone should be re-evaluated from time to time (yearly) to adjust the value for the technical progress

- The salt should make it impossible for an attacker to pre-calculate rainbow tables [http://en.wikipedia.org/wiki/Rainbow_table] for the zone
- The salt is a hexadecimal number, every hex digit contains 4 bit of information

13.8. PROBLEMS WITH NSEC3

- The iterations had more an negative impact on the CPU load of the authoritative DNS servers than preventing attacks
- The salt is not really needed, as each zone naturally has unique hashes so that a rainbow table for multiple zones is not possible

13.9. RFC 9276 - GUIDANCE FOR NSEC3 PARAMETER SETTINGS

- RFC 9276 - Guidance for NSEC3 Parameter Settings [https://www.rfc-editor.org/rfc/rfc9276.html] (Best current practice) gives updates guidelines for NSEC and NSEC3 deployments
- The iterations value should be set to `0` (meaning 1 iteration of SHA1 hashing)

  DNSSEC responses containing NSEC3 records with iteration counts greater than 150 are now treated as insecure by major DNS resolvers!

- As NSEC3 zones are inherently salted, the salt parameter should be set to –
- The current recommendation for NSEC3PARAM is `1 0 0 -` (SHA1 Hash, no flags, 1 iteration, no salt)

13.10. NSEC3 NEEDED?

- Most DNS zone can work with NSEC instead of NSEC3

    Don't store names in DNS that should be *secret* (internal names, product names etc)

    DNS is a service to *publish* data, not for *hiding* data

13.11. NSEC3 NARROW-MODE

- RFC 7129 [https://tools.ietf.org/html/rfc7129] (also RFC 4470 [https://www.ietf.org/rfc/rfc4470.txt] and RFC 4471 [https://www.ietf.org/rfc/rfc4471.txt]) describe a special variant of NSEC3 usage, called the *narrow mode*

    With *narrow mode*, the NSEC3 records are not pre-calculated when the zone is signed, but they are created *on the fly* whenever a negative response is needed

        To be able to calculate the signatures for such answers, **every** authoritative DNS server for the zone must have access to the private DNSSEC keys (at least the ZSK)!

- With NSEC3 *narrow mode*, the DNS server does not return an NSEC3 chain of existing records, but a synthetic NSEC3 record that proves that the requested name does not exist

    This prevents zone walking, as the number of possible NSEC3 records returned is near infinite. It will exhaust the memory of the attacker that will try to store this NSEC3 chain

- Known implementations:

    Phreebird [https://dankaminsky.com/phreebird/] (Dan Kaminski, Proof-of-Concept, not maintained)

    PowerDNS Authoritative [https://doc.powerdns.com/authoritative/dnssec/modes-of-operation.html]

    Cloudflare CDN [https://blog.cloudflare.com/black-lies/] NSEC3 Narrow-Mode (closed source implementation)

13.12. EXERCISE: SIGN A STATIC ZONE WITH NSEC3 INSTEAD OF NSEC

- Sign a zone with NSEC3 (`salt` must be a hexadecimal number with even count of octets (8,10,12 ...))
- See *Take your DNSSEC with a grain of salt* for additional information on iterations and salt

```
% dnssec-signzone -3 <salt> -H <iterations> -o zXX.dane.onl \
       -k <KSK-private-file> zXX.dane.onl \
       <ZSK-private-file>
```

13.13. USE OF NSEC3 WITH INLINE-SIGNING OR DYNAMIC SIGNING

- To use NSEC3 instead of NSEC for DNSSEC signing with a dynamic DNS zone an NSEC3PARAM record must exist in the zone before the zone is signed. For inline-signing use rndc

13.14. EXAMPLE FOR INLINE-SIGNING:

- Edit the zone file

```
   % $EDITOR /etc/bind/inline.zXX.dane.onl
```

- Add the NSEC3PARAM record

```
   inline.zXX.dane.onl. 0  IN NSEC3PARAM 1 0 0 -
```

- reload and resign the zone

```
   % rndc reload
   % rndc sign inline.zXX.dane.onl
```

13.15. EXAMPLE FOR DYNAMIC ZONE SIGNING

- Add a NSEC3PARAM record to the zone

```
% nsupdate -l
> add dynamic.zXX.dane.onl. 0  IN NSEC3PARAM 1 0 0 -
> send
> quit
```

- Create the DNSSEC keys and sign the zone

```
 % rndc sign dynamic.zXX.dane.onl
```

13.16. SWITCHING A DYNAMIC ZONE FROM NSEC TO NSEC3

- An already signed dynamic zone can be switched to NSEC3 by using the `rndc sign` command

```
% rndc signing -nsec3param 1 0 0 - inline.zXX.dane.onl
```

- It is possible to switch back to NSEC with a similar command

```
% rndc signing -nsec3param none inline.zXX.dane.onl
```

- There are no timing issues when switching between NSEC and NSEC3

CHAPTER 14. SECONDARY DNS SERVER

14.1. EXERCISE: A SECONDARY SERVER FOR THE DNSSEC SIGNED ZONE

- Work on your secondary server `secXX.dane.onl`
- Enable and start BIND 9
- Remove the Debian 12 BIND 9 configuration and start a new BIND 9 configuration in `/etc/bind/named.conf` from scratch

```
% cd /etc/bind
% rm named*
% $EDITOR named.conf
```

- Disable recursion in the BIND 9 configuration and add a sensible logging block to the BIND 9 configuration. (Here's a template you can use [https://dnslab.org/dns/logging.conf].)
- Test the configuration and reload the BIND 9 service
- Create a new directory `/etc/bind/secondaries`

```
% mkdir /etc/bind/secondaries
```

- Add a configuration for a secondary zone into the BIND 9 configuration file `/etc/bind/named.conf` for your DNSSEC signed zone `zXX.dane.onl`. Use the IPv4 and IPv6 addresses of your primary server in the `primaries` statement:

```
[...]
zone "zXX.dane.onl" IN {
  type secondary;
  primaries { xxx.xxx.xxx.xxx; xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx; };
  file "secondaries/zXX.dane.onl";
};
```

- Test the configuration

```
% named-checkconf -z
```

- Re-load the BIND 9 configuration

```
% rndc reconfig
```

- Is the file for the secondary zone being created? If not, why not? What might be the problem?
- Check the logs

```
% journalctl -u named   # or check your logs
```

- The folder `secondaries` must belong to the user `bind`. Without that, the BIND 9 process is not permitted to write into that folder:

```
% chown bind: /etc/bind/secondaries
```

- Check that the secondary server is now authoritative for the DNSSEC signed zone (AA-Flag!)

```
$ dig @127.0.0.1 zXX.dane.onl
```

- Check that all authoritative DNS server for the zone are reporting the same SOA serial number:

```
$ dig zXX.dane.onl +nssearch
```

- Do all servers show up? If not, what might be wrong?

14.2. EXERCISE: ADJUST THE PRIMARY ZONE TO CONTAIN THE SECONDARY SERVER

- Your DNSSEC signed zone is now also available on the secondary. For DNS resolver to find and use the new secondary:

  the new server must be added as a NS record into the DNSSEC signed zone

  the new server must be added as a NS record into the parent zone of the signed zone (delegation) - this has been done by the trainer (the operator of the parent zone)

- Add a new NS record for the name of your secondary server into the zonefile (`zonefile.db`) of the zone on your primary DNS server:

```
zXX.dane.onl. 60 IN NS authXX.dane.onl.
zXX.dane.onl. 60 IN NS secXX.dane.onl.
```

- Increment the SOA serial number, re-sign the zone (manual signing with `dnssec-signzone`, check for errors and then reload the zone into the BIND 9 server (primary)
- Check that both servers now appear in the `+nssearch` feature of `dig` with the same SOA serial number

```
$ dig zXX.dane.onl +nssearch
```

- Make a small change to the zone file on the primary server (for example a new TXT record)

```
text.zXX.dane.onl. 30 IN TXT "Hello from a DNSSEC training"
```

- Increment SOA serial, check zonefile, re-sign the zone, re-load BIND 9

```
% $EDITOR /etc/bind/zonefile.db
% dnssec-signzone -o zXX.dane.onl -k <KSK> <zone-file> <ZSK>
% named-checkconf -z
% rndc reconfig
```

- Check that both servers now carry the new TXT record

```
$ dig @authXX.dane.onl text.zXX.dane.onl TXT
$ dig @secXX.dane.onl  text.zXX.dane.onl TXT
```

CHAPTER 15. DNSSEC KEY-ROLLOVER

15.1. WHY DNSSEC ROLLOVER

- The DNSSEC key material is public

  Including the signatures and the matching clear text (DNS record sets)

- The private key can get in un-authorized hands

  By accident

  By attacks on the signing server/HSM

  When using *online-signing*, the private keys must be live on the signing DNS server

- The key length or the algorithm used is not considered secure for a longer amount of time (for example 1024bit RSA keys)

15.2. THE CHALLENGES

- The DNS is not consistent

  DNS zone data can differ for some time between authoritative server of the same zone (delay in zone transfer)

  DNS data is cached in DNS resolvers, operating systems, and applications

- During a DNSSEC key-rollover, the chain of trust must be un-broken at all times from all vantage points of the Internet

15.3. DNSSEC KEYROLLOVER DOCUMENTATION

- **RFC 6781** - "DNSSEC Operational Practices, Version 2"
- **RFC 7583** - "DNSSEC Key Rollover Timing Considerations"

15.4. KEY ROLLOVERS, WHEN AND HOW OFTEN?

- DNSSEC keys do not have a technical lifetime, they don’t *expire*
- The *operational* life time of DNSSEC keys is decided by the responsible administrator(s) and can be changed at any time
- The DNSSEC community has different views on KSK key rollovers

  Often and regularly

  Often but irregular (to not give attackers information when the system might be more vulnerable due to the key rollover)

