# DNS & BIND 9 DNSSEC TRAINING

## CARSTEN STROTMANN

Version 20251106, 06.11.2025

# INHALTSVERZEICHNIS

# CHAPTER 1. 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 9499 *DNS terminology* ( https://tools.ietf.org/html/rfc9499 [https://tools.ietf.org/html/rfc9499] ) attempts to collect and document the current usage of DNS terminology.

# CHAPTER 2. DNS 1X1

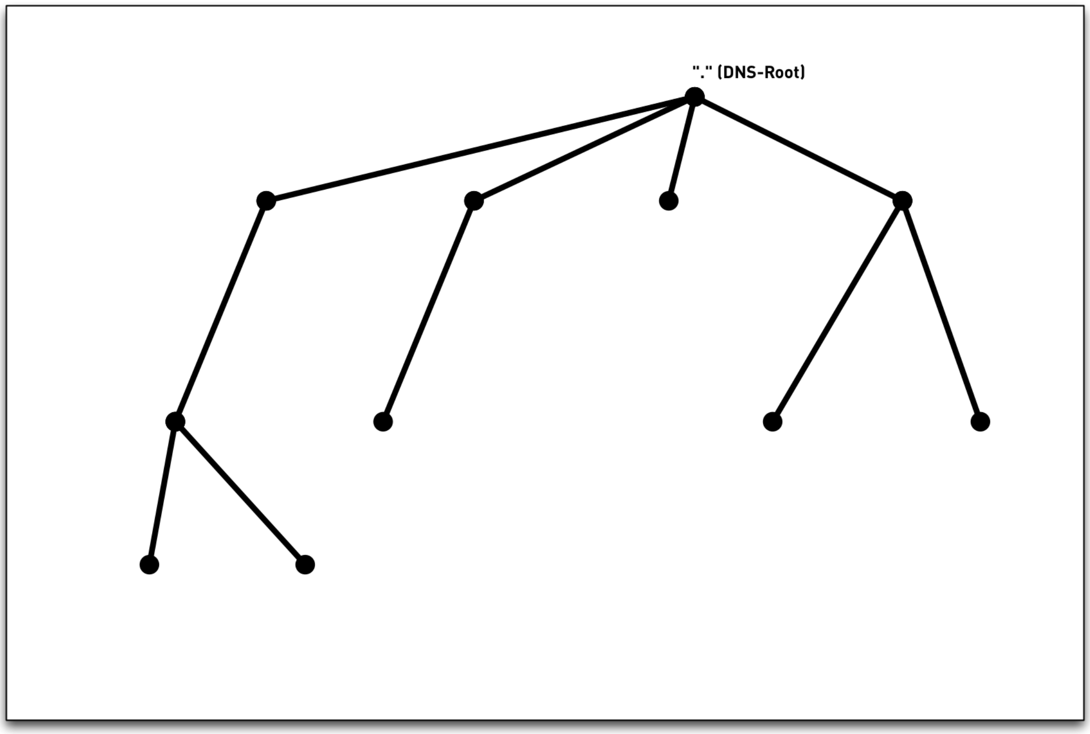
## 2.1. DNS - THE DOMAIN NAME SYSTEM

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

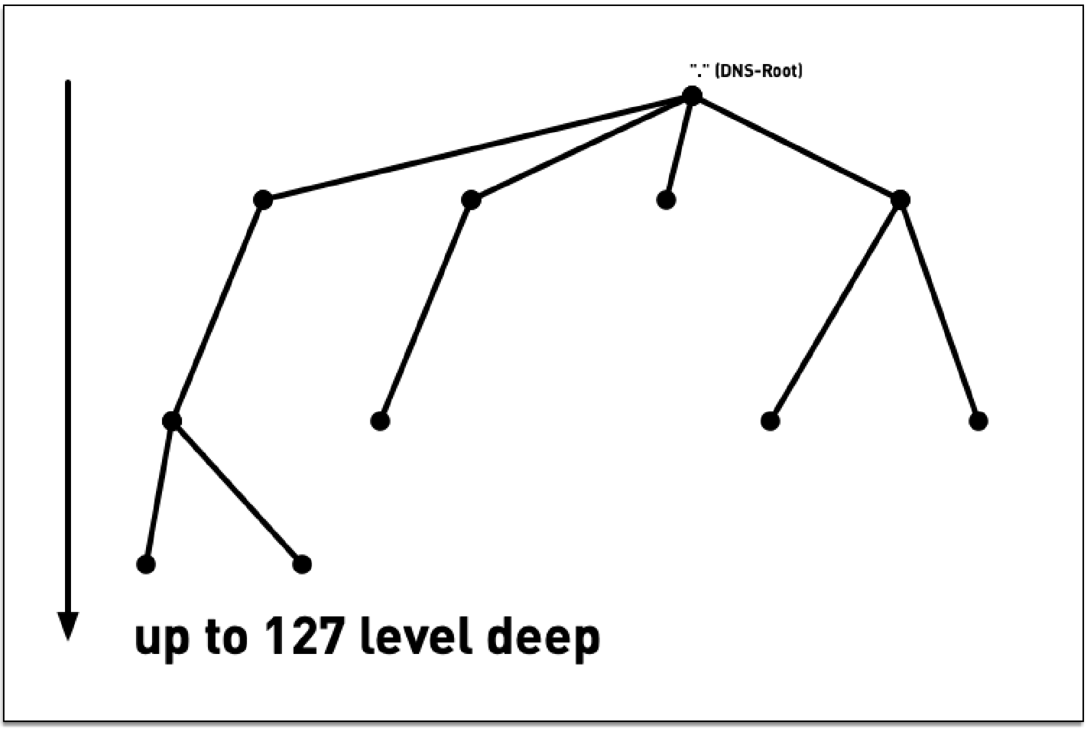
## 2.2. DNS - THE DOMAIN NAME SYSTEM

- 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

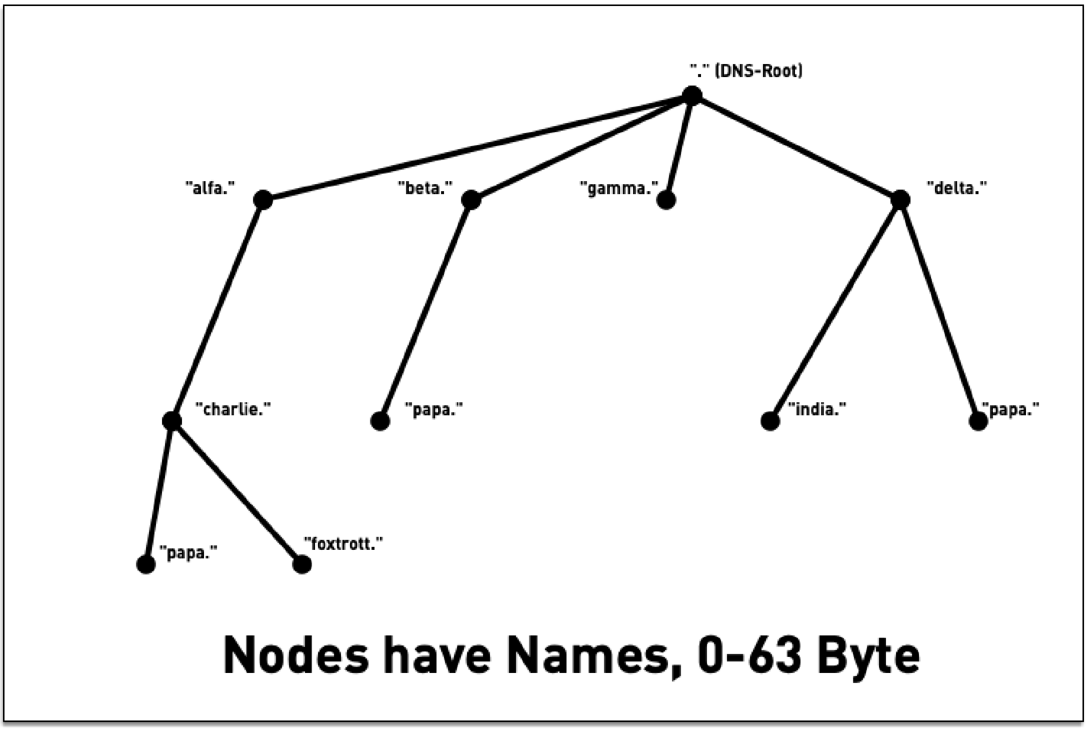
## 2.3. DNS NAMESPACE



### 2.4. DNS NAMESPACE

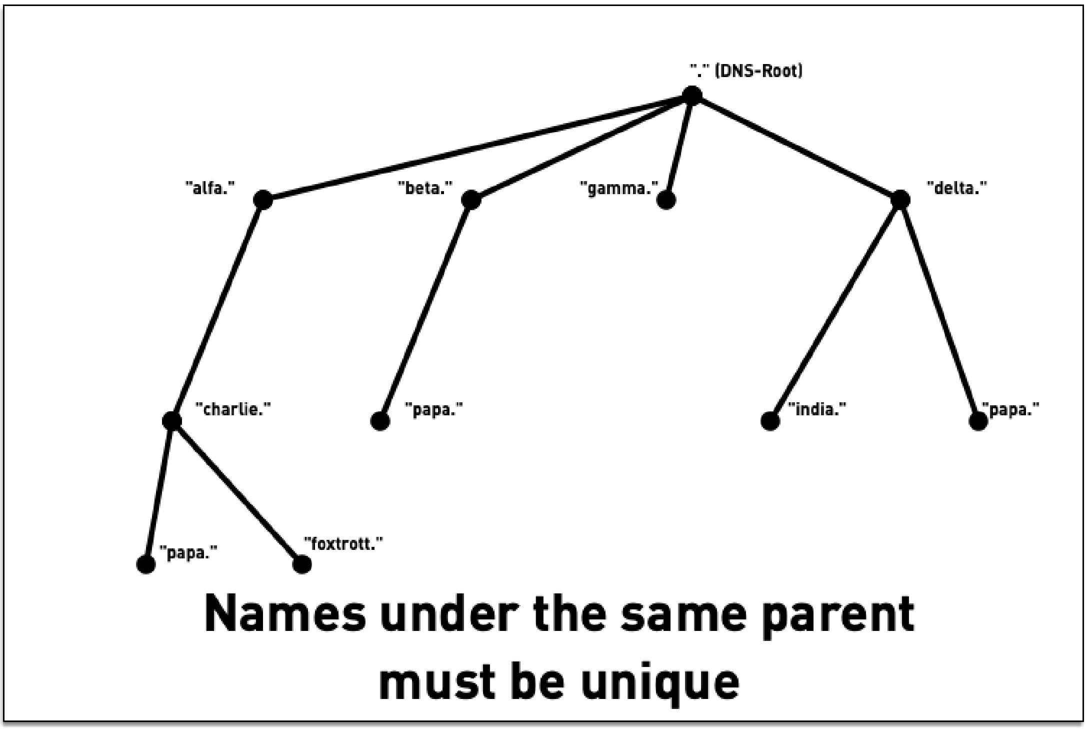


## 2.5. DNS NAMESPACE - "LABEL"



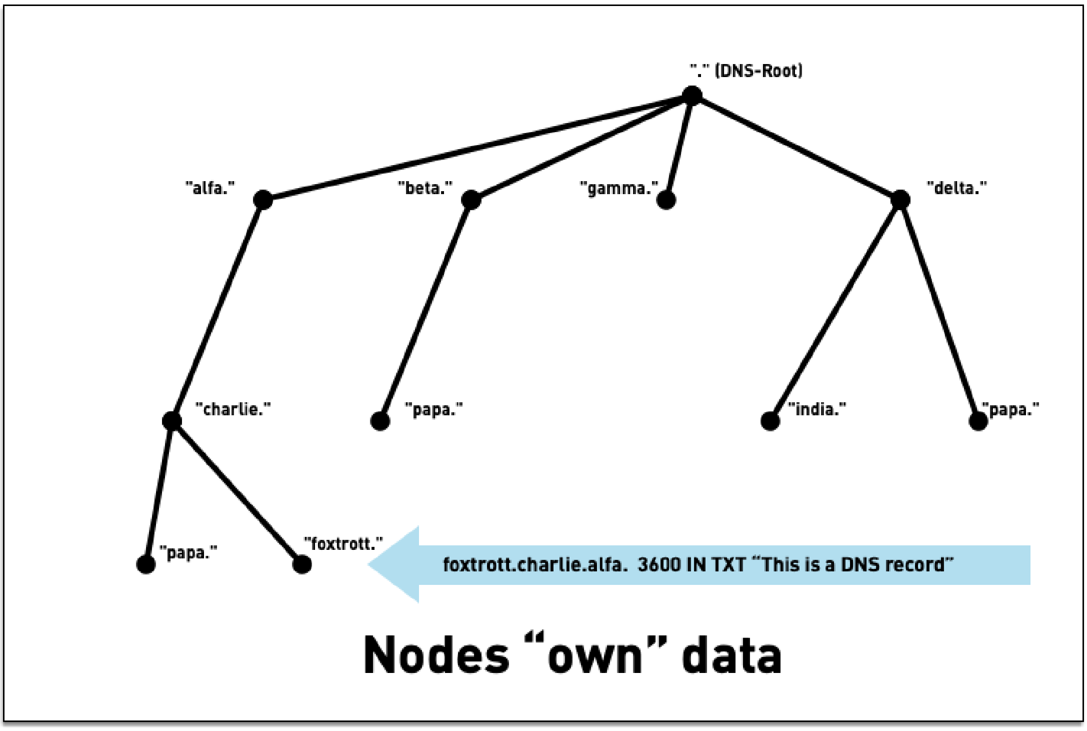
Nodes have Names, 0-63 Byte

## 2.6. DNS NAMESPACE - "LABEL"



Names under the same parent must be unique

## 2.7. DNS NAMESPACE - "LABEL"

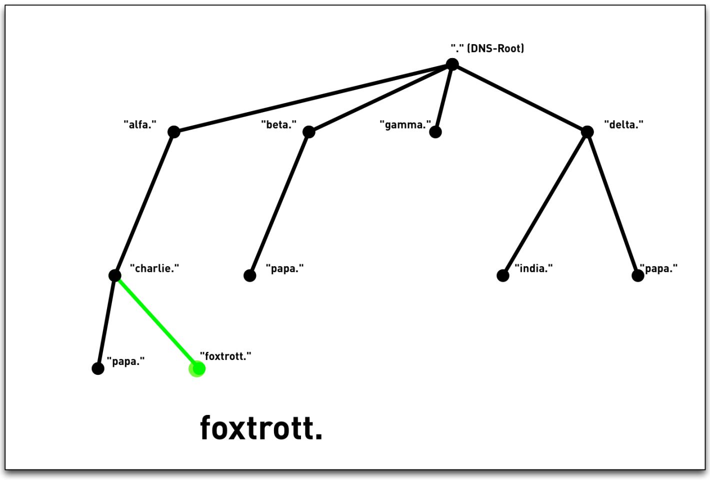


foxtrott.charlie.alfa.  3600 IN TXT "This is a DNS record"