  Only if there is evidence that the key has been compromised or stolen

15.5. ZSK ROLLOVER

- The Zone-Signing-Key has no dependencies to external resources (such as the parent zone)
- A ZSK Rollover can be started at any time
- The 'pre-publication' rollover scheme is used for ZSK rollover

15.5.1. ZSK - pre-publication - Step 1

- Create a new ZSK key pair
- Publish the public part of the new key (DNSKEY) of the ZSK in the zone
- The current/old ZSK is kept in the zone
- The zone is signed with the current/old (not the new) ZSK and KSK

15.5.2. ZSK - pre-publication - Step 2

- Wait for the zone with the new ZSK be visible on all authoritative DNS servers of the zone (zone transfer)
- Wait for the TTL of the DNSKEY RRset (+ some buffer for security)
- Now we can be sure the new ZSK DNSKEY record is in all caches (DNS resolvers, operating systems, applications)

15.5.3. ZSK - pre-publication - Step 3

- Sign the zone with the new ZSK
- The new ZSK is now *active*, the old ZSK is now *retired*
- The old ZSK will be kept in the zone for now (it is needed to validate old signatures that still exist in the caches in the network)

15.5.4. ZSK - pre-publication - Step 4

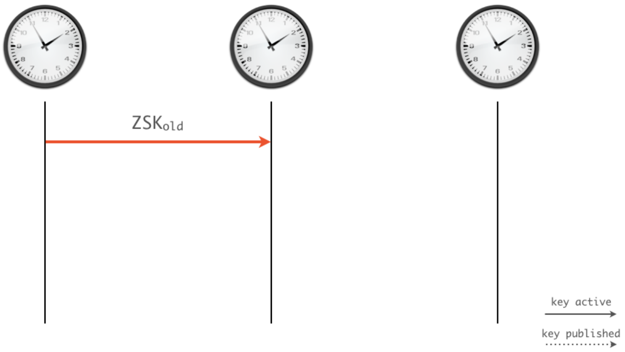
- Wait for the new zone version with the signatures from the new ZSK to be visible on all authoritative DNS servers of the zone (zone transfer)
- Wait for the largest TTL in the zone (plus some buffer)
- Now the signatures created by the old ZSK are expired from the caches, and the new signatures are available

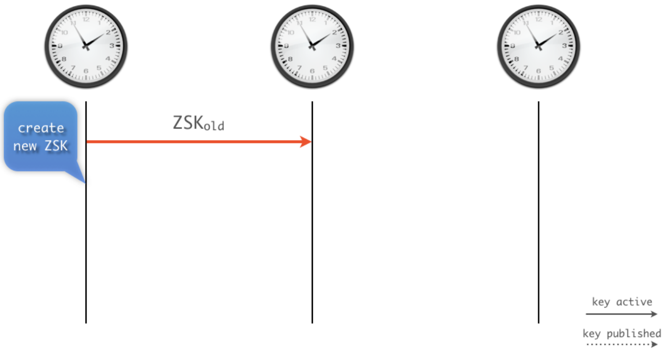
15.5.5. ZSK - pre-publication - Step 5

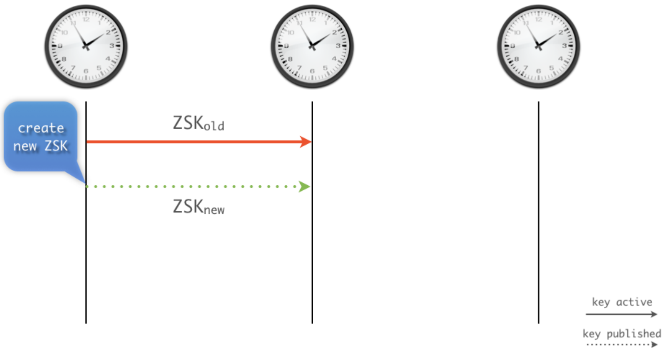
- Remove the old ZSK from the DNSKEY record set of the zone
- Continue signing the zone with the new ZSK

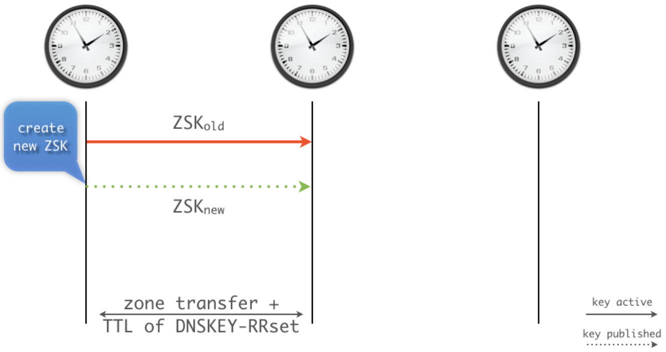
15.5.6. ZSK - pre-publication in pictures

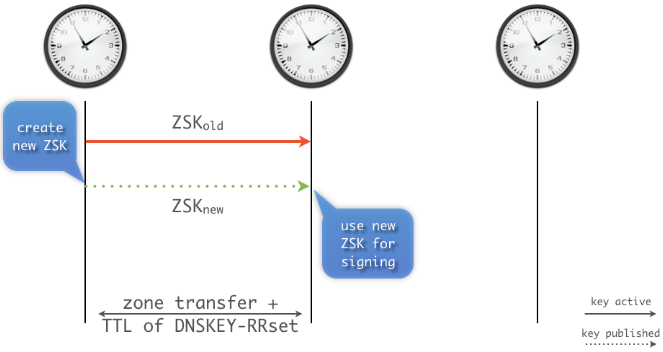
```
   [sidebar]
image::./img/zsk-roll-00.png[scaledwidth=65%]
```











create new ZSK

ZSK_{old}

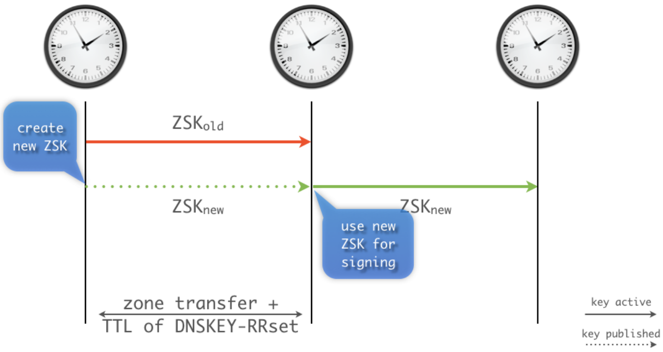
ZSK_{new}

use new ZSK for signing

zone transfer + TTL of DNSKEY-RRset

key active

key published



create new ZSK

ZSK_{old}

ZSK_{new}

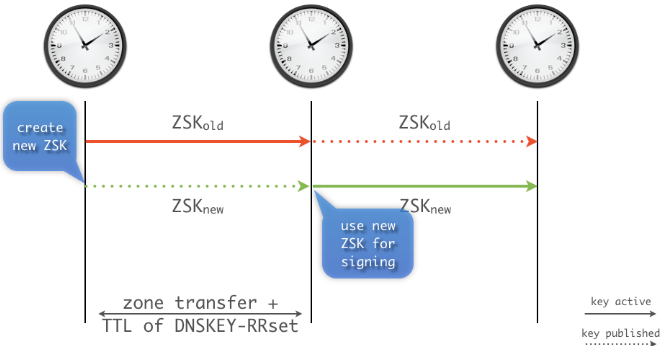
ZSK_{new}

use new ZSK for signing

zone transfer + TTL of DNSKEY-RRset

key active

key published



create new ZSK

ZSK_{old}

ZSK_{old}

ZSK_{new}

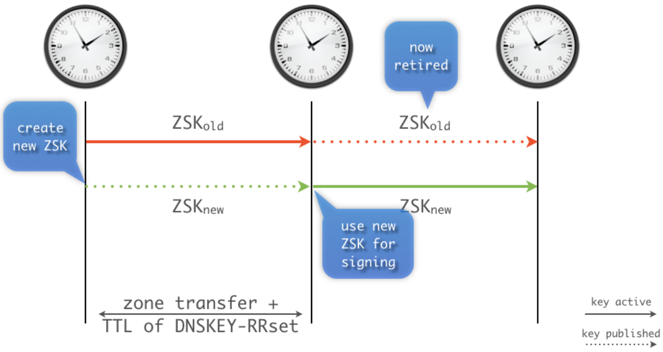
ZSK_{new}

use new ZSK for signing

zone transfer + TTL of DNSKEY-RRset

key active

key published



now retired

create new ZSK

ZSK_{old}

ZSK_{old}

ZSK_{new}

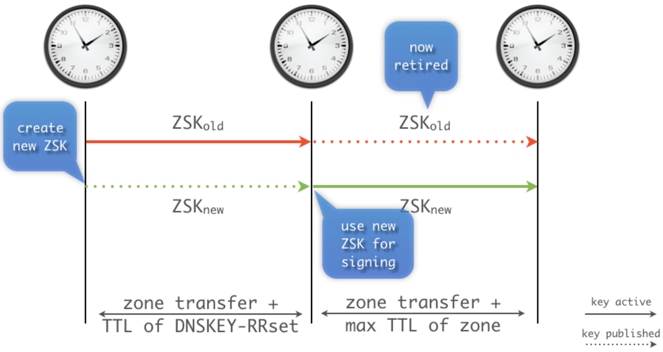
ZSK_{new}

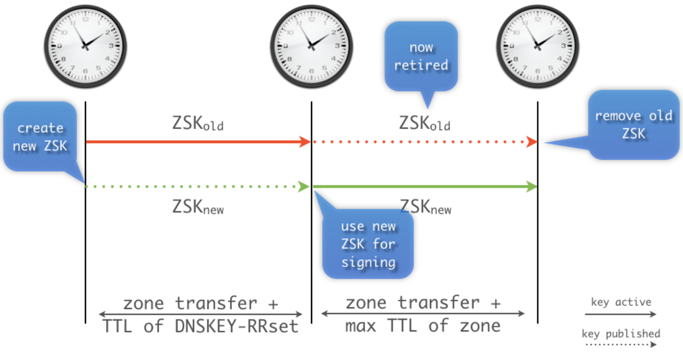
use new ZSK for signing

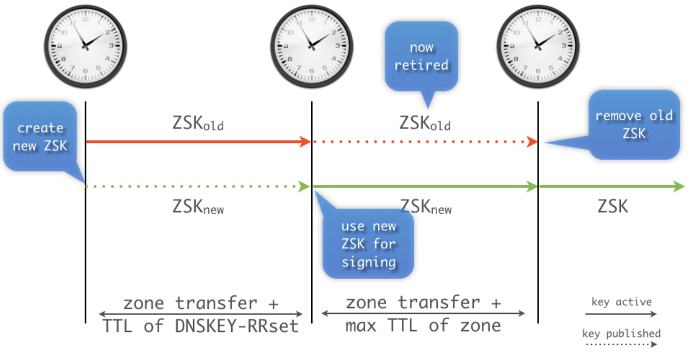
zone transfer + TTL of DNSKEY-RRset

key active

key published







15.6. EXERCISE: ZSK ROLLOVER

- In this exercise we will perform a ZSK-Rollover on the manually-signed zone `zXX.dane.onl`. This will be a pre-publish roll-over.
- Step 1: create a new ZSK key-pair and note down it's Key-ID

```
% dnssec-keygen -a RSASHA256 -b 1024 zXX.dane.onl
```

- Publish the public part of the new key in the zone. Make sure to use `>>` when using shell redirection.

```
% cat KzXX.dane.onl.+008+<ID-of-the-new-ZSK>.key >> /etc/bind/zonefile.db
```

- Increment the SOA serial of the zone (or use the *Unixtime* as the SOA-serial when signing the zone)
- Sign the zone with the **old/current** ZSK (to publish the new ZSK)

```
% dnssec-signzone -N unixtime -o zXX.dane.onl \
   -k <KSK-private-file> \
   zonefile.db  \
   <ZSK-private-file>
```

- Load the signed zone in the BIND DNS server and check that all secondaries have loaded the new zone (check the SOA serial number)

```
% rndc reload
$ dig zXX.dane.onl +nssearch
```

- Wait for the TTL of the DNSKEY record-set (60 seconds in this case)
- Increment the SOA serial of the zone (or use the Unixtime), then sign the zone (without any other change) with the new ZSK

```
% dnssec-signzone -N unixtime -o zXX.dane.onl \
    -k <KSK-private-file> \
    zonefile.db  \
<NEW-ZSK-private-file>
```

- Load the signed zone into the BIND 9 DNS server and check that all secondaries have the new zone loaded (check SOA serial)

```
% rndc reload
$ dig zXX.dane.onl +nssearch
```

- Wait for the largest TTL used in the zone (should be 60 seconds in our case)
- Remove the old ZSK from the zone file `/etc/bind/zonefile.db`, increment the SOA serial and sign the zone with the new ZSK:

```
% dnssec-signzone -N unixtime -o zXX.dane.onl \
      -k <KSK-private-file> \
      zonefile.db  \
  <NEW-ZSK-private-file>
```

- Load the signed zone into the BIND 9 DNS server and check that all secondaries have the new zone loaded (check SOA serial)

```
% rndc reload
$ dig zXX.dane.onl +nssearch
```

- Check that the zone still is being DNSSEC validated (AD-Flag!)

```
$ dig zXX.dane.onl SOA +dnssec +multi
```

- This is the end of the ZSK rollover

15.7. KSK ROLLOVER

- The KSK has a dependency on its DS record in the parent zone
- For the KSK rollover we will use the 'double-signing' rollover scheme (the DNSKEY record set will get two signatures, from both, old and new, KSK)

15.7.1. KSK - double-signing - Step 1

- Create a new KSK key pair
- Publish the DNSKEY record of the new key in the zone
- Sign the DNSKEY RRset in the zone with both KSKs (old and new)

15.7.2. KSK - double-signing - Step 2

- Wait until the new zone content with the new KSK is visible on all authoritative DNS server of the zone
- Wait for the TTL of the DNSKEY RRSet (+ some buffer)
- The new KSK is now visible to all DNS clients (through DNS resolver caches)

15.7.3. KSK - double-signing - Step 3

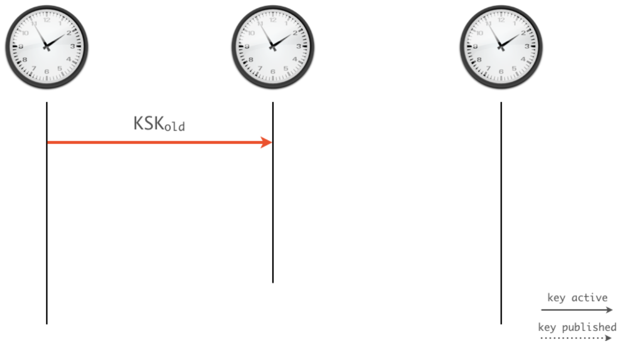
- Send the new DS record to the operator of the parent DNS zone (usually through an API or through an Web-Interface)
- Wait for the DS record to be updated in the parent zone
- Wait for the TTL of the DS record in the parent zone (+ some buffer)
- The new DS record is now visible for all DNS clients

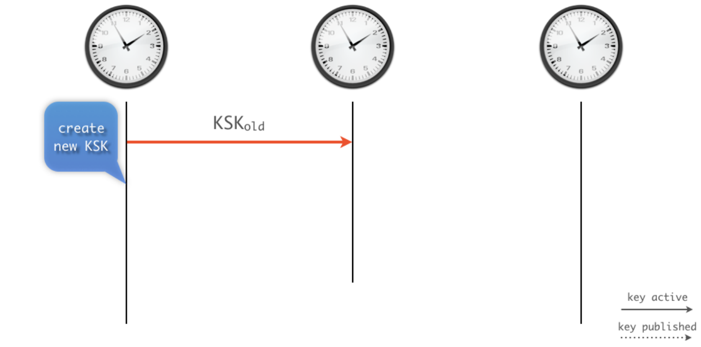
15.7.4. KSK - double-signing - Step 4

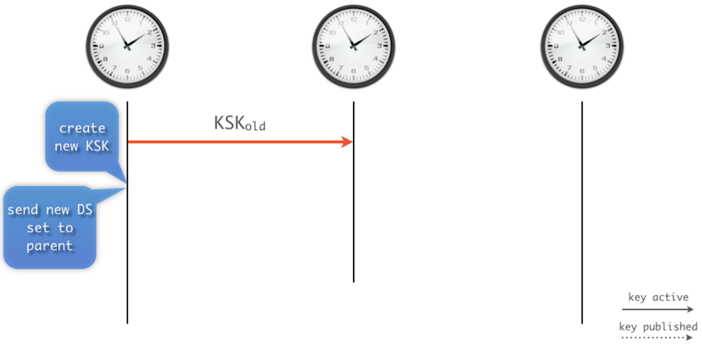
- Remove the old KSK from the DNSKEY record set of the zone
- Sign the zone with only the new KSK
- Wait and remove the old DS from the parent zone

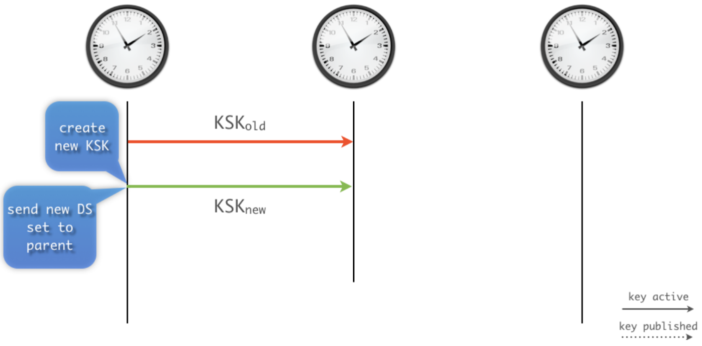
15.7.5. KSK - double-signing in pictures

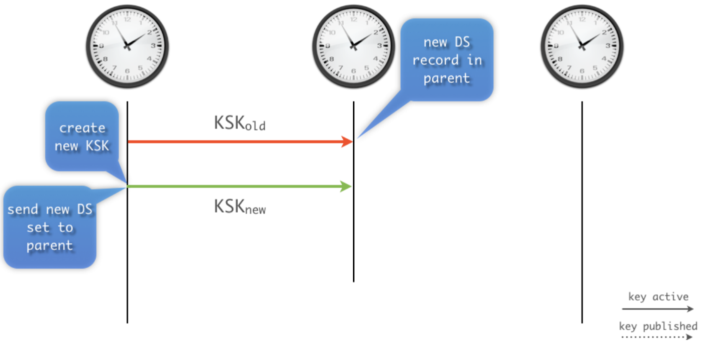
```
    [sidebar]
image::./img/ksk-roll-00.png[scaledwidth=65%]
```