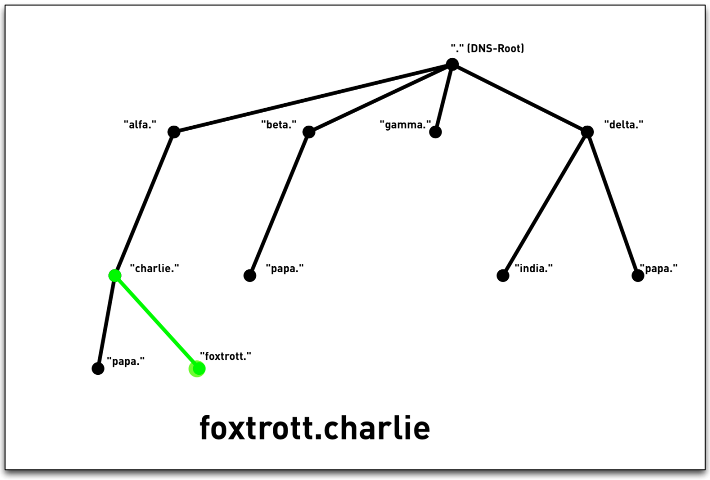
Nodes "own" data

## 2.8. DNS NAMESPACE

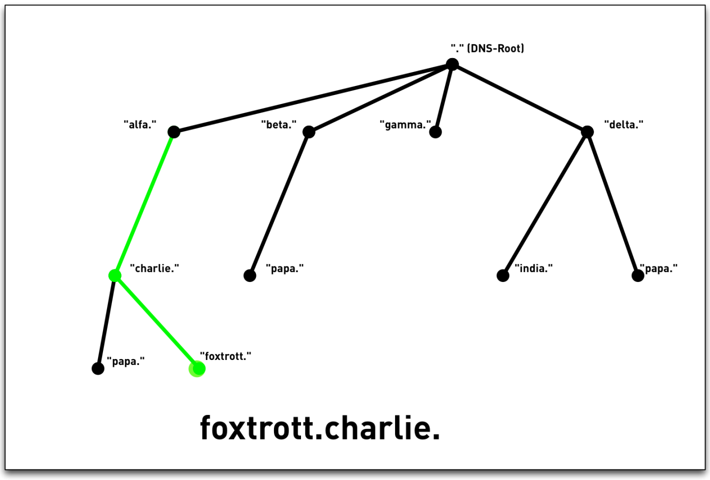
```
    [sidebar]
image::./img/DNS-namespace-00.png[scaledwidth=65%]
```



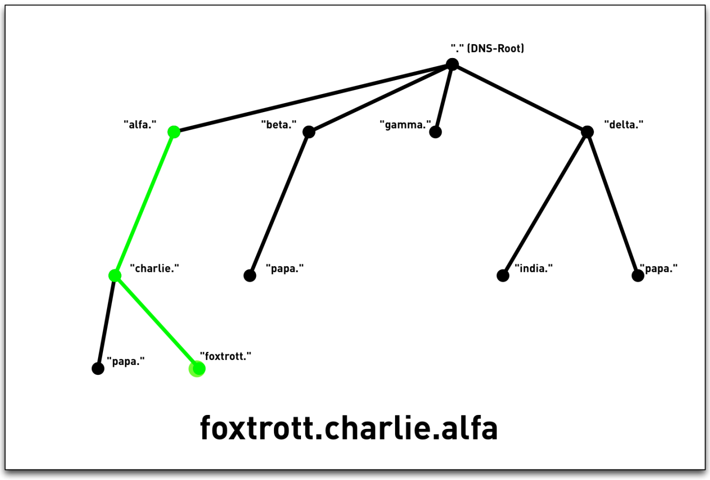
foxtrott.



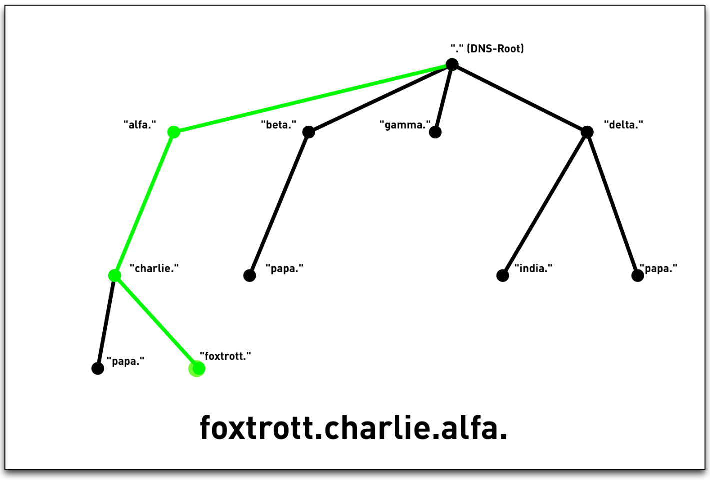
foxtrott.charlie



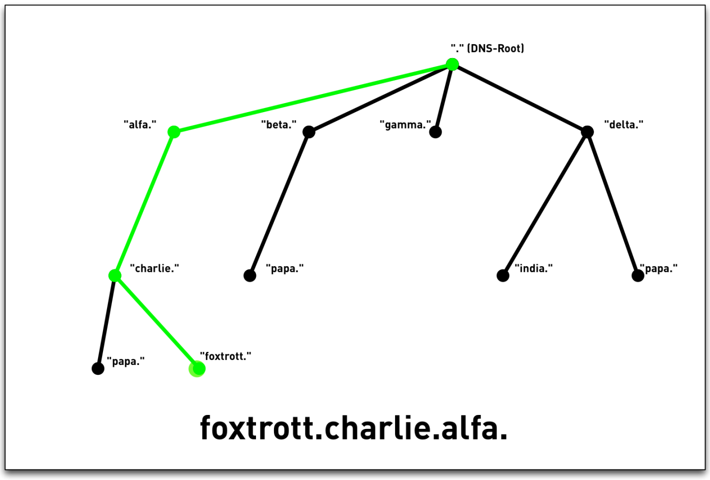
foxtrott.charlie.



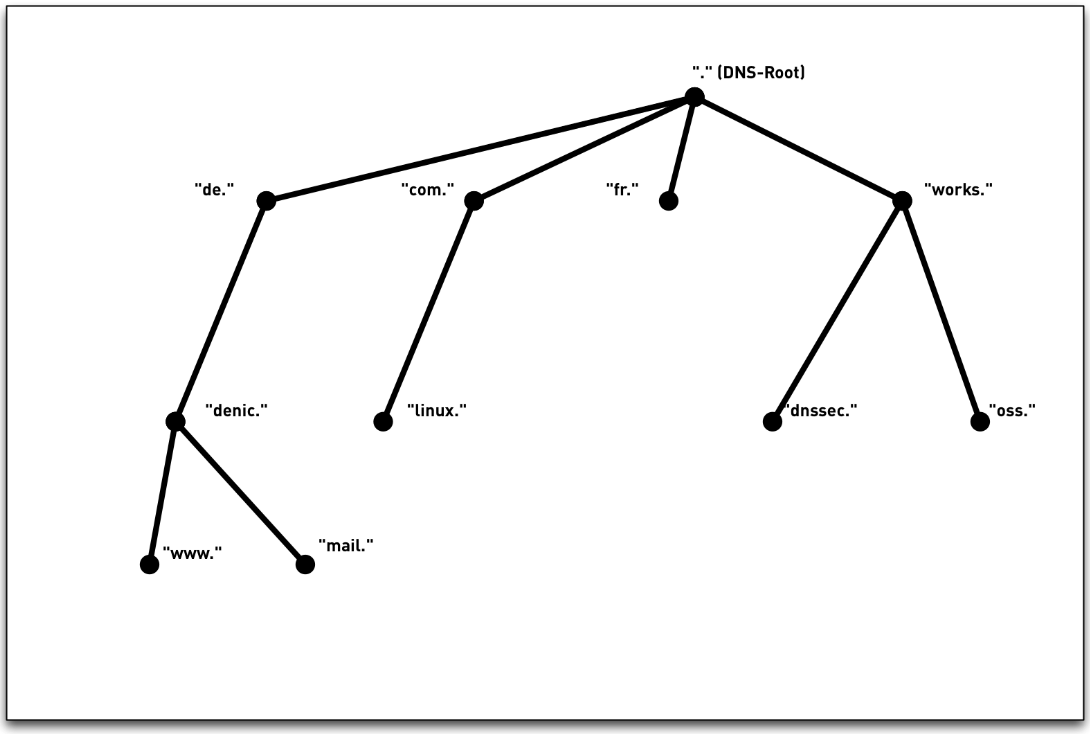
foxtrott.charlie.alfa



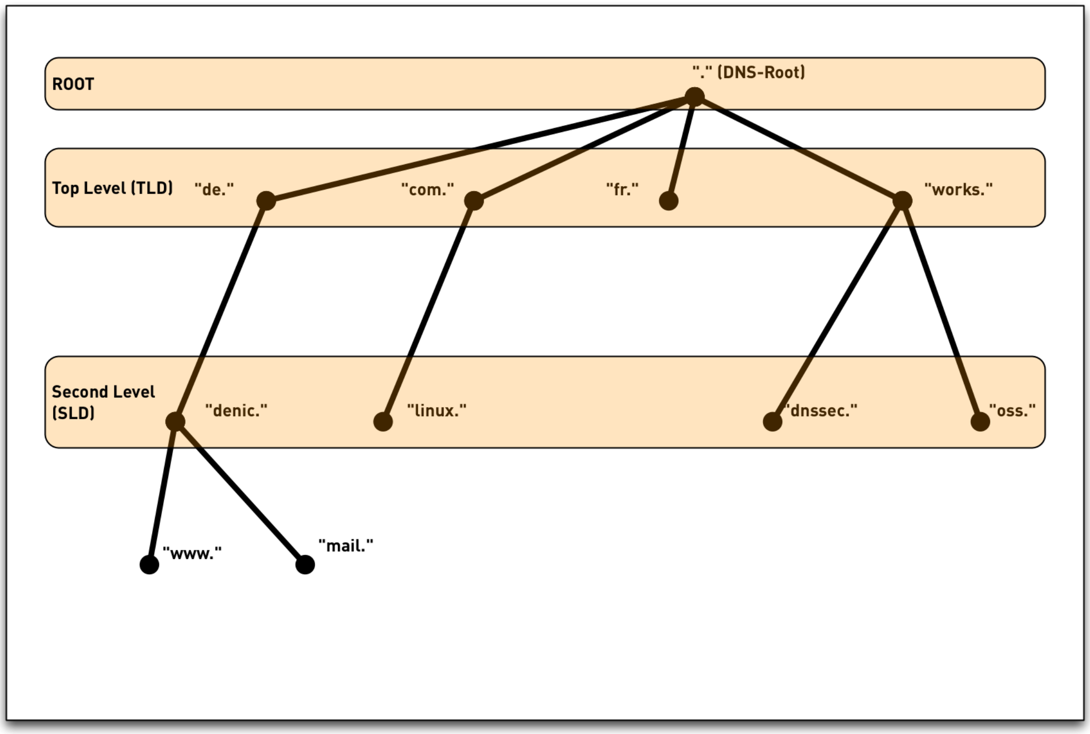
[sidebar] | *./img/DNS-namespace-06.png*



### 2.9. INTERNET NAMESPACE



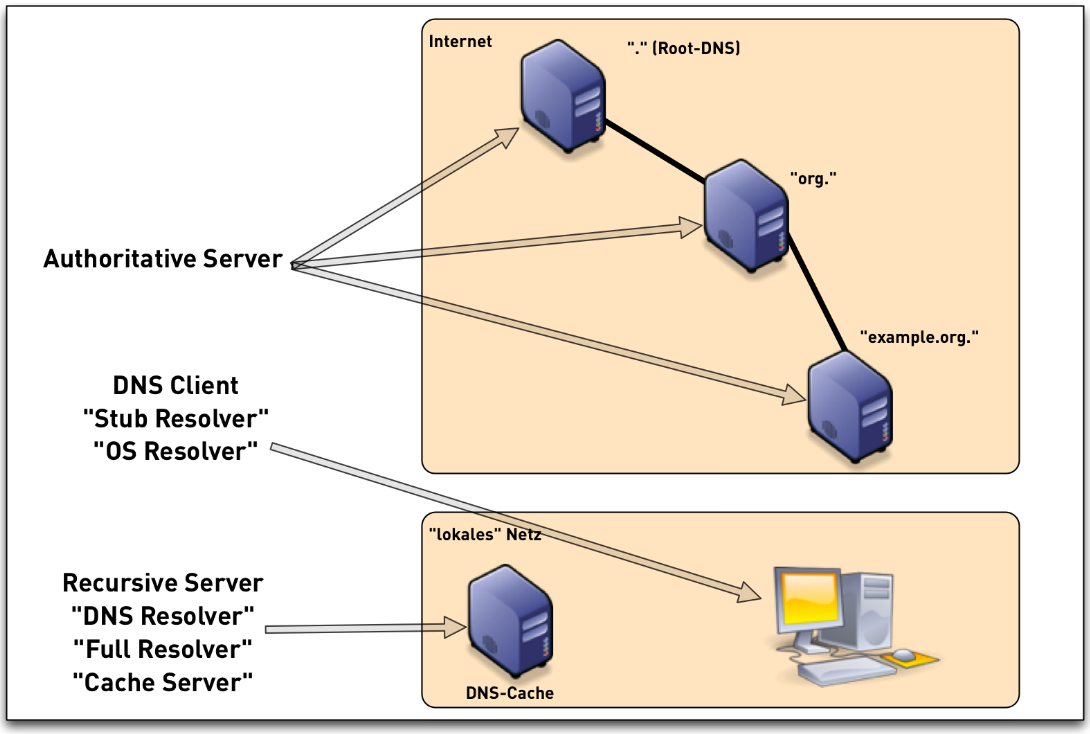
## 2.10. INTERNET NAMESPACE



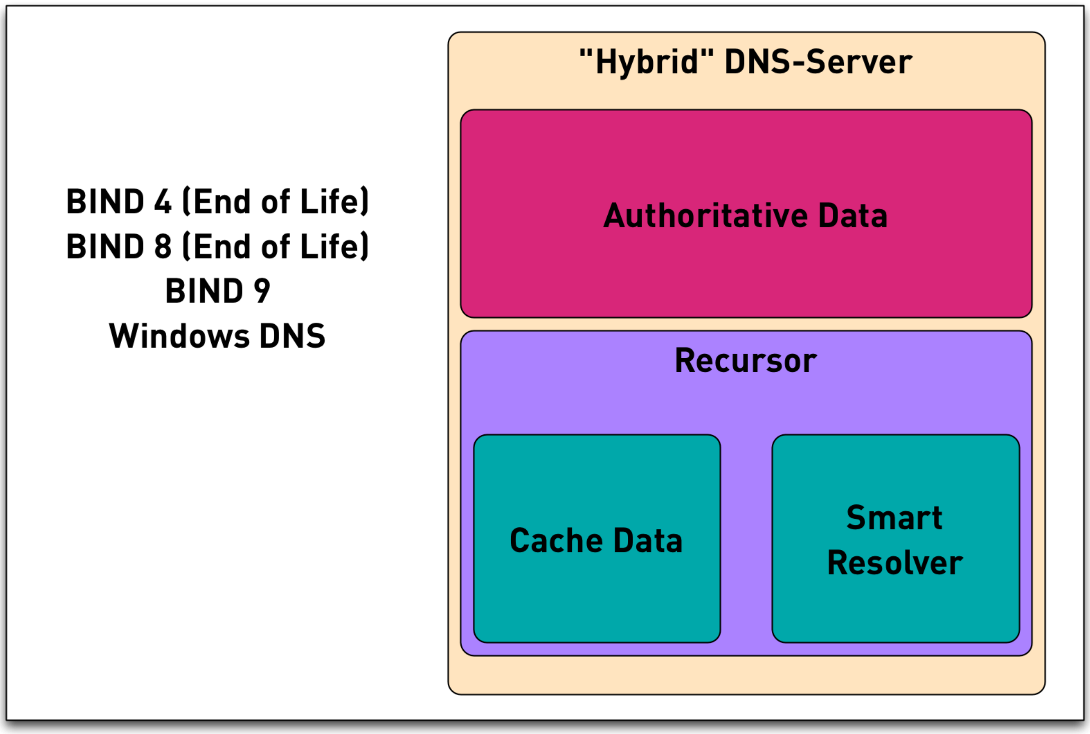
### 2.11. RFC 9476 - THE .ALT SPECIAL-USE TOP-LEVEL DOMAIN

- The `.alt` TLD has been created for all applications that want to use a DNS style name resolution, but don't use the DNS protocol
- Applications that use DNS style names without the DNS protocol should use names under the `.alt` TLD to not collide with DNS names
- RFC 9476: https://www.rfc-editor.org/rfc/rfc9476.html [https://www.rfc-editor.org/rfc/rfc9476.html]