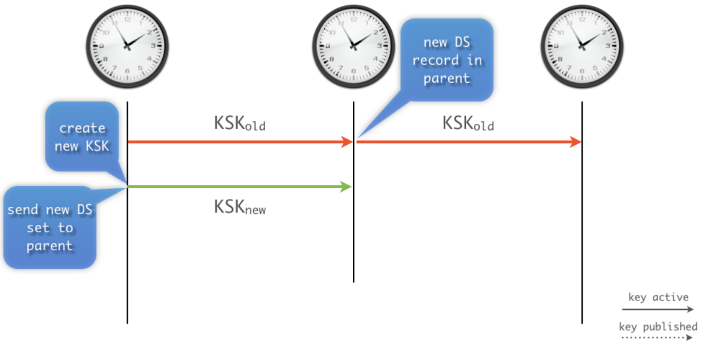


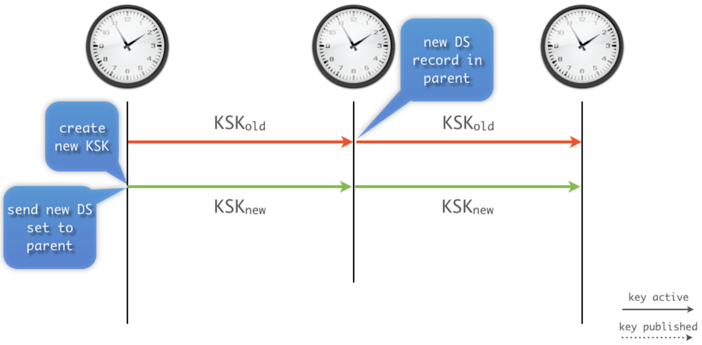

create new KSK
KSK$_{old}$
key active
key published

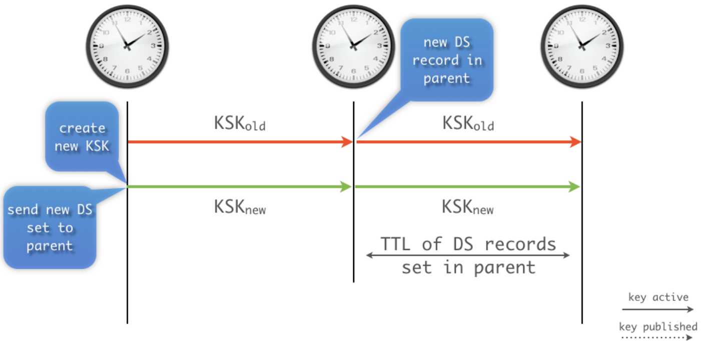

create new KSK
send new DS set to parent
KSK$_{old}$
key active
key published

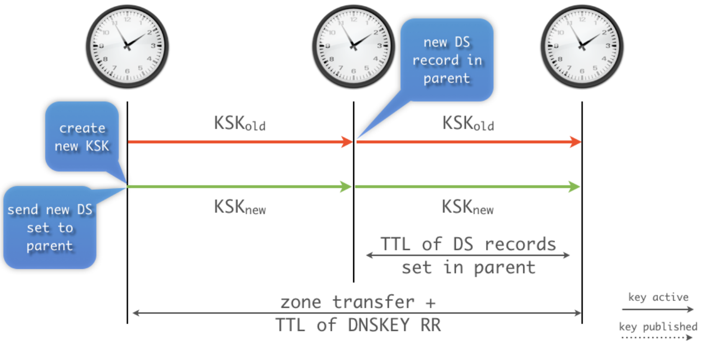

create new KSK
send new DS set to parent
KSK$_{old}$
KSK$_{new}$
key active
key published

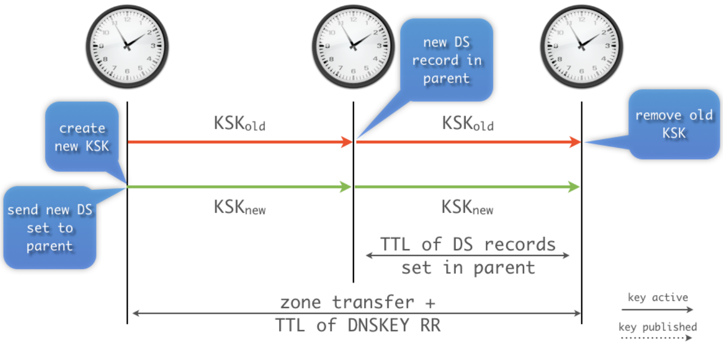

new DS record in parent
create new KSK
send new DS set to parent
KSK$_{old}$
KSK$_{new}$
key active
key published

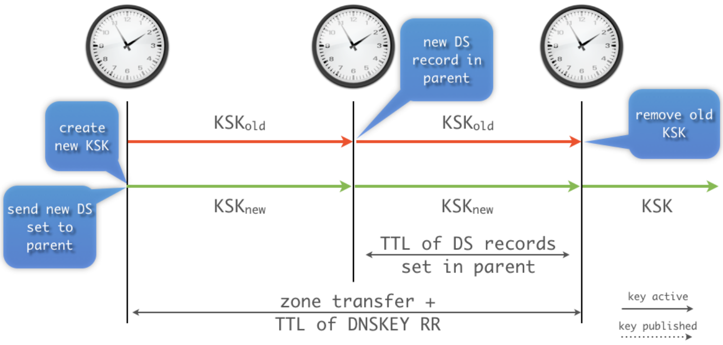

new DS record in parent
create new KSK
send new DS set to parent
KSK_{old}
KSK_{old}
KSK_{new}
key active
key published


new DS record in parent
create new KSK
send new DS set to parent
KSK_{old}
KSK_{old}
KSK_{new}
KSK_{new}
key active
key published


new DS record in parent
create new KSK
send new DS set to parent
KSK_{old}
KSK_{old}
KSK_{new}
KSK_{new}
TTL of DS records set in parent
key active
key published


new DS record in parent
create new KSK
send new DS set to parent
KSK_{old}
KSK_{old}
KSK_{new}
KSK_{new}
TTL of DS records set in parent
zone transfer + TTL of DNSKEY RR
key active
key published





15.8. EXERCISE: KSK ROLLOVER

- Use your virtual lab machine, `authXX.dane.onl` (primary authoritative)
- Create a new KSK key pair

```
% dnssec-keygen -a RSASHA256 -b 2048 -f KSK zXX.dane.onl
```

- Publish the new DNSKEY record in the zone, increment SOA-serial (or use Unixtime), sign the zone with both KSK keys (old and new)

```
% cat KzXX.dane.onl.+008+<ID-of-new-KSK>.key >> /etc/bind/zonefile.db
% dnssec-signzone -N unixtime -o zXX.dane.onl \
        -k <OLD-KSK-private-file> \
        -k <NEW-KSK-private-file> \
        zonefile.db \
        <Current-ZSK-private-file>
```

- Load zone into the BIND 9 server and verify that the zone still validates (AD-Flag!)

```
% rndc reload
$ dig zXX.dane.onl SOA +dnssec +multi
```

- Submit the new DNSKEY record to the parent zone

```
% scp <NEW-KSK-public-key-file> user@auth15.dane.onl:.
```

- Wait for the new DS record to be in the parent zone

```
$ dig zXX.dane.onl DS
```

- Wait for the TTL of the DS record in the parent zone and the TTL of the DNSKEY record in the zone (whichever is larger)
- Remove the old KSK DNSKEY record from the zone file, increment the SOA serial (or use the unixtime feature) and sign the zone with only the new KSK and the current ZSK

```
  % dnssec-signzone -N unixtime -o zXX.dane.onl \
              -k <NEW-KSK-private-file> \
              zonefile.db \
              <Current-ZSK-private-file>
```

- Load the signed zone into the BIND 9 server

```
% rndc reload
```

- Check that the zone still validates (AD-Flag!)

```
% dig zXX.dane.onl SOA +dnssec +multi
```

CHAPTER 16. DNSSEC KEY AND SIGNING POLICY

- introduces a new `dnssec-policy` feature as a further step in automation above and beyond what `auto-dnssec maintain` is capable of doing.
- Where *inline* or *dynamic* signing expected keys to have been created with `dnssec-keygen`, this signing method automates the task
- Configuring a zone to use KASP (*Key And Signing Policy*) can be as easy as

```
zone "example.net" {
    type primary;
    dnssec-policy default;
      ...
};
```

- Benefits

  More intuitive and a higher level of automation

  Several vendors use KASP (Knot, OpenDNSSEC)

  Robust

  > No need to rely on metadata added by humans
  >
  > Use key timing state machine

16.1. DEFAULT POLICY

- Single CSK

  ECDSAP256SHA256 (algo 13) with unlimited lifetime

  RRSIG validity 14 days, refreshed 5 days before expiration
- NSEC
- Key timings:

  DNSKEY TTL: 3600, max zone TTL: 86400 (1 day)

  Key publish and retire safety times: 3600 (1 hour)

  Propagation delay: 300 (5 minutes)
- Parent timings

  DS TTL: 86400 (1 day)

  Propagation delay: 3600 (1 hour)
- Prevent policy changes on upgrade by using an explicitly defined dnssec-policy, rather than `default`

16.2. WRITING CUSTOM DNSSEC POLICIES

```
zone "example.net" {
    dnssec-policy "one";
};
```

```
dnssec-policy "one" {
    keys {
        ksk lifetime 365d algorithm rsasha256 2048;
        zsk lifetime 60d  algorithm rsasha256 1024;
        csk lifetime P6M2T12H3M15S algorithm 13;
    };
};
```

- `keys` specify *roles* instead of specific keys
- `lifetime` specifies duration or `unlimited`
- `algorithm` uses mnemonic or number

16.3. ADDITIONAL CONFIGURATION OF CUSTOM POLICY

```
dnssec-policy "one" {
    keys {
        ksk lifetime 365d algorithm rsasha256 4096;
        zsk lifetime 60d  algorithm rsasha256 1024;
    };
    dnskey-ttl 600;
    publish-safety PT2H;
    signatures-refresh 7d;

    nsec3param iterations 0 optout no salt-length 0;
};
```

16.4. KEY-AND-SIGNING-POLICY (KASP) SYNTAX

```
dnssec-policy <string> {
    dnskey-ttl <duration>;
    keys { ( csk | ksk | zsk ) [ ( key-directory ) ]
        lifetime <duration_or_unlimited> algorithm <string> [ <integer> ]; ...
    };
    max-zone-ttl <duration>;
    nsec3param [ iterations <integer> ] [ optout <boolean> ] [ salt-length <integer> ];
    parent-ds-ttl <duration>;
    parent-propagation-delay <duration>;
    publish-safety <duration>;
    purge-keys <duration>;
    retire-safety <duration>;
    signatures-refresh <duration>;
    signatures-validity <duration>;
    signatures-validity-dnskey <duration>;
    zone-propagation-delay <duration>;
};
```

CHAPTER 17. EASY DNSSEC WITH BIND 9.16 "DEFAULT-POLICY"

- Using DNSSEC policy configuration (available since) makes DNSSEC easy to deploy
- BIND automatically creates the DNSSEC keys, signs the zone and keeps the signatures updated
- Policy `default` causes the zone to be signed with a single combined signing key (CSK) using algorithm ECDSAP256SHA256; this key will have an unlimited lifetime (no key rollover).

17.1. DNSSEC SIGNING THE ZONE WITH KASP

- We work on the primary authoritative server
- Ask the Trainer to remove the DS record for your zone `zXX.dane.onl` (manually signed=) from the parent `dane.onl` zone → this disables DNSSEC validation for your zone
- Test that DNSSEC validation is gone → no `AD` record for any zone below `zXX.dane.onl`
- Open the file `/etc/bind/named.conf` in an editor and add a `dnssec-policy default;` line to the zone block for the zone `zXX.dane.onl`. This will create a new CSK DNSSEC key for the zone in the BIND 9 "keys" directory :

```
zone "zXX.dane.onl" {
  type primary;
  file "zonefile.db";
  dnssec-policy  default;
  inline-signing yes;
};
```

- Open the zonefile `zonefile.db` and remove all DNSKEY records (but not the DS records used for secure delegation) from the zone, and remove the `zonefile.db.signed` file

```
$EDITOR /etc/bind/zonefile.db
rm /etc/bind/zonefile.db.signed
```

- Check the configuration and restart BIND 9

```
% named-checkconf -z
% systemctl restart named
```

- You should now see key files and additional zone-files in the BIND 9 home directory

```
% ls -l /etc/bind
% ls -l /etc/bind/keys
```

- Test if the DNS server returns DNSSEC resource records

```
$ dig @localhost zXX.dane.onl +dnssec +multi
$ dig @localhost zXX.dane.onl DNSKEY +dnssec +multi
```

- Check that the new DNSSEC signed zone will be synchronized to the secondary

```
$ dig zXX.dane.onl +nssearch
```

- Resolve a domain name `zXX.dane.onl` from your zone on your resolver machine. Does it show the `AD` -Flag? What might be missing?

- Try to finish the DNSSEC setup to get an `AD` flag in the requests

17.2. MIGRATION BIND 9 DNSSEC ZU EINER KASP POLICY

- BIND 9.16/9.18 führt eine neue (bessere) Automatisierung der DNSSEC-Signierung von DNS-Zonen ein: während in den älteren BIND 9 Versionen die Key-Rollover mittels der Meta-Daten auf den Schlüssel-Dateien automatisiert werden konnten, können die Key-Rollover nun innerhalb der BIND 9 Konfiguration in einer KASP (Key-and-Signing-Policy) definiert werden. Der BIND 9.18 führt ggf. Key-Rollover automatisiert durch.
- Zonen, welche schon vor BIND 9.16 oder BIND 9.18 mit DNSSEC abgesichert waren sollen auf das neue KASP System migriert werden, da die alte DNSSEC-Automatisierung in BIND 9 abgekündigt ist und in einer zukünftigen Version von BIND 9 (9.19-dev, 9.20-release) nicht mehr verfügbar sein wird.
- Für eine erfolgreiche Migration ist es erforderlich, eine DNSSEC-KASP zu definieren, welche zu den bestehenden DNSSEC-Schlüsseln in der Zone passt (Algorithmus, Anzahl Schlüssel = CSK oder KSK/ZSK).

  Erkennt der BIND 9 beim Laden der Zone ein Differenz zwischen den existierenden Schlüsseln und der neuen Policy, so wird BIND 9 die bestehenden Schlüssel **nicht** mehr benutzen und stattdessen neue Schlüssel erstellen und **sofort** die Zone mit den neuen Schlüsseln signieren. Dies kann zu einem Total-Ausfall der Zone führen, da der DS-Record in der Eltern-Zone nicht zu KSK/CSK der Zone passt.

  Bei der Migration auf eine KASP-Konfiguration sollte die Lebensdauer der Schlüssel mit "unlimited" angegeben werden. Nach der erfolgreichen Migration kann die KASP-Konfiguration angepasst und die Lebensdauer der Schlüssel begrenzt werden.

  Es ist sinnvoll, eine Migration von einer pre-KASP Konfiguration einer Zone auf eine Zone mit DNSSEC-KASP and einer Test-Zone zu üben.

  Bei einer erfolgreichen Migration auf eine KASP-Konfiguration übernimmt der BIND 9 die bestehenden DNSSEC-Schlüssel und signiert die Zone weiterhin mit diesen Schlüsseln.

  Werden neuen Schlüssel erzeugt und für die Signierung verwendet, so war die Migration nicht erfolgreich.

  Bei der Benutzung eines *hidden-primary* DNS-Servers, sollte für die Aktivierung der KASP-Konfiguration die Kommunikation zwischen den aktiven Auth-Servern und dem *hidden-primary* unterbrochen werden, bis die erfolgreiche Migration bestätigt ist (in der Regel direkt nach dem Laden der neuen Konfiguration sichtbar).

  Bei der Benutzung einer Infrastruktur ohne *hidden-primary* (der Primary ist aktiver DNS-Server in der Delegation) kann der Primary-Server temporär in einen "hidden-primary" umgewandelt werden (z.B. durch ein Block von UDP/TCP Port 53 in der Firewall). Der oder die restlichen DNS-Server werden die bestehende DNSSEC-Zone ohne Unterbrechung weiter ausliefern, bis die erfolgreiche Migration bestätigt ist und die Blockade in der Firewall aufgehoben wurde.
- Beispiel einer Pre-KASP-BIND 9 DNSSEC-Konfiguration. Die Schlüssel (ZSK/KSK) wurden manuell mit `dnssec-keygen` mit dem Algorithmus RSASHA256 erstellt:

```
   options {
      directory "/var/named";
      key-directory "keys";
   };
+
   zone "example.com" IN {
     type primary;
     file "example.com";
     auto-dnssec maintain;
     inline-signing yes;
   };
```

- Beispiel einer KASP BIND 9 DNSSEC-Konfiguration für diese Zone:

```
   dnssec-policy "base-dnssec" {
      keys {
        ksk lifetime unlimited algorithm RSASHA256;
        zsk lifetime unlimited algorithm RSASHA256;
      };
   };
+
   options {
      directory "/var/named";
      key-directory "keys";
   };
+
   zone "example.com" IN {
     type primary;
     file "example.com";
     dnssec-policy "base-dnssec";
     inline-signing yes;
   };
```

- Eine produktive KASP-Konfiguration mit ZSK-Rollover (aber keinem automatischem KSK-Rollover):

```
dnssec-policy "DNSSEC-ZSK-ROLL" {
       dnskey-ttl 60;
       keys { ksk lifetime unlimited algorithm ECDSAP256SHA256;
              zsk lifetime P30D      algorithm ECDSAP256SHA256;
            };
       max-zone-ttl 3600;
       publish-safety 1h;
       purge-keys P90D;
       retire-safety 1h;
       signatures-refresh 5d;
       signatures-validity 14d;
       signatures-validity-dnskey 14d;
       zone-propagation-delay 300;
 };

 zone "example.de" {
     type primary;
     inline-signing yes;
     dnssec-policy "DNSSEC-ZSK-ROLL";
     file "primary/example.de";
};
```

- Beispiel einer KASP BIND 9 DNSSEC-Konfiguration mit automatischem ZSK/KSK Rollover mittels CDS/CDNSKEY (Schweiz):

```
parental-agents "switch" {
   130.59.31.41;
   130.59.31.43;
   194.0.25.39;
   194.0.17.1;
   194.146.106.10;
   2001:620:0:ff::56;
   2001:620:0:ff::58;
   2001:678:20::39;
   2001:678:3::1;
   2001:67c:1010:2::53;
};
```

```
dnssec-policy "DNSSEC-AUTO" {
    keys {
        ksk lifetime P90D  algorithm ecdsap256sha256;
        zsk lifetime P45D  algorithm ecdsap256sha256;
    };
    max-zone-ttl 3600;
    parent-ds-ttl 600;
    parent-propagation-delay 2h;
    publish-safety 7d;
    retire-safety 7d;
    signatures-refresh 5d;
    signatures-validity 15d;
    signatures-validity-dnskey 15d;
    zone-propagation-delay 2h;
};

zone "example.ch" {
     type primary;
     allow-update { key dns-update; };
     allow-transfer { secondaries; };
     dnssec-policy "DNSSEC-AUTO";
     parental-agents {
        "switch";
     };
     file "primary/example.ch";
};
```

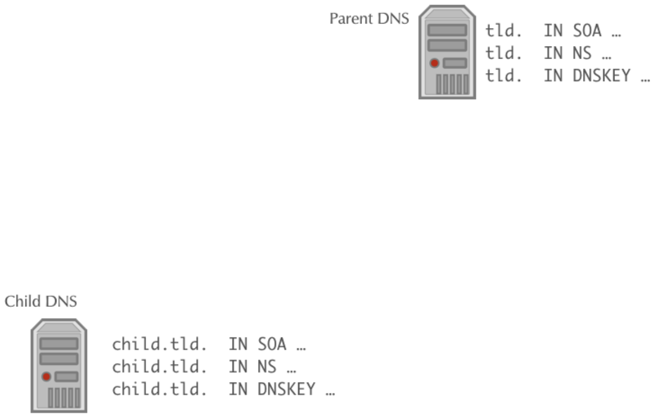
CHAPTER 18. DNSSEC KEY-ROLLOVER AUTOMATION

- A DNSSEC key rollover is a delicate process that needs to be done carefully
- Automation can prevent human errors in the process
- With BIND 9, a ZSK rollover can be fully automated
- A KSK rollover can be fully automated with a parent domain that supports *Automating DNSSEC Delegation Trust Maintenance* (RFC 7344/8078)