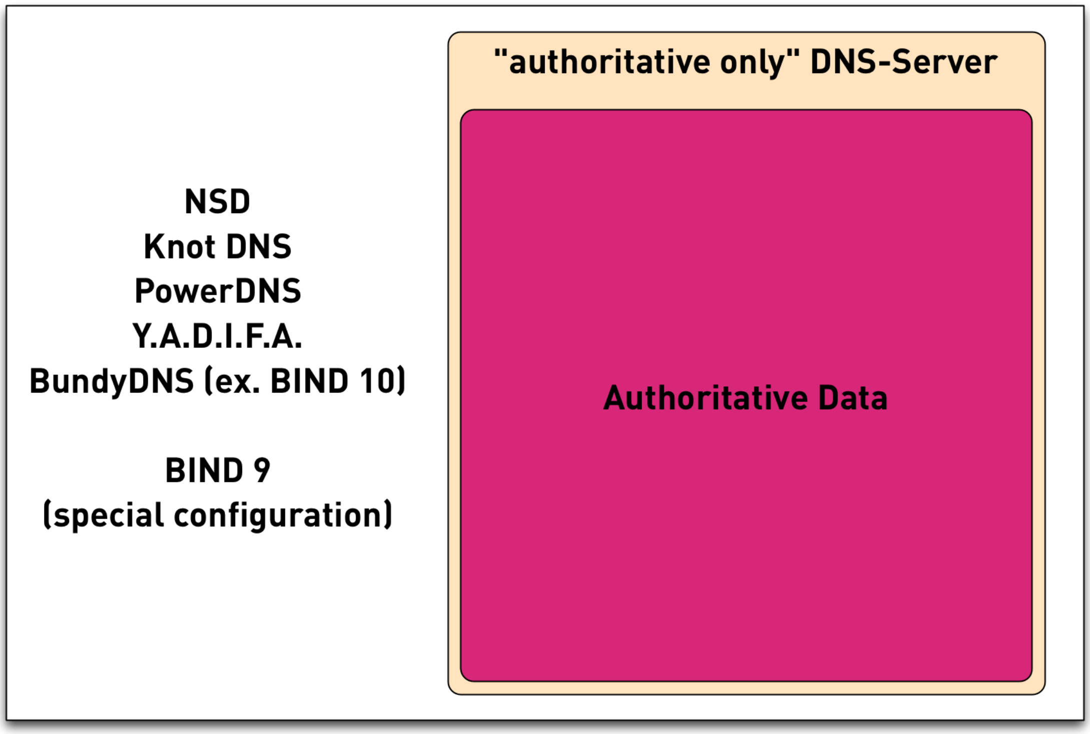
### 2.12. DNS COMPONENTS



## 2.13. DNS COMPONENTS - HYBRID-DNS



## 2.14. DNS COMPONENTS - AUTHORITATIVE ONLY



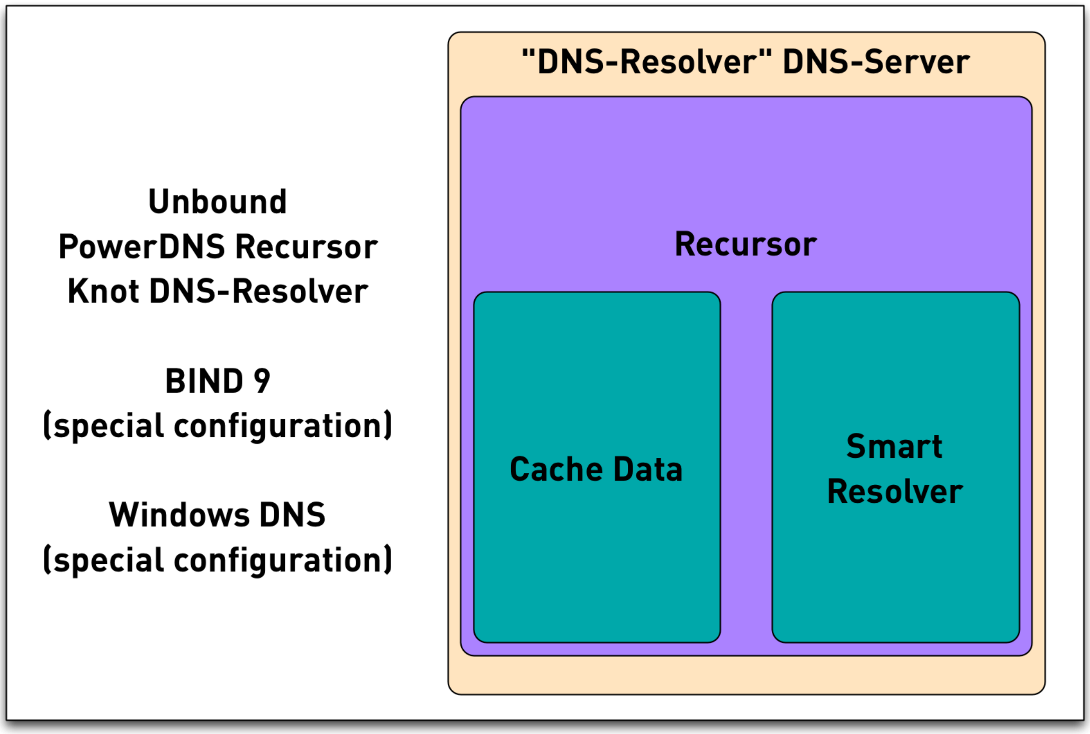
NSD
Knot DNS
PowerDNS
Y.A.D.I.F.A.
BundyDNS (ex. BIND 10)

BIND 9
(special configuration)

"authoritative only" DNS-Server

Authoritative Data

2.15. DNS COMPONENTS - DNS RESOLVER

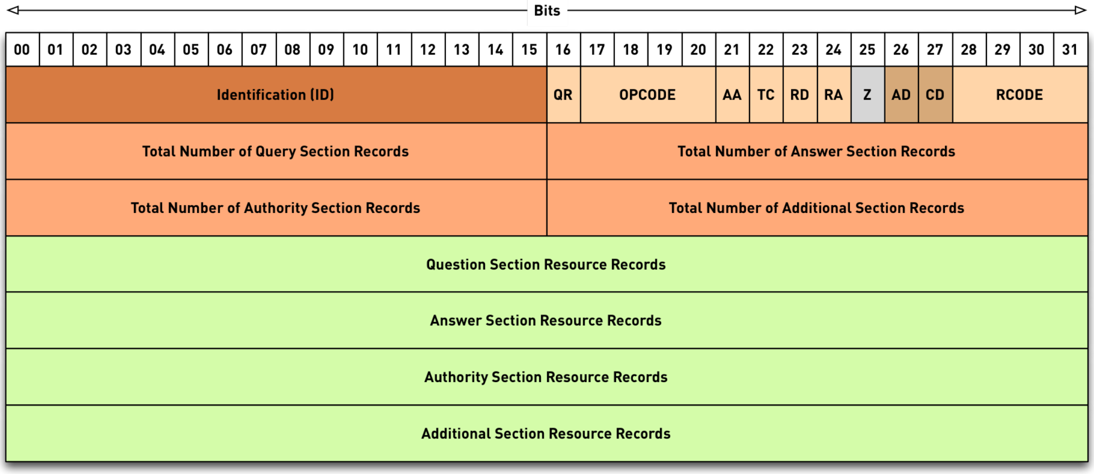


Unbound
PowerDNS Recursor
Knot DNS-Resolver

BIND 9
(special configuration)

Windows DNS
(special configuration)

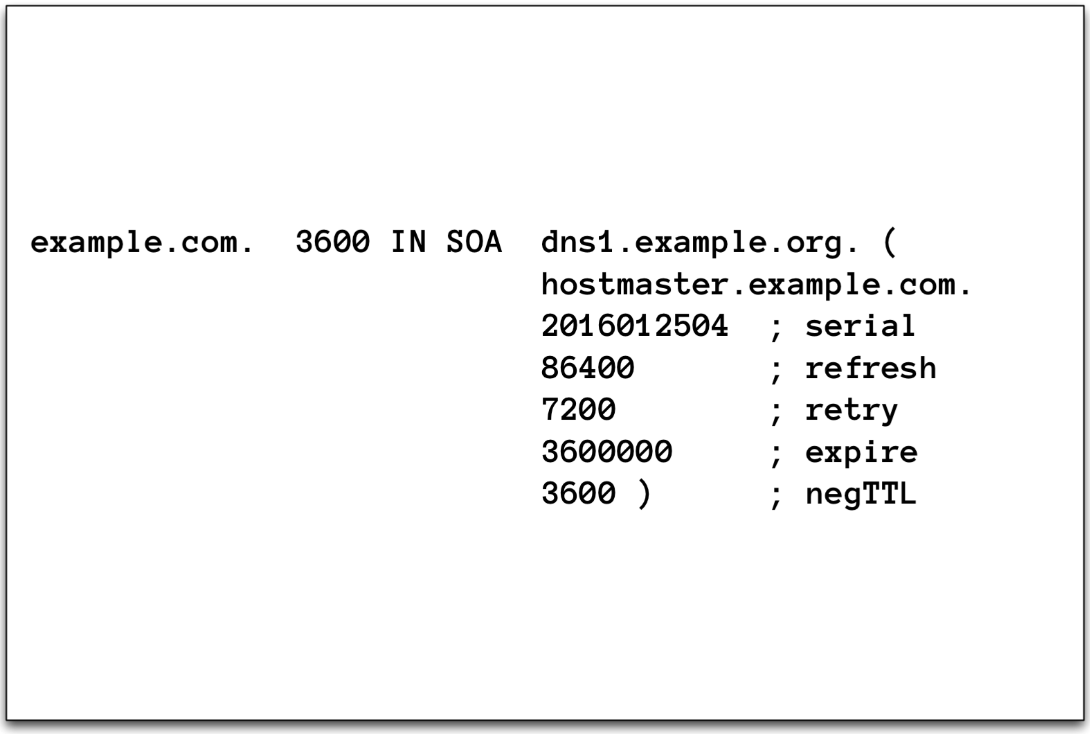
## 2.16. DNS MESSAGE FORMAT



<table>
<tr><td colspan="32">Bits</td></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>

## 2.17. DNS RESOURCE RECORDS

### 2.17.1. SOA Record

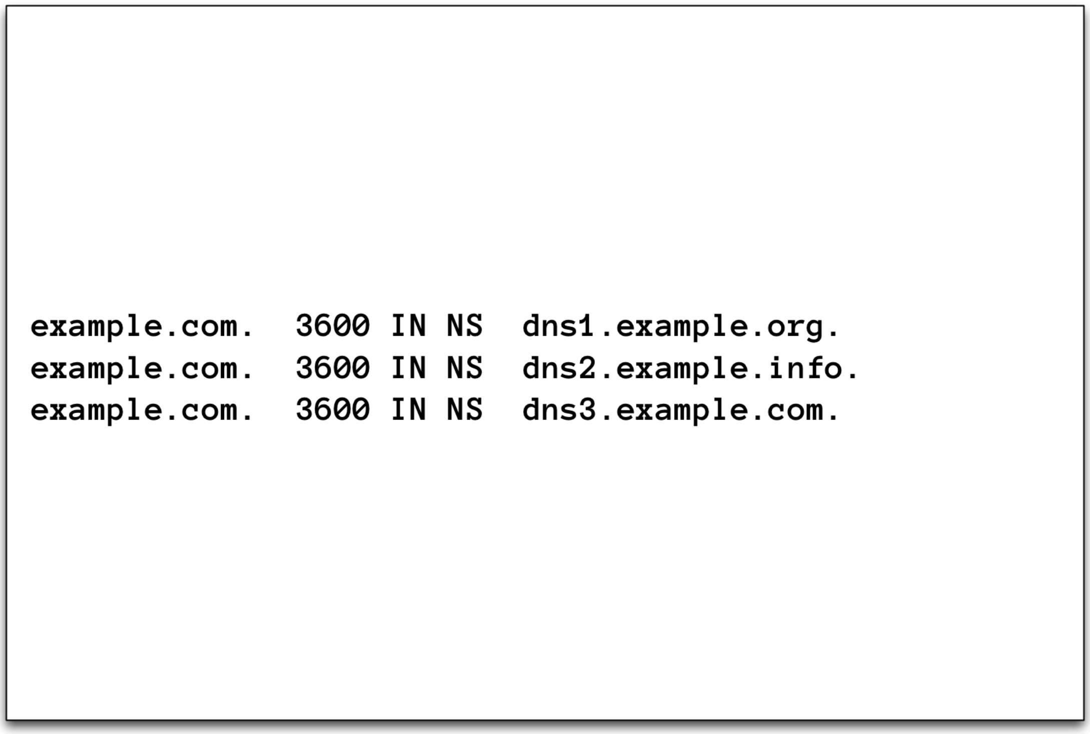


```
example.com.  3600 IN SOA  dns1.example.org. (
                           hostmaster.example.com.
                           2016012504   ; serial
                           86400        ; refresh
                           7200         ; retry
                           3600000      ; expire
  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

### 2.17.2. 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)
- RFC 9520 - Negative Caching of DNS Resolution Failures [https://www.rfc-editor.org/rfc/rfc9520.html] (December 2023) mandates that DNS resolution failures MUST be cached between 1 second (min) and 5 minutes (max)

### 2.17.3. NS record

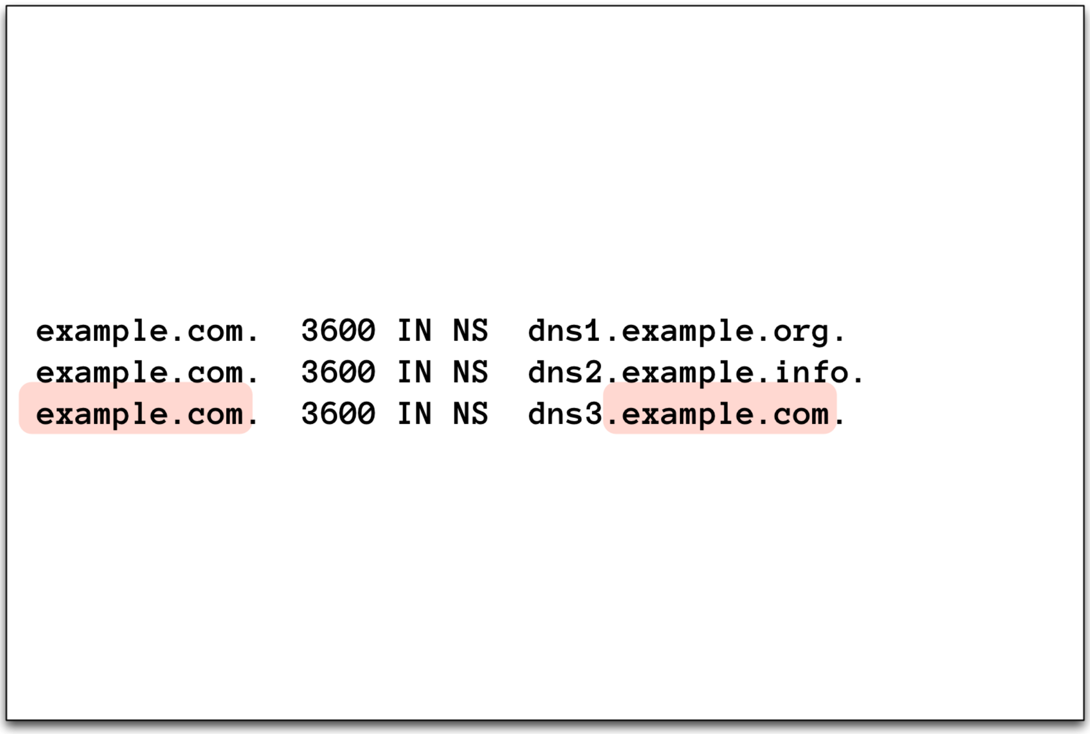


```
example.com.  3600 IN NS  dns1.example.org.
example.com.  3600 IN NS  dns2.example.info.
example.com.  3600 IN NS  dns3.example.com.
```

### 2.17.4. NS record

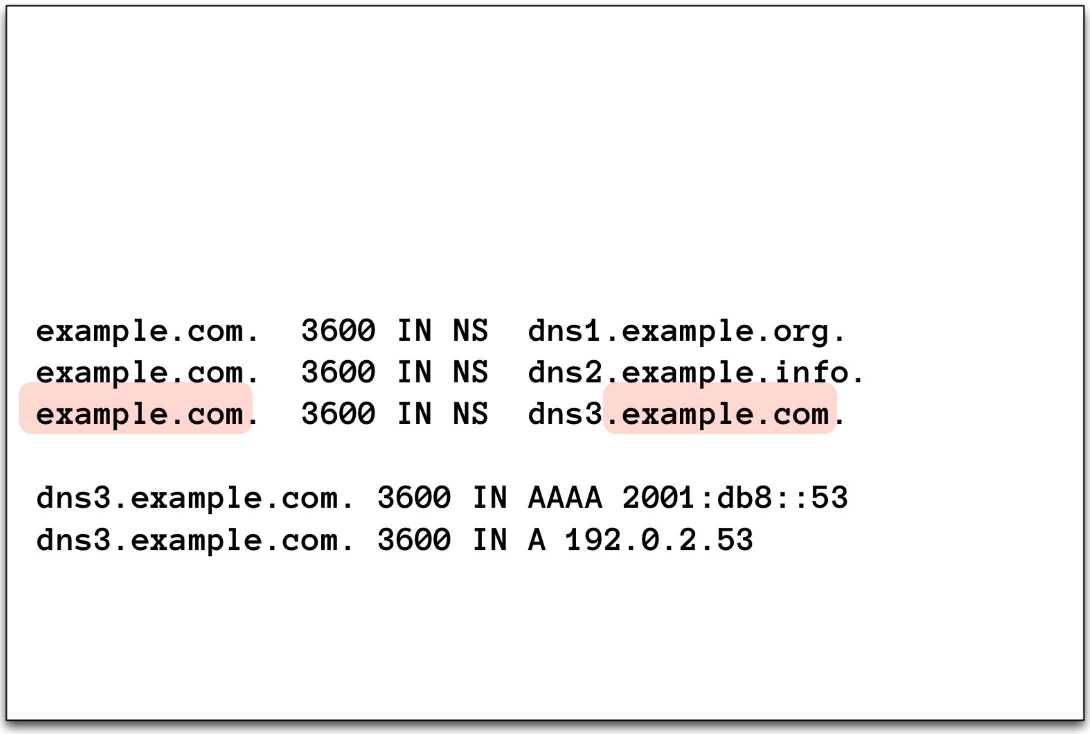
- NS records create the DNS delegation
- The NS record in the parent zone (delegation) SHOULD always match the NS records in the (delegated) zone (see *When parents and children disagree*)
- For domain names of authoritative DNS servers inside the delegated domain, the parent zone must be augmented with *Glue* records (A/AAAA records of the authoritative DNS Servers) to make the delegation work.
- See RFC 9471 - DNS Glue Requirements in Referral Responses [https://www.rfc-editor.org/rfc/rfc9471.html] (September 2023) for details.

### 2.17.5. 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.
```

2.17.6. Glue (2)



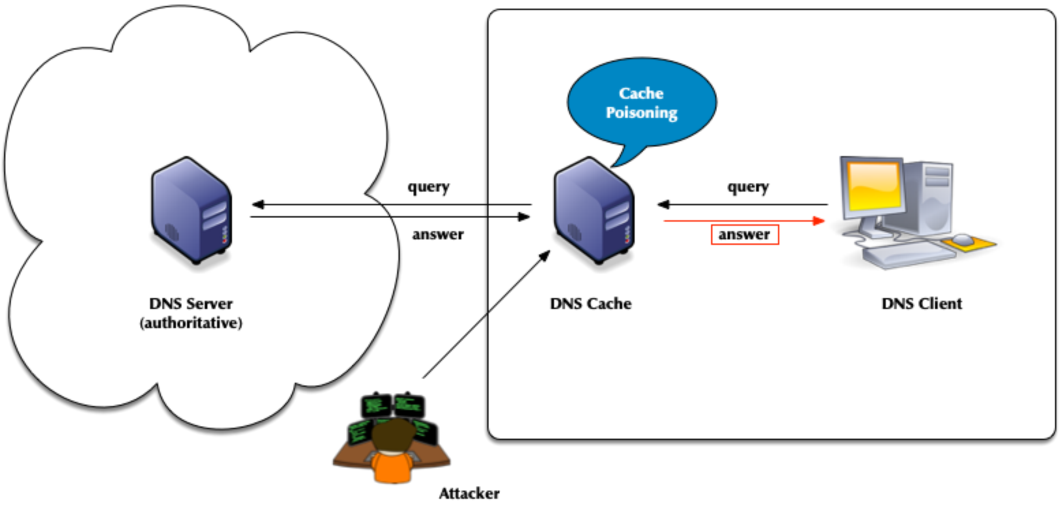
```
example.com.  3600 IN NS  dns1.example.org.
example.com.  3600 IN NS  dns2.example.info.
example.com.  3600 IN NS  dns3.example.com.

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

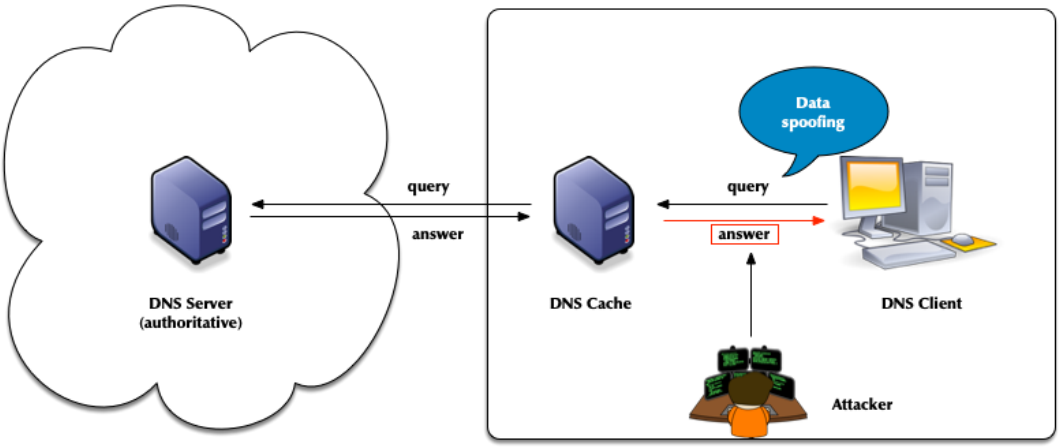
# CHAPTER 3. AN INTRO TO DNSSEC

## 3.1. DNS SECURITY (OR LACK OF)

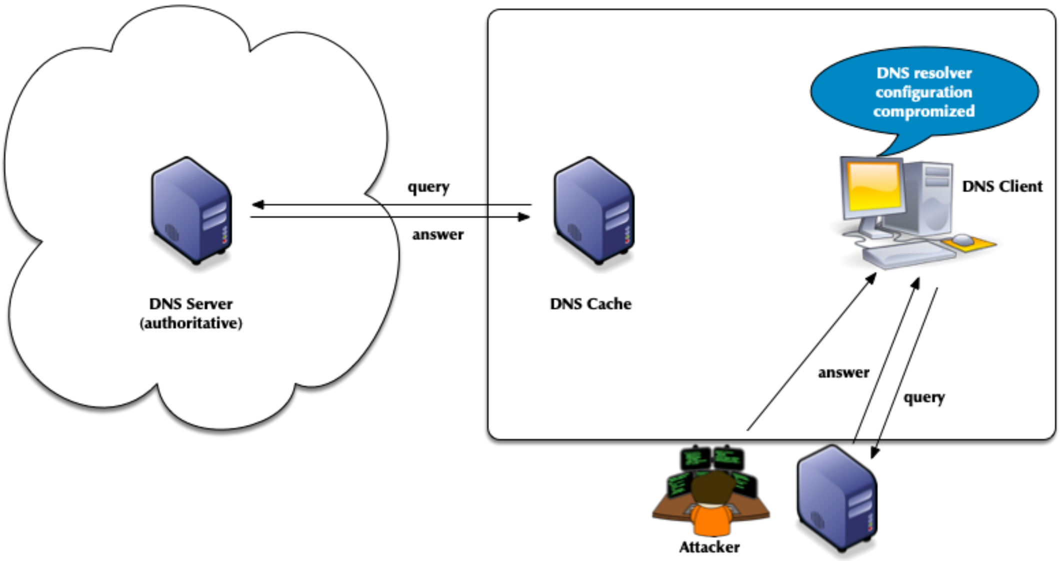
- The classic DNS from 1983 has not been designed with security in mind
- Attack vector: DNS cache poisoning



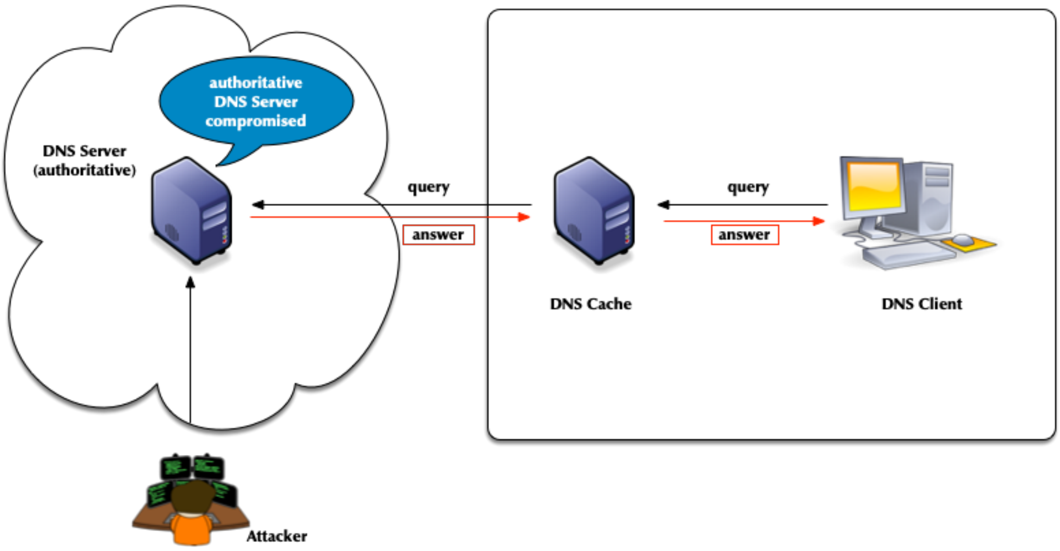
- Attack vector: Men-in-the-Middle data spoofing



- 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
DNS Cache
query
answer
DNS Client
Attacker

- DNSSEC can help

WITH DNSSEC
THIS WOULD
NOT HAVE
HAPPENED

# CHAPTER 4. CRYPTOGRAPHY IN DNS (SHORT)

## 4.1. 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]
  - RFC 8484 "DNS Queries over HTTPS (DoH)" (DNS-over-HTTPS) [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

## 4.2. 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).

#### 4.2.1. 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.

#### 4.2.2. 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) or 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**.

#### 4.2.3. Message Authentication Codes

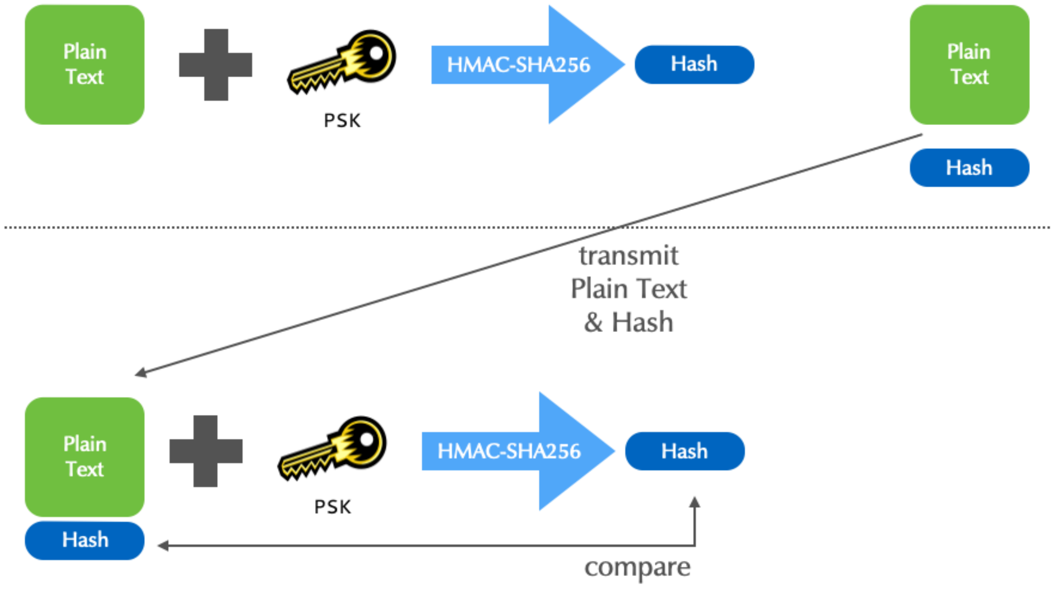
- Hashing, with a symmetric key added to the input, efficiently provides **integrity** and **authenticity**.

  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`?"

### 4.3. 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.

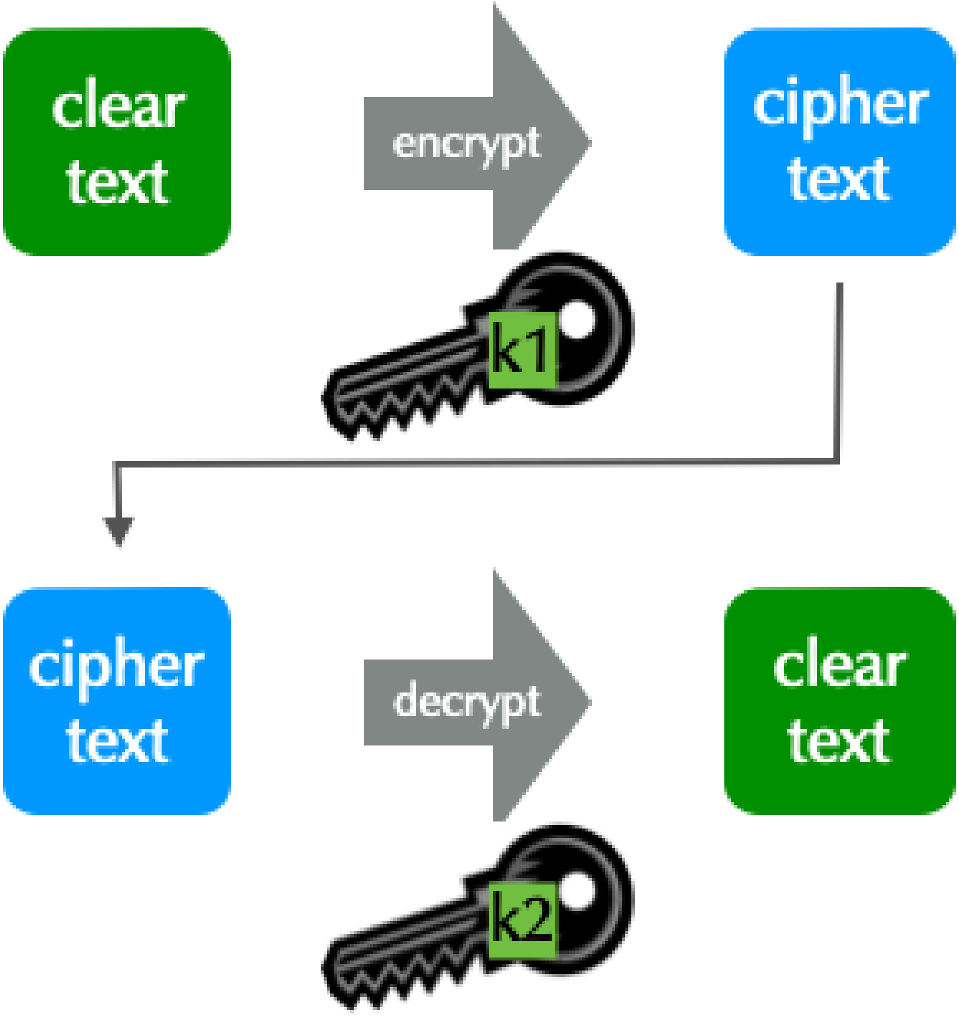
  A key cannot decrypt what it encrypted!

  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.



### 4.3.1. 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

### 4.3.2. 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.

### 4.3.3. PK Application 2: Encryption for Authenticity: Digital Signatures

- For efficiency, the message itself is not encrypted.

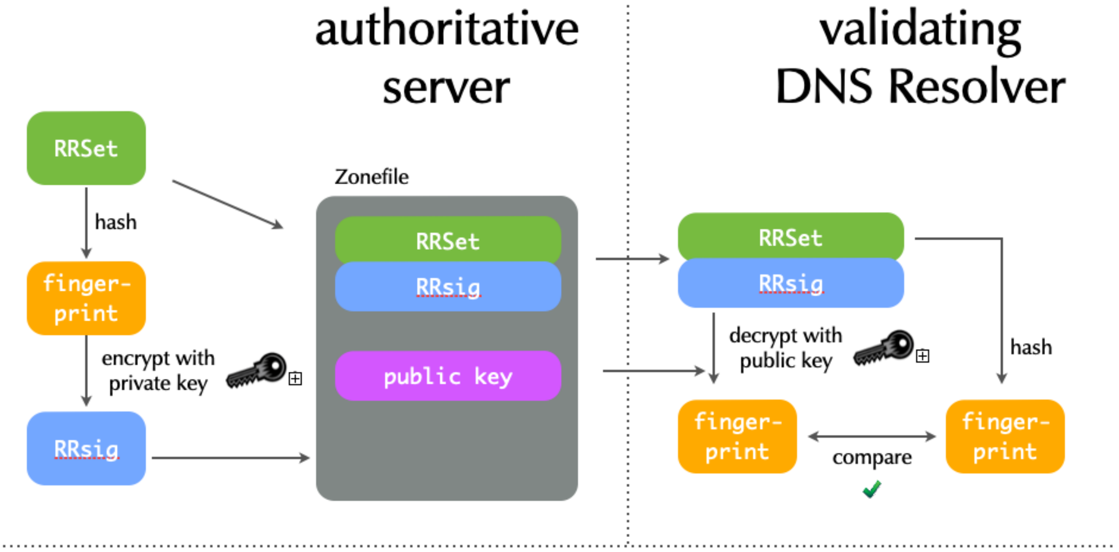
  The message is first hashed to create a fingerprint.

  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.

### 4.3.4. Digital Signatures in DNSSEC



### 4.4. 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: Authoritative server and validating resolver
  - NSD from NLnet Labs: Fast authoritative server
  - Unbound from NLnet Labs :Fast and secure validating resolver
  - Windows 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`

### 4.5. 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)

### 4.6. 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 5. 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.

## 5.1. DNSSEC RECORDS

### 5.1.1. 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.

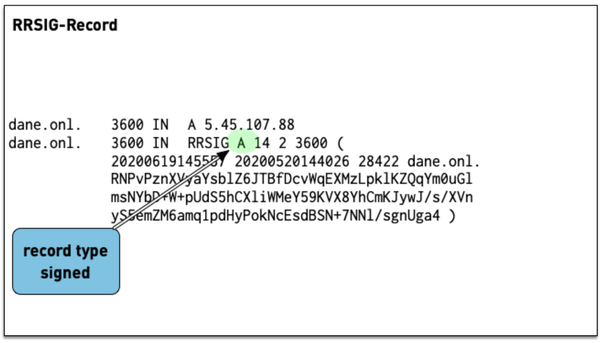
    The signatures are added to the zone as RRSIG RRs.

    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 less apart).

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

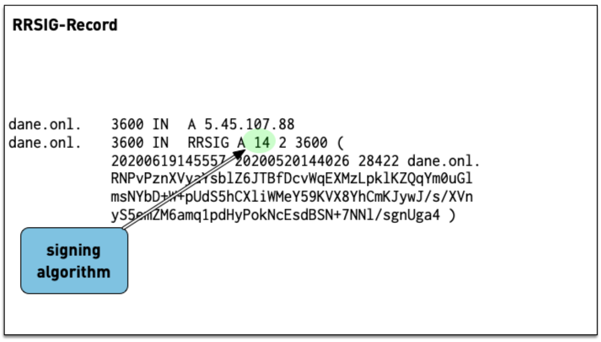
    Keys don't have expiration timers.



**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 )
```