18.1. AUTOMATING DNSSEC DELEGATION TRUST MAINTENANCE (RFC 7344/8078)

- RFC 7344/8087 defines an automatic way to update the chain of trust towards the parent zone in an KSK key rollover
- The operator of the zone creates a new KSK for the zone and then publishes the new DS record or/and DNSKEY record for the parent zone in a CDS and/or CDNSKEY record in the zone
- The authoritative DNS server for the parent zone periodically polls the child zones for new CDS or CDNSKEY records
- Once a new CDS record is found, it will be validated against the current KSK and DS records, and if it is valid, it will be imported into the parent zone (replacing the DS record)

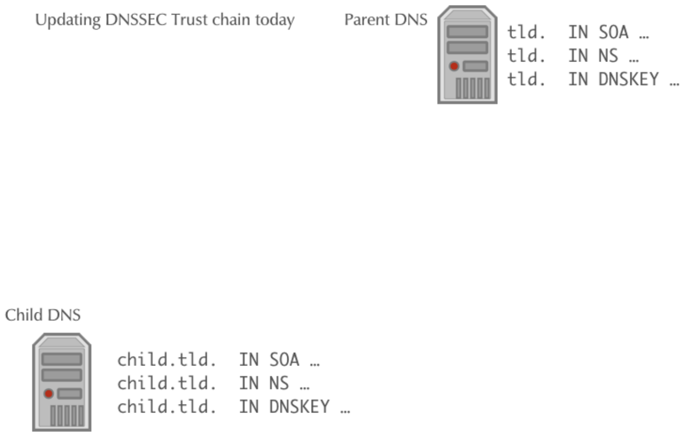
  If a new CDNSKEY record is found in the child zone, the authoritative DNS server of the parent zone will validate the record, calculate the hash to get a DS record, and will then replace the DS record with the newly calculated DS record
- BIND 9 supports CDS and CDNSKEY since version 9.11
- The `dnssec-cds` utility can change DS records for a child zone based on CDS/CDNSKEY records
- CDS and CDNSKEY is already supported by some TLDs (Czech Republic ".cz". Switzerland ".ch", Lichtenstein ".li", Sweden ".se" …)



```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```

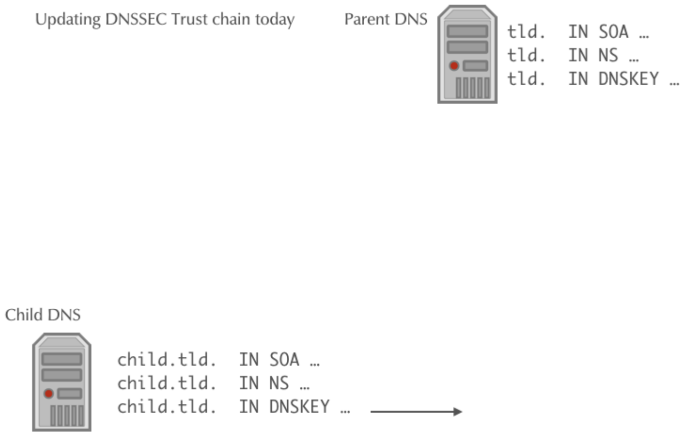
Updating DNSSEC Trust chain today



```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```

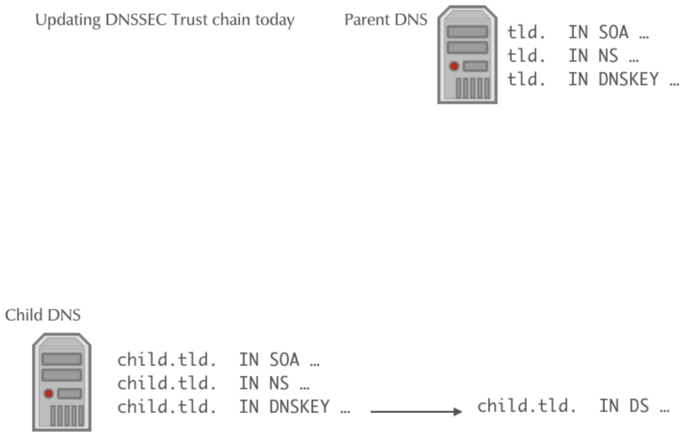
Updating DNSSEC Trust chain today



```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

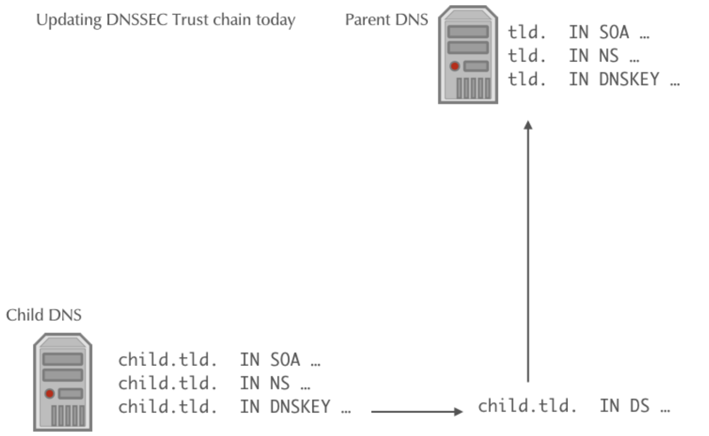
```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```

Updating DNSSEC Trust chain today



```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …  ——→  child.tld.  IN DS …
```

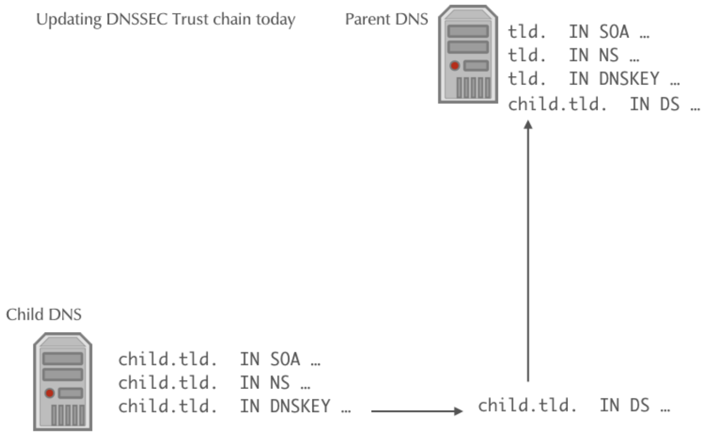


Updating DNSSEC Trust chain today

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```

```
child.tld.  IN DS …
```

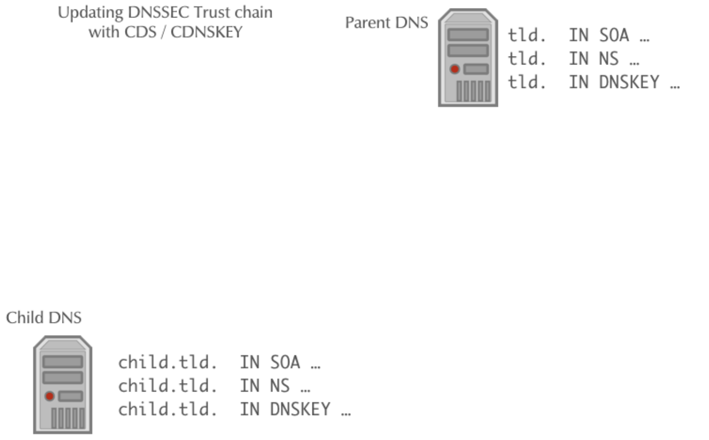


Updating DNSSEC Trust chain today

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
child.tld.  IN DS …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```

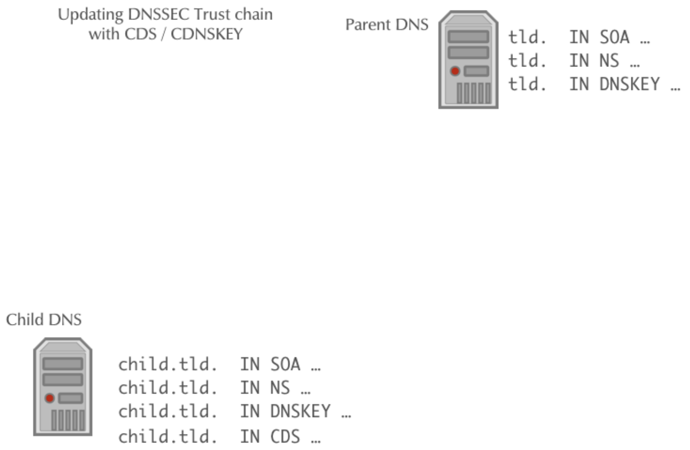
```
child.tld.  IN DS …
```



Updating DNSSEC Trust chain
with CDS / CDNSKEY

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
```



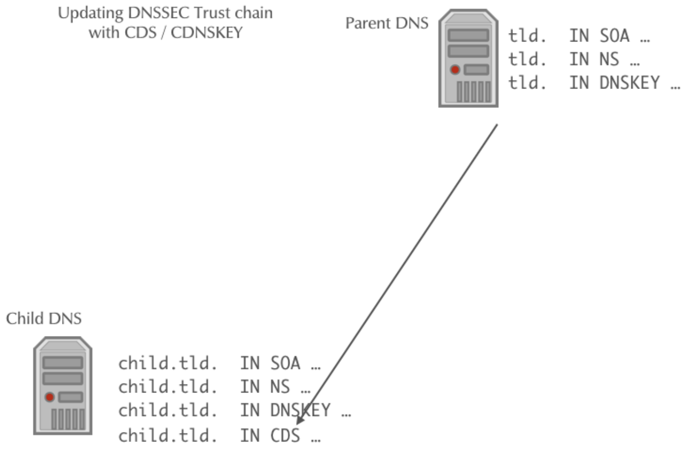
Updating DNSSEC Trust chain
with CDS / CDNSKEY

Parent DNS

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

Child DNS

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
child.tld.  IN CDS …
```