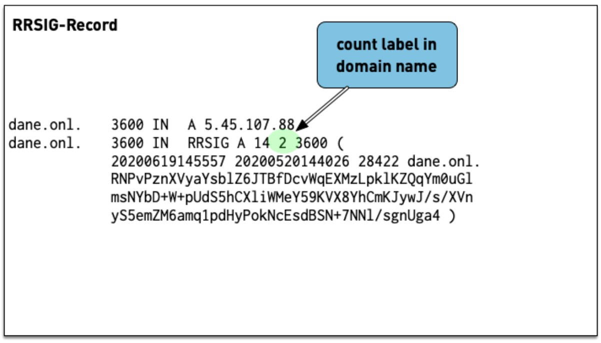
record type signed



**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 )
```

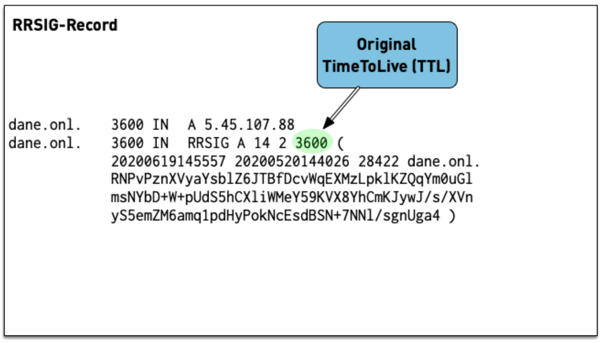
signing algorithm



**RRSIG-Record**

count label in domain name

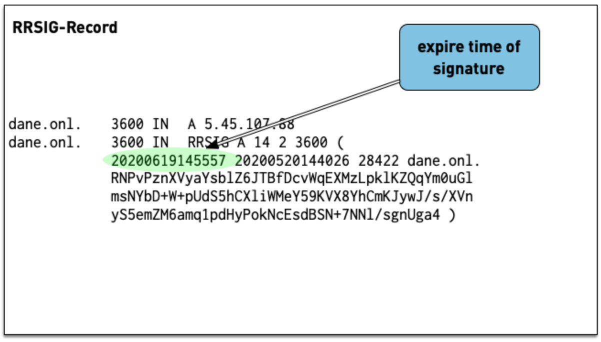
```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4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



RRSIG-Record

Original TimeToLive (TTL)

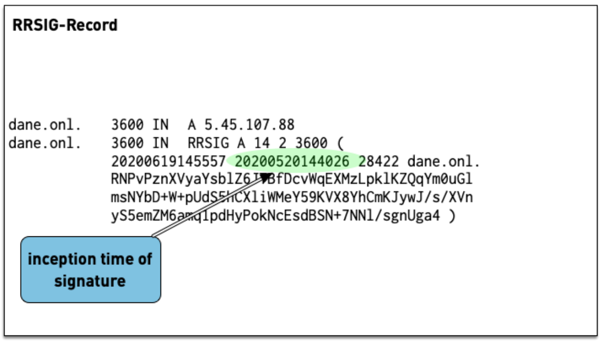
```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4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



RRSIG-Record

expire time of signature

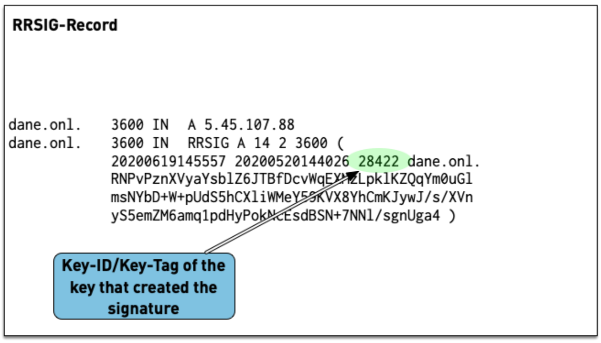
```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4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



RRSIG-Record

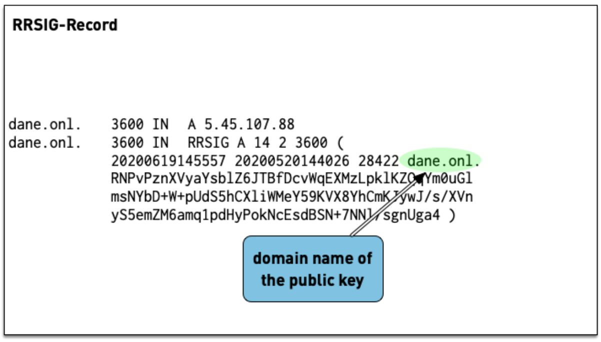
```
dane.onl.     3600 IN   A 5.45.107.88
dane.onl.     3600 IN   RRSIG A 14 2 3600 (
              20200619145557 20200520144026 28422 dane.onl.
              RNPvPznXVyaYsblZ6JTBfDcvWqEXMzLpklKZQqYm0uGl
              msNYbD+W+pUdS5hCXliWMeY59KVX8YhCmKJywJ/s/XVn
              yS5emZM6amq1pdHyPokNcEsdBSN+7NNl/sgnUga4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

inception time of signature



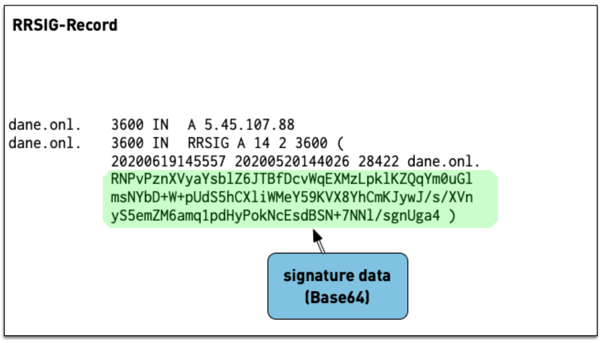
RRSIG-Record

```
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 )
```



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 )
```



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 )
```

- Work on the DNS resolver machine

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

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

- Answer these questions

  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?

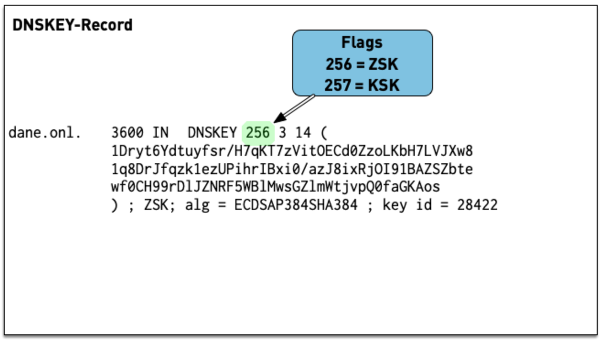
### 5.1.2. DNSKEY

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

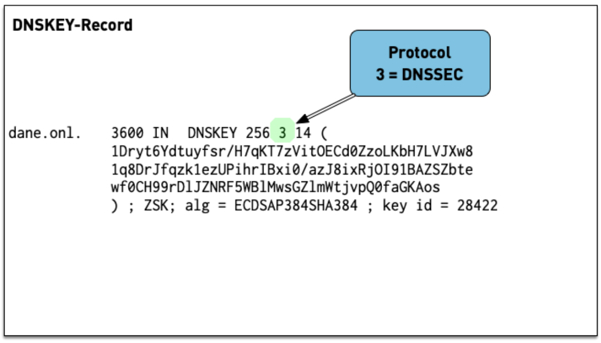
  The private key is not publicly available or accessible through DNS.

- 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.

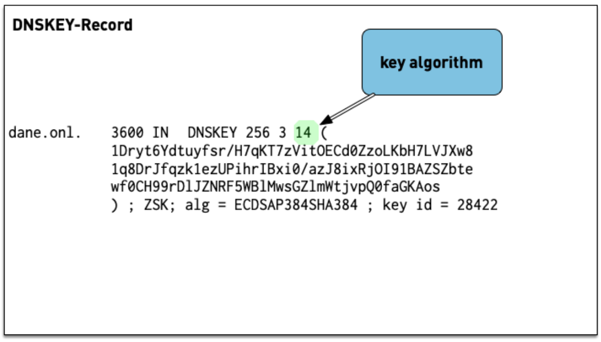


```
dane.onl.     3600 IN  DNSKEY 256 3 14 (
              1Dryt6Ydtuyfsr/H7qKT7zVitOECd0ZzoLKbH7LVJXw8
              1q8DrJfqzk1ezUPihrIBxi0/azJ8ixRjOI91BAZSZbte
              wf0CH99rDlJZNRF5WBlMwsGZlmWtjvpQ0faGKAos
              ) ; ZSK; alg = ECDSAP384SHA384 ; key id = 28422
```



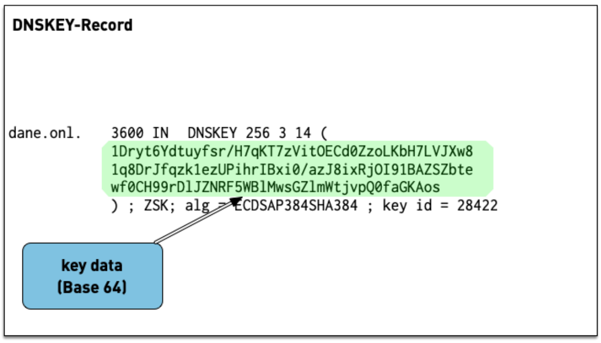
DNSKEY-Record

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



DNSKEY-Record

```
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
```

- Work on the DNS resolver machine

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

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

- Answer these questions

  Which algorithm is this zone using?

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

  The sizes of both DNS answer messages as reported by `dig`

### 5.2. 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

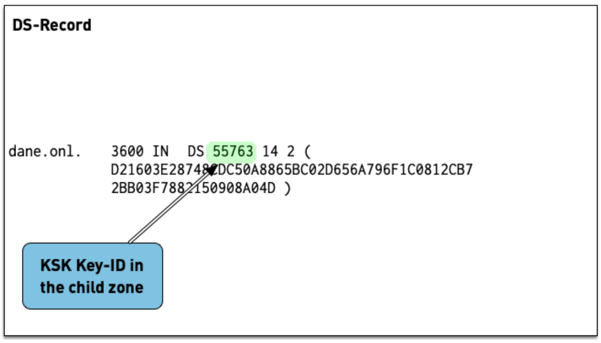
#### 5.2.1. DS Record

- 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)

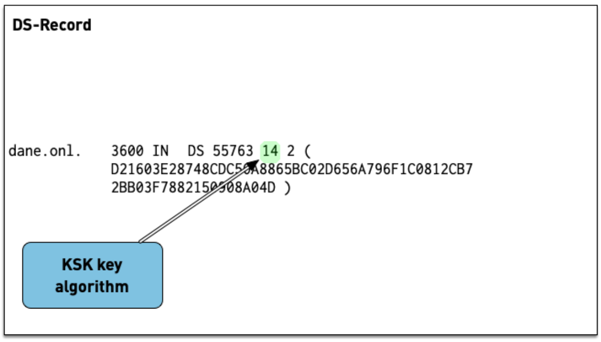
    This interface can become a target for attacker that want to change data in a DNSSEC signed zone and should be protected (two factor signon, registry lock etc).



**DS-Record**

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

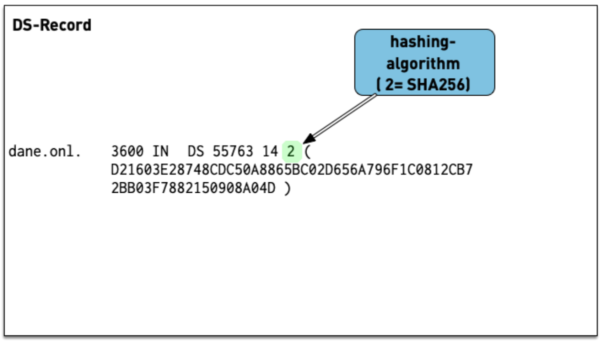
KSK Key-ID in the child zone



**DS-Record**

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

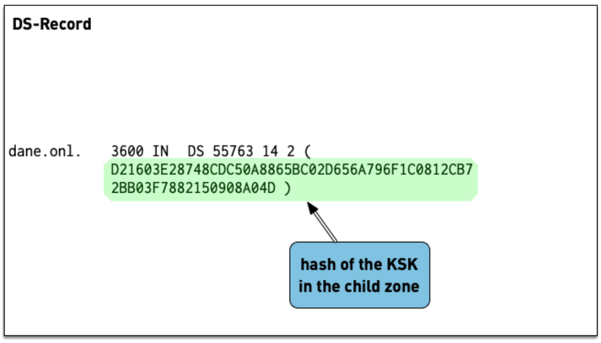
KSK key algorithm



**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`:

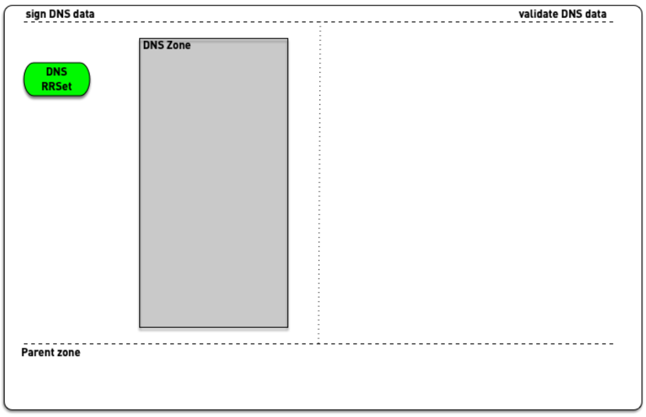
```
$ dig isc.org DS +dnssec +multi
```

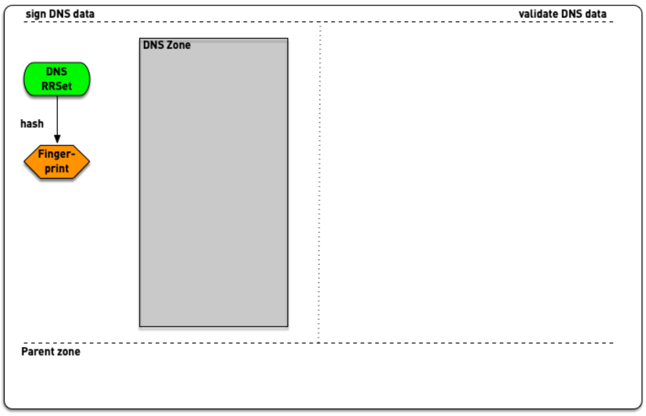
- Answer these questions into the chat

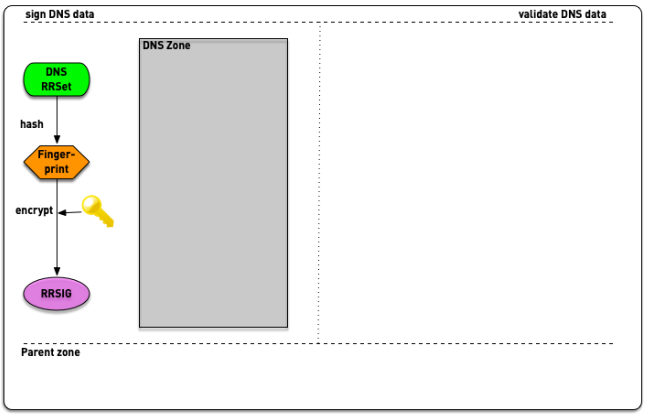
  Which DNSSEC-Key-Algorithm is this zone using?

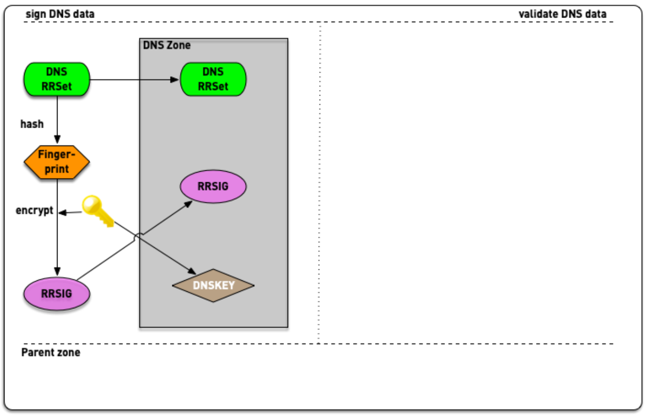
  What is the current key-id of the KSK in the zone `isc.org`?

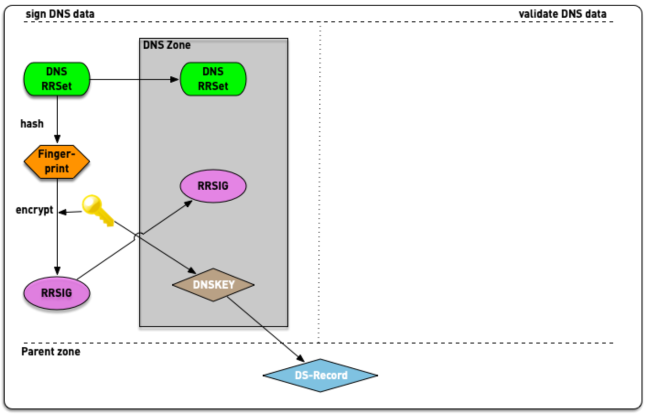
  What hashing algorithm is used for the DS record of `isc.org` check against Delegation Signer (DS) Resource Record (RR) Type Digest Algorithms [https://www.iana.org/assignments/ds-rr-types/ds-rr-types.xhtml]?

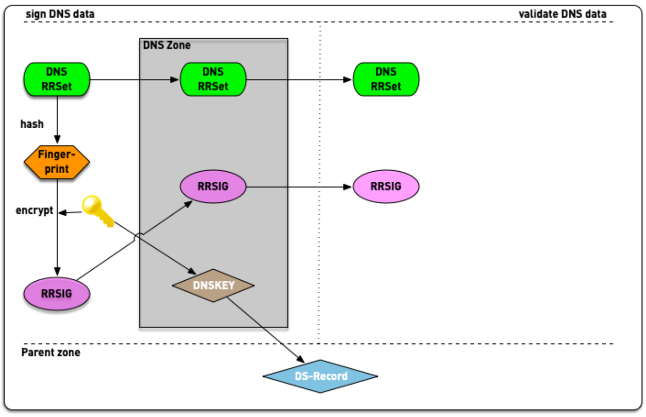

sign DNS data
validate DNS data
DNS RRSet
DNS Zone
Parent zone

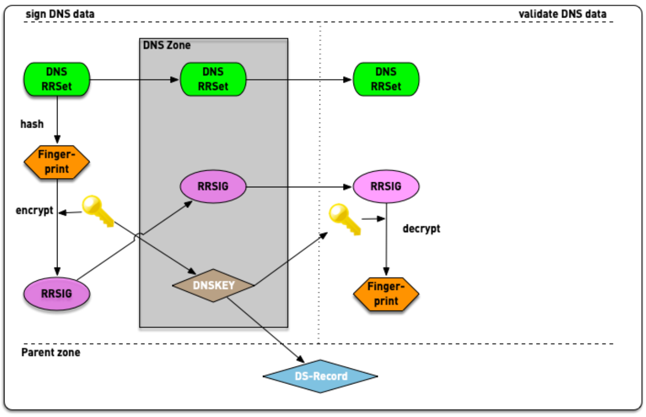

sign DNS data
validate DNS data
DNS RRSet
hash
Finger-print
DNS Zone
Parent zone

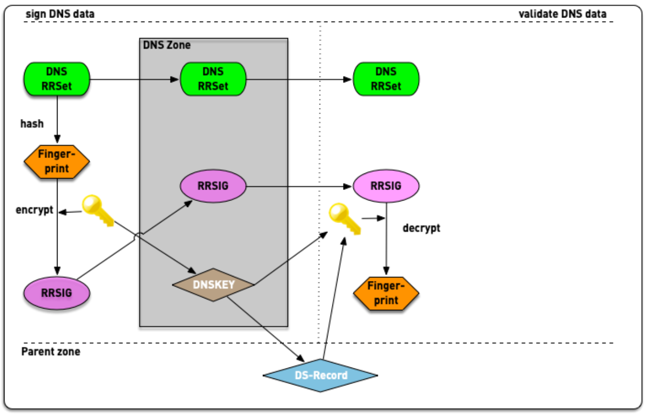

sign DNS data
validate DNS data
DNS RRSet
hash
Finger-print
encrypt
RRSIG
DNS Zone
Parent zone

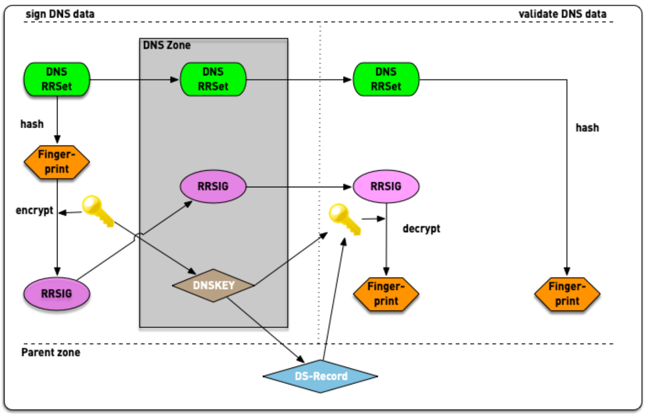

sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
hash
Finger-print
encrypt
RRSIG
RRSIG
DNSKEY
Parent zone

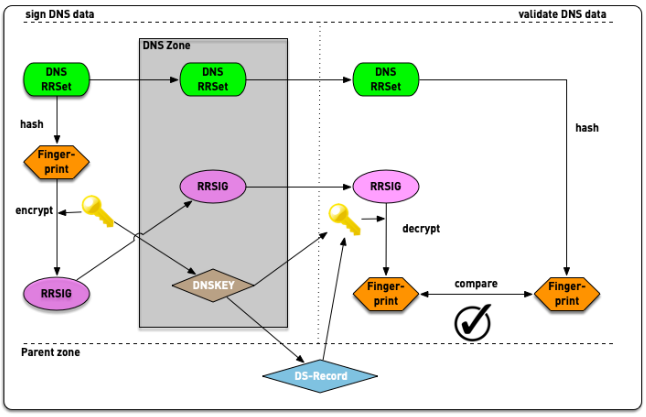

sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
hash
Finger-print
encrypt
RRSIG
RRSIG
DNSKEY
Parent zone
DS-Record


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


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


sign DNS data
validate DNS data
DNS Zone
DNS RRSet
DNS RRSet
DNS RRSet
hash
Finger-print
encrypt
RRSIG
RRSIG
RRSIG
decrypt
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
encrypt
RRSIG
RRSIG
RRSIG
decrypt
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 7. DNSSEC VALIDATION

## 7.1. 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
```

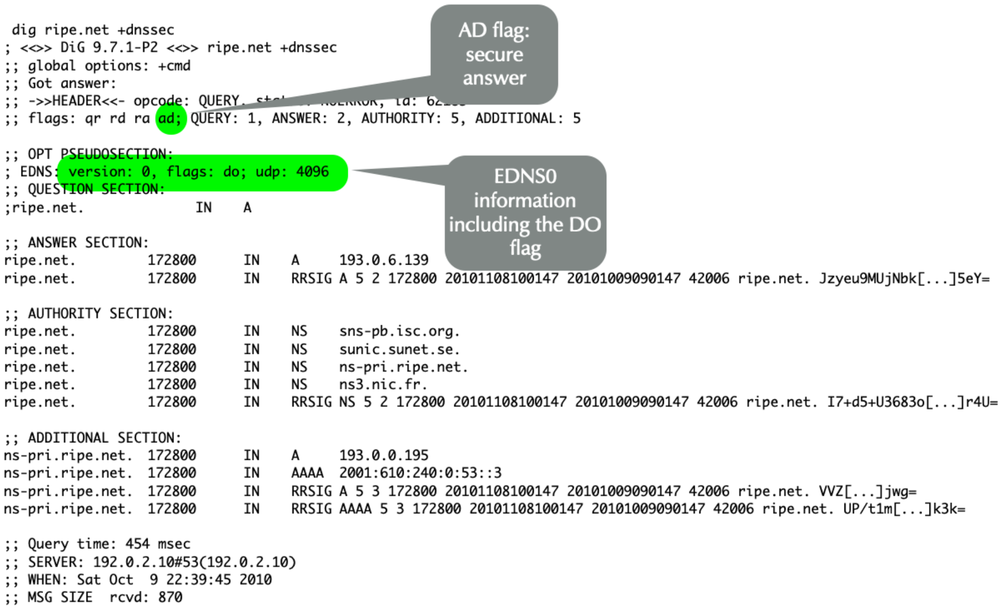
- `CO` Flag - Compact Answers OK

  *Compact Denial of Existence* is a new DNSSEC extension that allows *authenticated Denial of Existence* with `NSEC` (and optionally `NSEC3`) for online signer it also prevents *Zone Walking*, even with `NSEC` records

the `CO` Flag is send by an DNS resolver towards authoritative DNS-servers (primary or secondary) to inform the receiver that the sender does understand the new *compact answers*

the `CO` Flag is encoded inside the EDNS-Flag-space (alongside the `DO`-Flag)