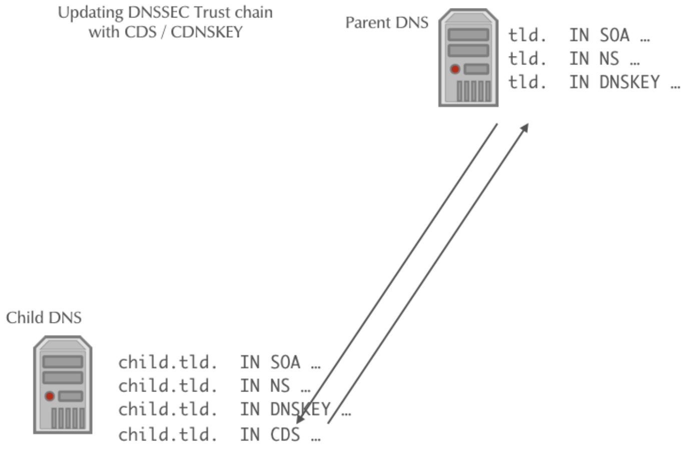
Updating DNSSEC Trust chain
with CDS / CDNSKEY

Parent DNS

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

Child DNS

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
child.tld.  IN CDS …
```



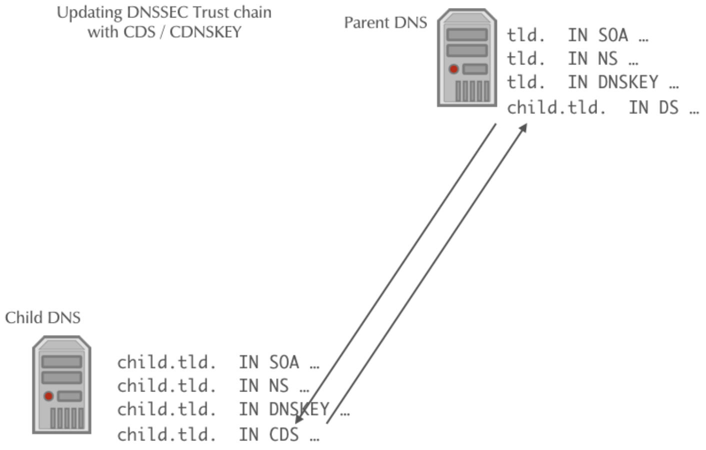
Updating DNSSEC Trust chain
with CDS / CDNSKEY

Parent DNS

```
tld.  IN SOA …
tld.  IN NS …
tld.  IN DNSKEY …
```

Child DNS

```
child.tld.  IN SOA …
child.tld.  IN NS …
child.tld.  IN DNSKEY …
child.tld.  IN CDS …
```



18.1.1. Bootstrapping DNSSEC with CDS/CDNSKEY

- The CDS/CDNSKEY records can be used to bootstrap a DNSSEC signed zone

  The operator of a DNS zone signs the zone and places the CDS/CDNSKEYs in the zone

  The parent domain operator polls all non-DNSSEC zones for the existence of CDS/CDNSKEYs. Is the same CDS/CDNSKEY record found for a specific time in the zone, the parent imports the records and creates the DS record in the parent zone, closing the DNSSEC chain of trust

  The exact requirements and rules for DNSSEC bootstrapping depend on the registries policy. See for example for the `.ch` (Swiss) and `.li` (Lichtenstein) Top-Level-Domains: https://www.nic.ch/security/cds/ [https://www.nic.ch/security/cds/]

18.1.2. Bootstrapping DNSSEC trust

- The challenge with bootstrapping DNSSEC: there is no established cryptographic trust between the zone and the parent zone DNS server
- A new IETF draft (RFC "in the making") make use of already secured domains that hold the host-names of the authoritative DNS servers for the zone to be DNSSEC secured

  It only works for zones that have at least one NS record hostname outside the delegated zone (*out-of-bailick*), which is good practice for resilience

  In addition to the zone itself, the new CDS/CDNSKEY records are published as *signaling-records* in the domains of the zones nameserver hostnames

- Example:

  Zone to be signed `dnssec.works` has the following NS record set

```
dnssec.works.      3600 IN NS   ns01.dane.onl.
dnssec.works.      3600 IN NS   ns02.dnslab.org.
dnssec.works.      3600 IN NS   ns03.dnssec.works.
```

- The server `ns01` and `ns02` are located outside the zone, the server `ns03` is located inside the zone `dnssec.works`

- The zone `dane.onl` will publish the following *signalling record* (the CDS or CDNSKEY record):

```
_dsboot.dnssec.works._signal.ns01.dane.onl. IN CDS ....
```

- The zone `dnslab.org` will publish the following *signalling record* (the CDS or CDNSKEY record):

```
_dsboot.dnssec.works._signal.ns02.dnslab.org. IN CDS ....
```

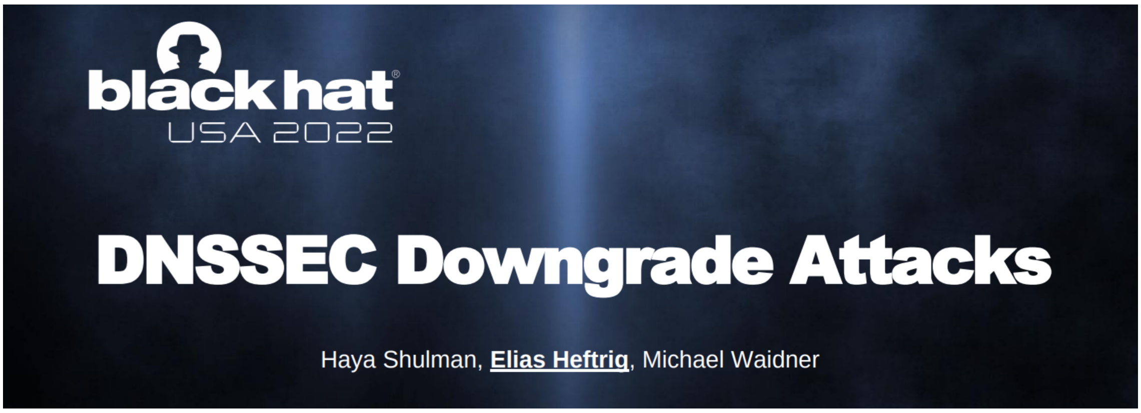
- Because both the zones `dane.onl.` and `dnslab.org.` are DNSSEC signed and provide the hostnames required for the delegation on the zone, the parent zone operator can now fetch the CDS or CDNSKEY in a DNSSEC secured way
- When using secure DNSSEC bootstrapping, there is no delay in publishing the new DS record in the parent compared with the insecure "Accept after delay" procedure documented in RFC 8078.
- Draft "DNSSEC bootstrapping" (A draft is not yet a standard, and might never be): https://datatracker.ietf.org/doc/draft-ietf-dnsop-dnssec-bootstrapping/ [https://datatracker.ietf.org/doc/draft-ietf-dnsop-dnssec-bootstrapping/]
- This draft is already supported by the `.ch` and `.li` TLDs

18.1.3. Further Reading

- DNSSEC provisioning automation with CDS/CDNSKEY in the real world https://jpmens.net/2021/10/05/dnssec-cds-cdnskey-in-the-real-world/ [https://jpmens.net/2021/10/05/dnssec-cds-cdnskey-in-the-real-world/]

CHAPTER 19. DNSSEC BEST-PRACTICES

19.1. DNSSEC BR0K3N?



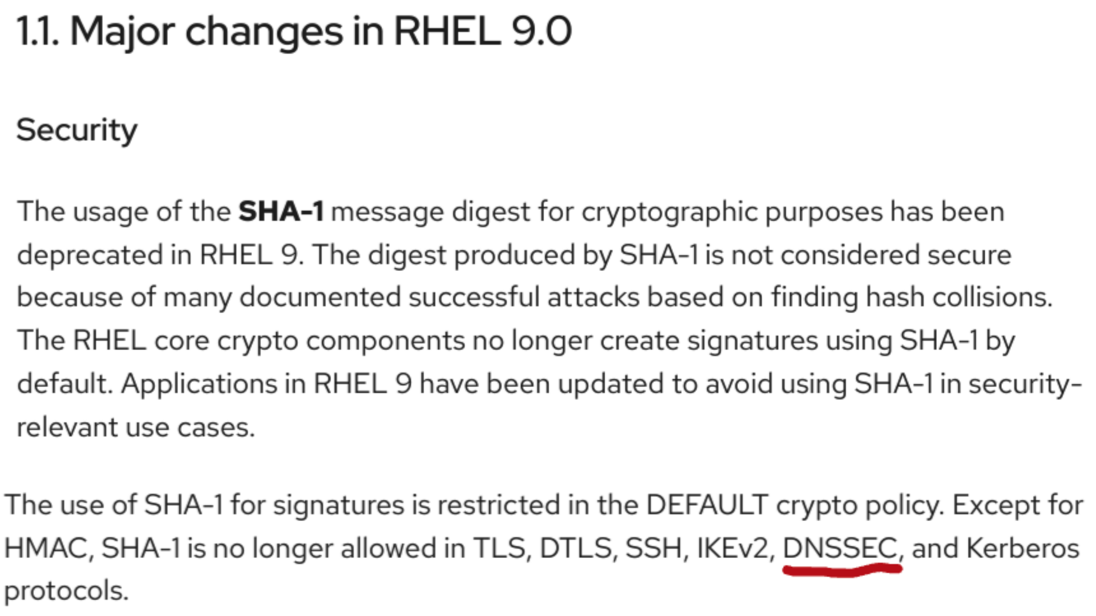
black hat USA 2022

DNSSEC Downgrade Attacks

Haya Shulman, **Elias Heftrig**, Michael Waidner

https://i.blackhat.com/USA-22/Thursday/US-22-Heftrig-DNSSEC-Downgrade-Attacks.pdf [https://i.blackhat.com/USA-22/Thursday/US-22-Heftrig-DNSSEC-Downgrade-Attacks.pdf]

19.2. SHA1 PHASE OUT IN RED HAT EL 9



1.1. Major changes in RHEL 9.0

Security

The usage of the **SHA-1** message digest for cryptographic purposes has been deprecated in RHEL 9. The digest produced by SHA-1 is not considered secure because of many documented successful attacks based on finding hash collisions. The RHEL core crypto components no longer create signatures using SHA-1 by default. Applications in RHEL 9 have been updated to avoid using SHA-1 in security-relevant use cases.

The use of SHA-1 for signatures is restricted in the DEFAULT crypto policy. Except for HMAC, SHA-1 is no longer allowed in TLS, DTLS, SSH, IKEv2, DNSSEC, and Kerberos protocols.

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/9.0_release_notes/overview [https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/9.0_release_notes/overview]

19.3. AVOID SHA1

- The hash algorithm SHA1 is weak
- It should be phased out for existing DNSSEC zones
- It should not be used for new DNSSEC deployments (use RSASHA256 or ECDSA)
- Double-Signing zones with SHA1 and other (stronger) algorithms does not help as downgrade attacks are possible

19.4. AVOID UDP FRAGMENTATION

- UDP fragmentation can be used to attack DNS traffic
- There is a specific danger for DNS resolver that does not validate DNSSEC

  But receive DNSSEC signed data, as DNSSEC signatures create large(r) DNS responses
- BSI Study: IP Fragmentation and Measures against DNS Cache Poisoning (Frag-DNS) [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Frag-DNS/Frag-DNS-Studie.html]

19.5. AVOID UDP FRAGMENTATION - DNS AUTHORITATIVE SERVER OPERATORS

- Avoid old(er) Linux kernels that are vulnerable to MTU downgrade attacks through ICMP spoofing (Long-Term/Enterprise Kernel)
- Configure the authoritative DNS server with an maximum UDP answer size of 1232byte (EDNS0 Buffer Size)
- Make sure the authoritative DNS server does respond on TCP port 53 (See RFC 7766 - DNS Transport over TCP - Implementation Requirements [https://www.rfc-editor.org/rfc/rfc7766])
- DNSSEC sign your zones so that resolvers can validate

19.6. AVOID UDP FRAGMENTATION - DNS RESOLVER OPERATORS

- Enable DNSSEC validation
- Configure the authoritative DNS server with an maximum UDP answer size of 1232byte (EDNS0 Buffer Size)
- Make sure the DNS resolver can query DNS content over TCP
- Block fragmented DNS responses coming from the Internet in the Firewall (they should never occur with an EDNS0 buffer of 1232byte)

CHAPTER 20. DNSSEC TROUBLESHOOTING

20.1. CHECKING DNS RESOLUTION ISSUES

20.1.1. dig

The DNS name resolution tool `dig` can be used to test the general function of a DNS resolver, or to test if an error condition exist at the remote DNS authoritative servers of a domain. * testing one DNS resolver over UDP To test the general connectivity and operation of a DNS resolver, a query for well known DNS data can be sent, such as for the list of name-server (NS records) of the root zone `"."`. The answer should contain the list of all 13 root name server in the Internet (with the names `a-m.root-servers.net`)

```
$ dig @IP-of-DNS-resolver NS .
```

- What are reasons we recommend using the IP address of a resolver instead of its name? Write answers in the chat.
- testing one DNS resolver over TCP The DNS resolvers must also be reachable over TCP. To send a query via TCP, add the `+tcp` flag to queries.

```
$ dig @IP-of-DNS-resolver NS . +tcp
```

- You can save 33% typing by using `+vc` instead of `+tcp`. What does `vc` do? Check the manual page for `dig(1)`. Write answers in the chat.
- testing reachability of all authoritative DNS servers The `dig` function `+nssearch` will first query all NS records for a given domain and then will try to query directly (without going to a DNS-resolver) the authoritative DNS servers of that domain:

```
$ dig @IP-of-DNS-resolver example.com +nssearch
```

The function will print out the SOA record of the zone (here `example.com`) for each DNS server that has send an answer to the query, including the SOA serial, the IPv4 and/or IPv6 address and the round-trip-time (RTT) of the query.

All SOA serial numbers should show the same number, else there might be an issue with zone synchronization via zone transfer which might be a possible cause for DNS lookup problems.

- Which type of DNS servers (authoritative, recursive, primary, secondary) are at fault if the SOA serial numbers of a zone are not in sync? Who can correct the fault? Write answers in the chat.
- testing the resolution chain The function `+trace` in `dig` will trace the DNS name resolution starting from the root DNS server system down to the requested name.

```
$ dig @IP-of-DNS-resolver example.com +trace
```

Only the very first query to find the root-server addresses will be done towards the DNS resolver given in the command, all other queries will be sent directly to the authoritative DNS servers. This function tests and prints one of usually many possible DNS resolution paths. A successful return of the command does not indicate that the resolution path is without errors, it is only an indication that at least one successful path exists. * testing for DNSSEC validation issues If a DNS query returns a `SERVFAIL` answer, it can be a DNSSEC validation issue at the DNS resolver, or it can be some kind of server malfunction on the DNS resolver or the remote authoritative server.

In order the check for DNSSEC validation issues, the administrator can send a DNS query with the `+cd` flag (Checking Disabled). With this flag set, the DNS resolver will skip DNSSEC validation and will return the DNS data to `dig` even in case the DNSSEC validation would fail.

So if a DNS query returns data when `+cd` is set, but returns `SERVFAIL` when `+cd` is not set, this indicates a DNSSEC validation issue. If the answer is always `SERVFAIL`, it is some other kind of problem (usually not DNSSEC).

20.1.2. external web services to check DNS

Several website services exist that help DNS administrators to check the health of a DNS system

Zonemaster

The website Zonemaster https://zonemaster.net [https://zonemaster.net] is a collaboration between the French TLD registry *AFNIC* and the Swedish registry *IIS*. The website takes a domain name and will generate a report of errors and best practice recommendations of the setup of this domain name.

DNSViz

DNSViz https://dnsviz.net [https://dnsviz.net] is a tool for visualizing the status of a DNS zone. It provides a visual analysis of the DNSSEC authentication chain for a domain name and its resolution path in the DNS namespace, and it lists configuration errors detected by the tool.

20.2. LOOKING INTO DNSSEC VALIDATION ISSUES

DNSEC validation issues are often a problem with misconfiguration at the authoritative DNS server side of the domain, not an issue of the DNS resolver system. However to be able to debug DNSSEC issues, for example to decide if an NTA (negative trust anchor) should be inserted into the DNS resolver system to temporarily disable DNSSEC validation for a specific domain, the DNSSEC validation issue should be investigated first.

20.2.1. DNSSEC validation troubleshooting with "delv"

The tool `delv` is part of the BIND 9 DNS server and implements a full DNS resolver and DNSSEC validator inside the command line tool. This tool works very similarly to `dig` (and shares the same command line syntax), but where `dig` always needs a DNS resolver to get the DNS information, `delv` can query the data and can validate the DNSSEC information itself.

20.2.2. Message trace

The flag `+mtrace` enables the message tracing in `delv`, the tool will print all DNS queries and answers during the processing of the query.

```
$ delv example.com +mtrace
```

20.2.3. Validation trace

The flag `+vtrace` will print a debug trace of all DNSSEC validation steps the tool will do in order to validate the received DNS answers

```
$ delv example.com +vtrace
```

20.2.4. DNSSEC path validation with drill

The tool `drill` is part of the LDNS tools (Debian/Ubuntu package `ldnsutils`) and is a `dig` lookalike tool with additional features. One notable feature is the function to trace the DNSSEC validation and to print the DNSSEC chain-of-trust together with the DNSSEC key information.

`drill` requires the current DNSSEC root trust anchor in a file to be able to work. This trust anchor can be received with `drill`, but also with `dig` or `delve`:

```
$ dig @a.root-servers.net . DNSKEY | grep 257 > root.key
```

This DNSSEC root trust-anchor key can be used to do a DNSSEC chain chase with `drill`:

```
$ drill -SD -k root.key example.com
```

- In the example above we fetched the root DNSKEY with `dig`. Is this method for obtaining the root DNSKEY record foolproof, and do you consider it secure? Write answers in the chat.

20.3. EXERCISE: DNSSEC TROUBLESHOOTING

- A customer owns the domain `dnssec.works`. This domain has 5 sub-domains, `fail01.dnssec.works` to `fail05.dnssec.works`

  all 5 subdomains contain DNS errors on the IPv4 address record(!)

- Your task: find out the DNSSEC issues with these zones
- Use `dig` to find the issues
- Also use the websites https://dnsviz.net [https://dnsviz.net] and https://zonemaster.net [https://zonemaster.net] to confirm your findings
- write your findings to the chat

CHAPTER 21. NEWS

> Breaking news, Microsoft is pulling the trigger on DANE next year: Implementing Inbound SMTP DANE with DNSSEC for Exchange Online Mail Flow - Microsoft Community Hub

https://techcommunity.microsoft.com/t5/exchange-team-blog/implementing-inbound-smtp-dane-with-dnssec-for-exchange-online/ba-p/3939694 [https://techcommunity.microsoft.com/t5/exchange-team-blog/implementing-inbound-smtp-dane-with-dnssec-for-exchange-online/ba-p/3939694]

CHAPTER 22. DNSDIST DNS LOAD-BALANCER

- ISC Webinar Slides https://webinar.defaultroutes.de/webinar/03-dnsdist.html [https://webinar.defaultroutes.de/webinar/03-dnsdist.html]