Details: RFC 9824 Compact Denial of Existence in DNSSEC [https://tools.ietf.org/html/rfc9824] (September 2025)



```
 dig ripe.net +dnssec
; <<>> DiG 9.7.1-P2 <<>> ripe.net +dnssec
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 5

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

;; ANSWER SECTION:
ripe.net.         172800      IN    A     193.0.6.139
ripe.net.         172800      IN    RRSIG A 5 2 172800 20101108100147 20101009090147 42006 ripe.net. Jzyeu9MUjNbk[...]5eY=

;; AUTHORITY SECTION:
ripe.net.         172800      IN    NS    sns-pb.isc.org.
ripe.net.         172800      IN    NS    sunic.sunet.se.
ripe.net.         172800      IN    NS    ns-pri.ripe.net.
ripe.net.         172800      IN    NS    ns3.nic.fr.
ripe.net.         172800      IN    RRSIG NS 5 2 172800 20101108100147 20101009090147 42006 ripe.net. I7+d5+U3683o[...]r4U=

;; ADDITIONAL SECTION:
ns-pri.ripe.net.  172800      IN    A     193.0.0.195
ns-pri.ripe.net.  172800      IN    AAAA  2001:610:240:0:53::3
ns-pri.ripe.net.  172800      IN    RRSIG A 5 3 172800 20101108100147 20101009090147 42006 ripe.net. VVZ[...]jwg=
ns-pri.ripe.net.  172800      IN    RRSIG AAAA 5 3 172800 20101108100147 20101009090147 42006 ripe.net. UP/t1m[...]k3k=

;; Query time: 454 msec
;; SERVER: 192.0.2.10#53(192.0.2.10)
;; WHEN: Sat Oct  9 22:39:45 2010
;; MSG SIZE  rcvd: 870
```

## 7.2. 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.

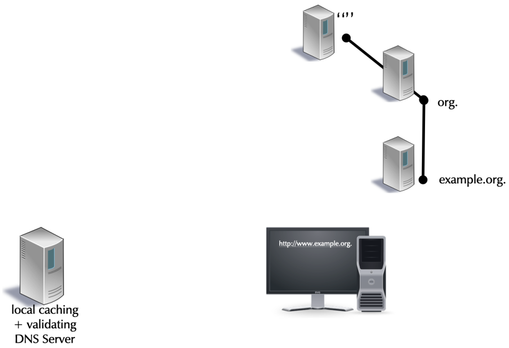
  If the chain of trust is broken the signature is ignored.

- The steps in the following animation are simplified.

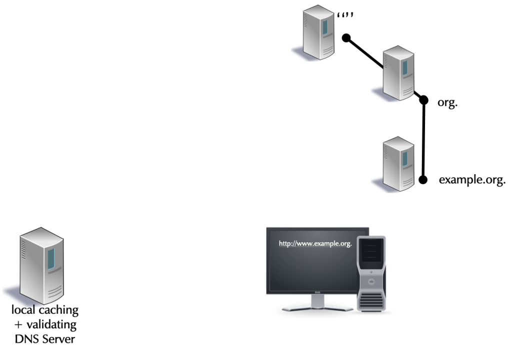
  It only shows validation using one key per zone (SSK/CSK).

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

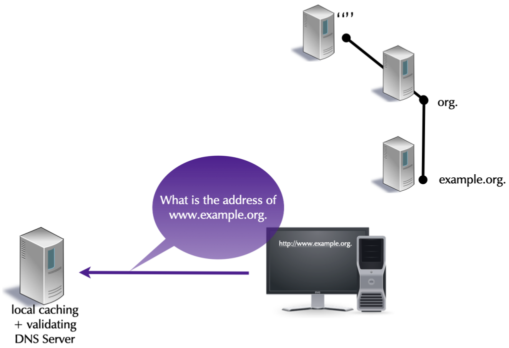
## DNSSEC Name Resolution



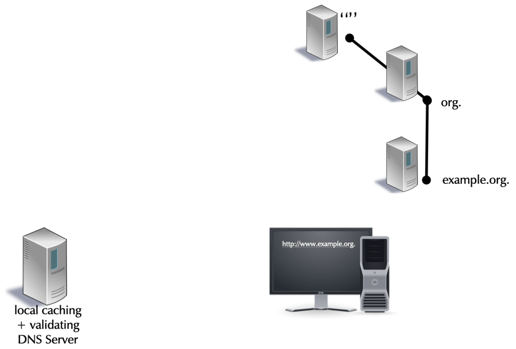
## DNSSEC Name Resolution



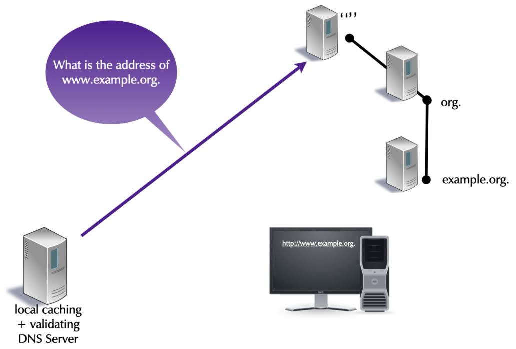
## DNSSEC Name Resolution



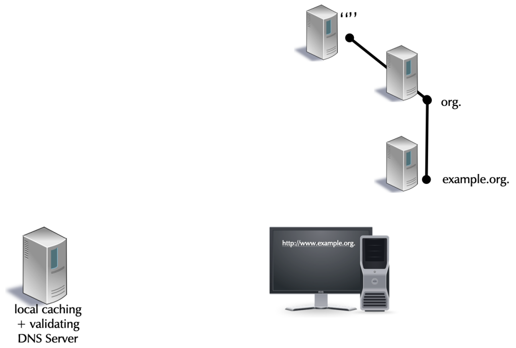
## DNSSEC Name Resolution



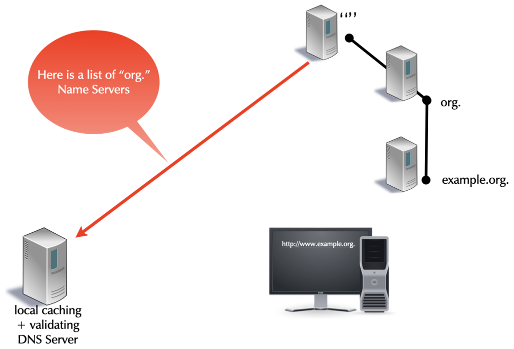
## DNSSEC Name Resolution



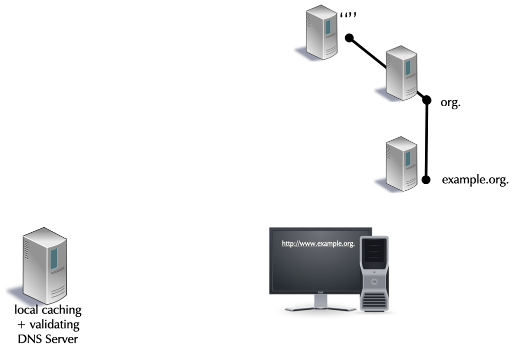
## DNSSEC Name Resolution



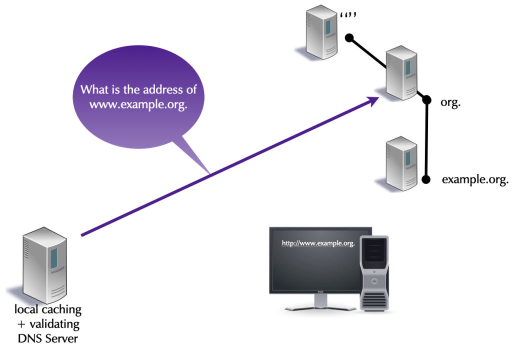
## DNSSEC Name Resolution



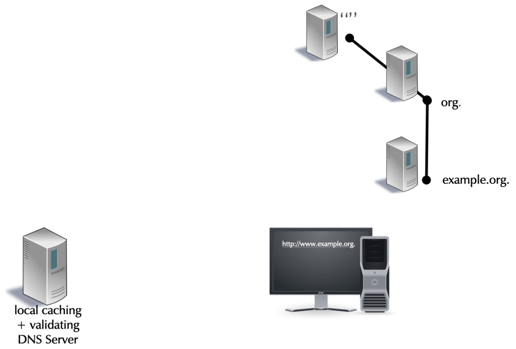
## DNSSEC Name Resolution



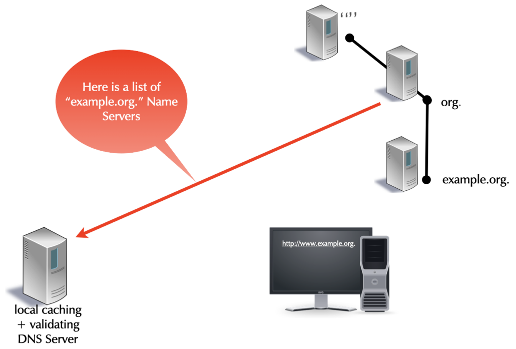
## DNSSEC Name Resolution



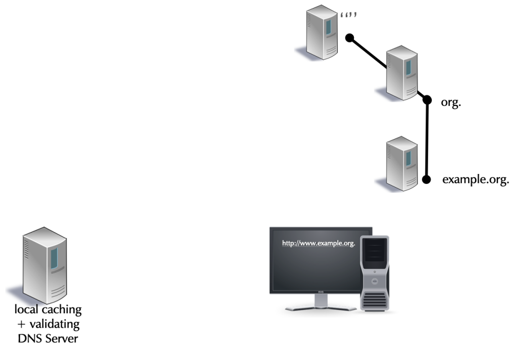
## DNSSEC Name Resolution



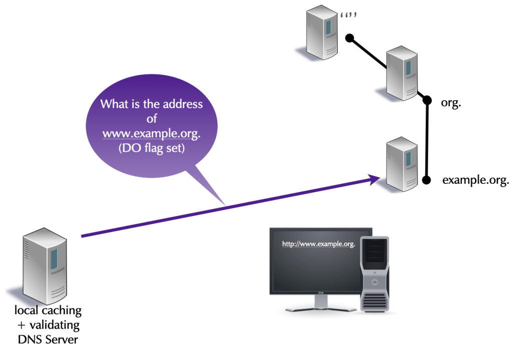
## DNSSEC Name Resolution



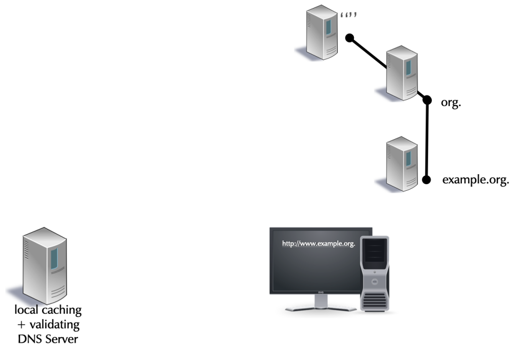
## DNSSEC Name Resolution



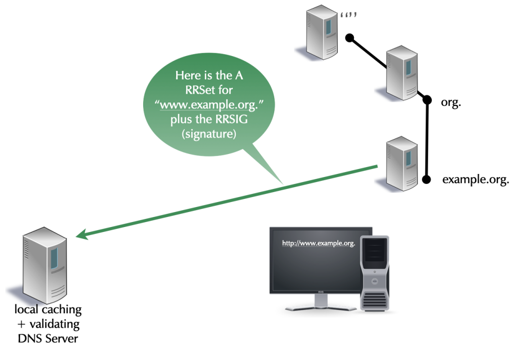
# DNSSEC Name Resolution


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

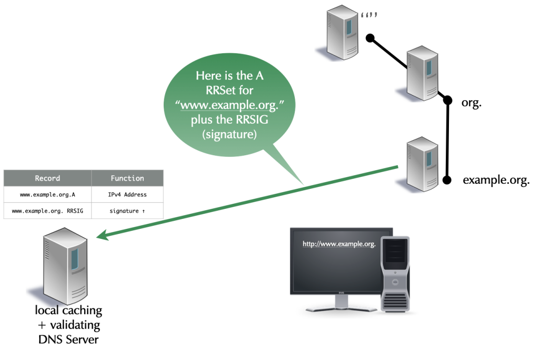
# DNSSEC Name Resolution



# 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

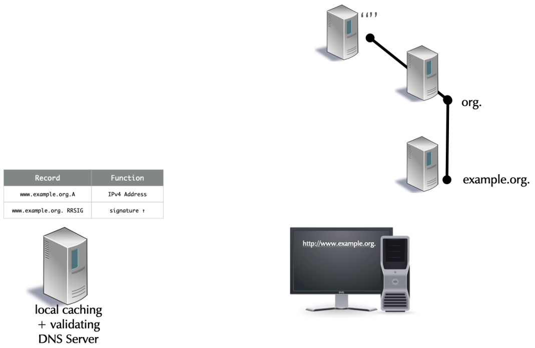
# DNSSEC Name Resolution



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

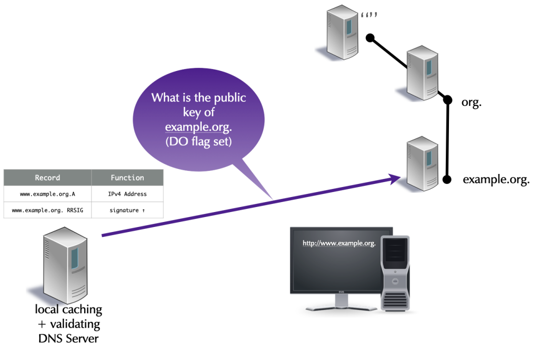
"."
org.
example.org.
Here is the A RRSet for "www.example.org." plus the RRSIG (signature)
http://www.example.org.
local caching + validating DNS Server

# DNSSEC Name Resolution



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

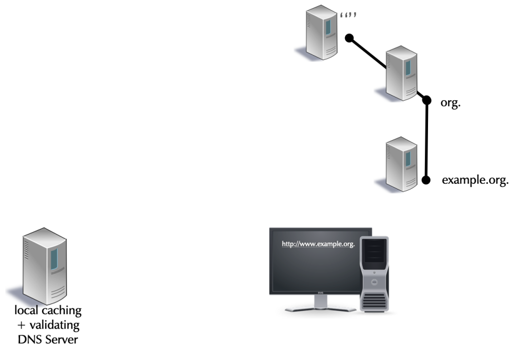
# DNSSEC Name Resolution



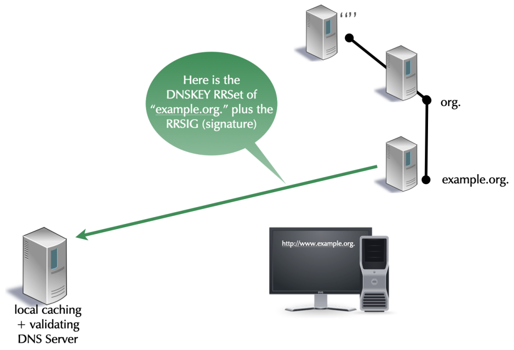
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |

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

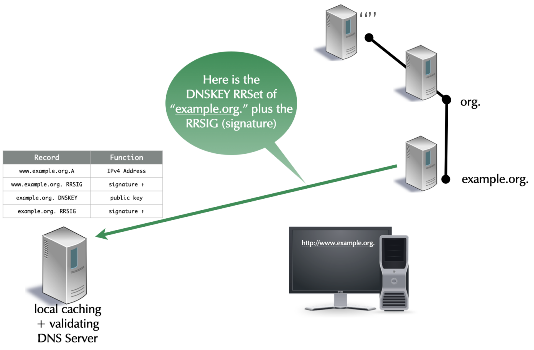
# DNSSEC Name Resolution



# DNSSEC Name Resolution

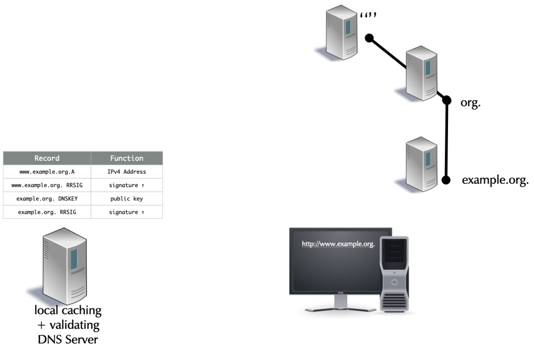


# DNSSEC Name Resolution



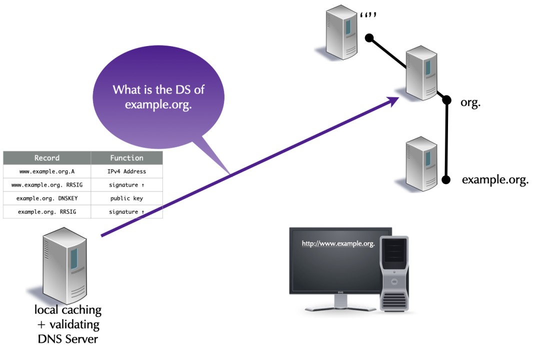
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |

## DNSSEC Name Resolution



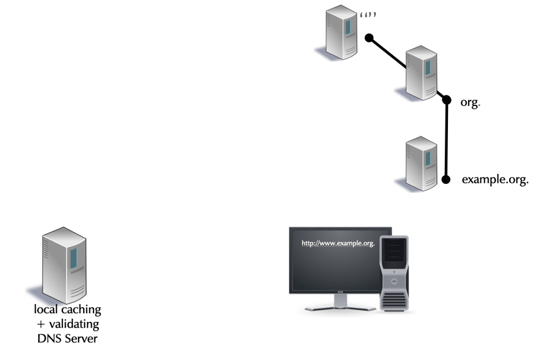
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |

## DNSSEC Name Resolution

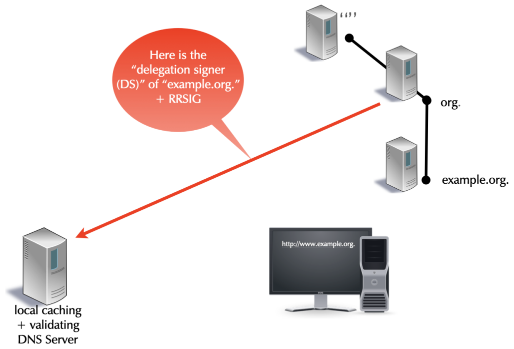


| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |

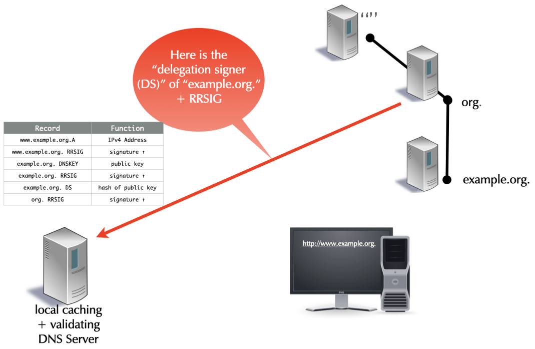
## DNSSEC Name Resolution



## DNSSEC Name Resolution

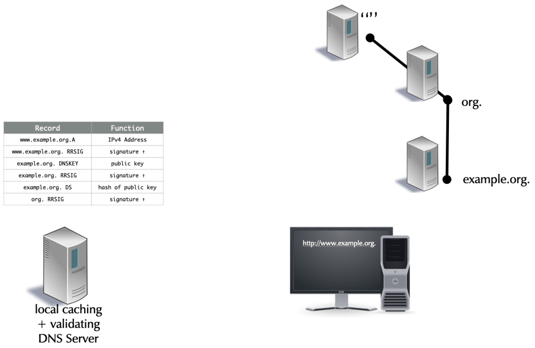

"."
Here is the "delegation signer (DS)" of "example.org." + RRSIG
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

## DNSSEC Name Resolution


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

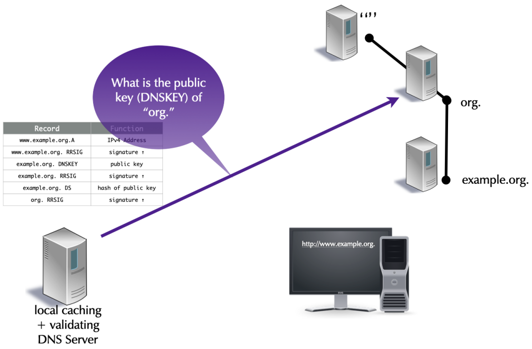
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |

## 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 ↑ |

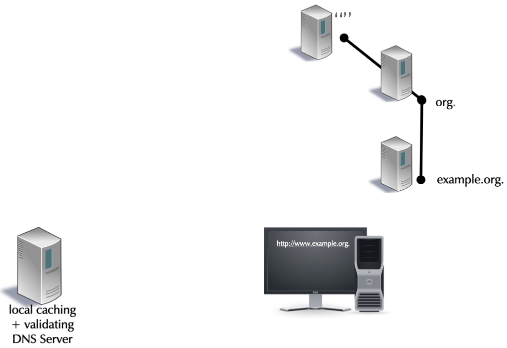
# DNSSEC Name Resolution



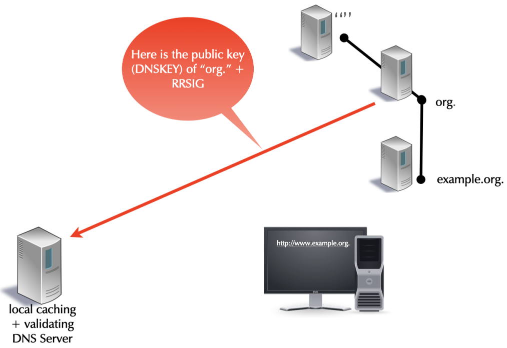
""
What is the public key (DNSKEY) of "org."
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

| 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 ↑ |

# DNSSEC Name Resolution

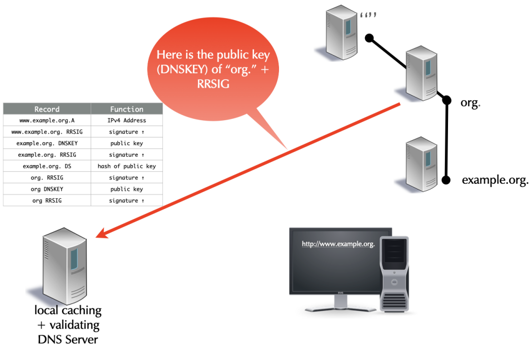


# DNSSEC Name Resolution



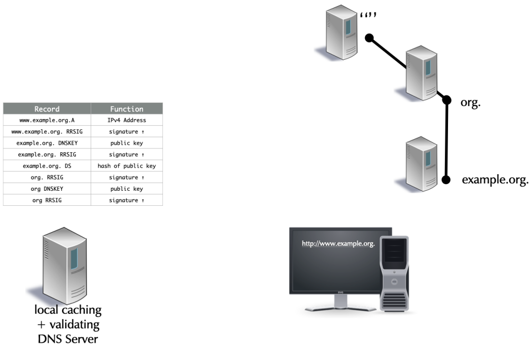
""
Here is the public key (DNSKEY) of "org." + RRSIG
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

## DNSSEC Name Resolution


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

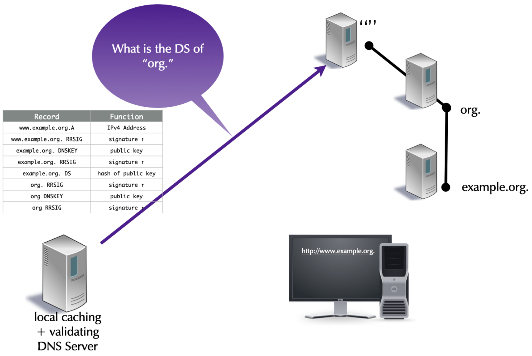
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

## DNSSEC Name Resolution



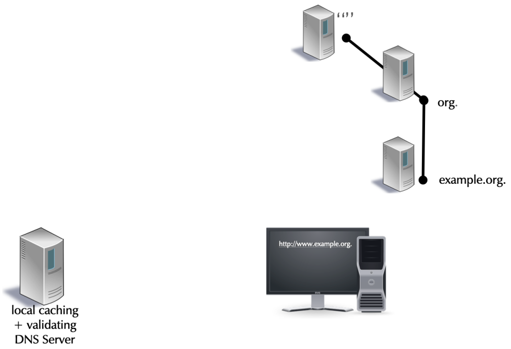
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

## DNSSEC Name Resolution

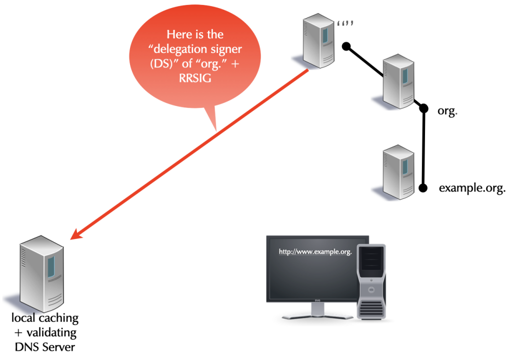


| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

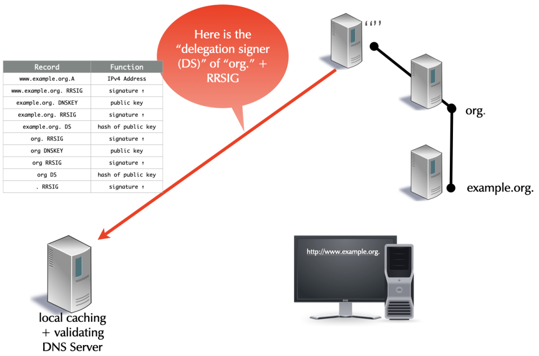
# DNSSEC Name Resolution



# DNSSEC Name Resolution

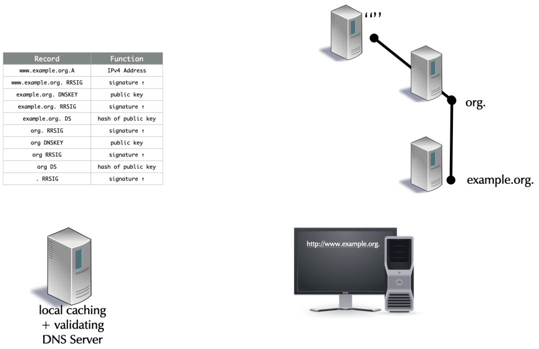


# DNSSEC Name Resolution



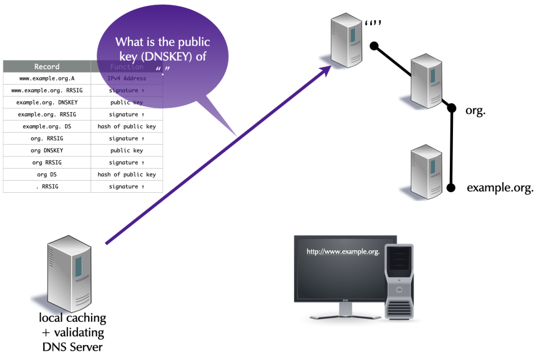
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

# DNSSEC Name Resolution



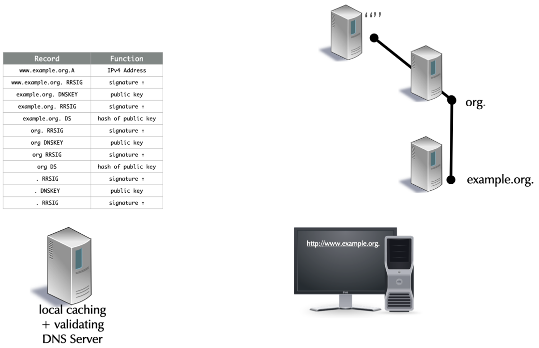
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

# DNSSEC Name Resolution



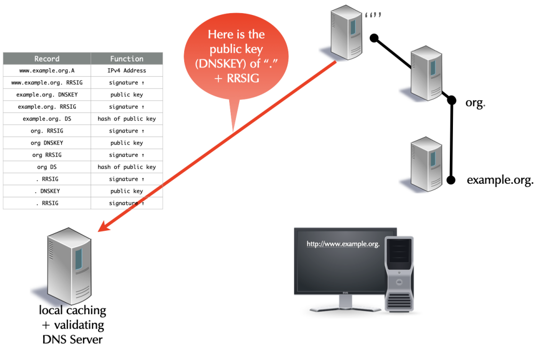
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

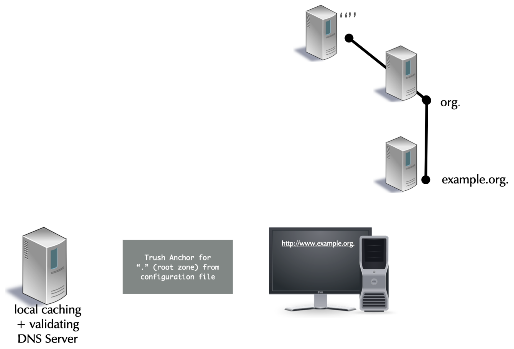
# DNSSEC Name Resolution



| 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 ↑ |

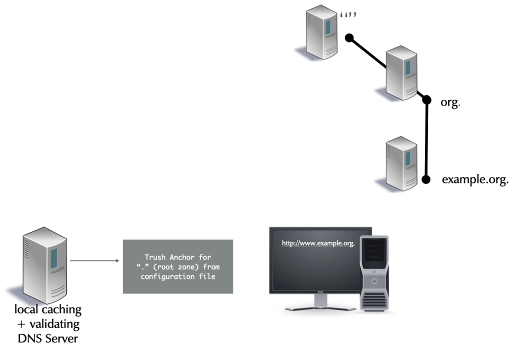
Here is the public key (DNSKEY) of "." + RRSIG
"."
org.
example.org.
http://www.example.org.
local caching + validating DNS Server

# DNSSEC Name Resolution



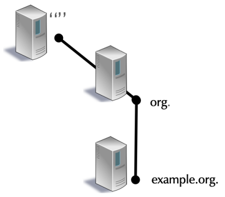
"."
org.
example.org.
Trush Anchor for "." (root zone) from configuration file
http://www.example.org.
local caching + validating DNS Server

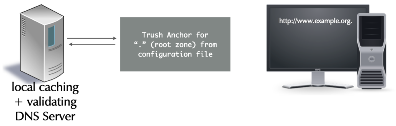
# DNSSEC Name Resolution



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

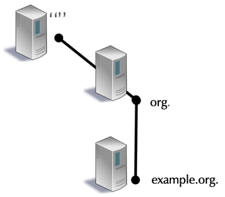
# DNSSEC Name Resolution

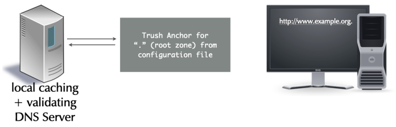




# DNSSEC Name Resolution

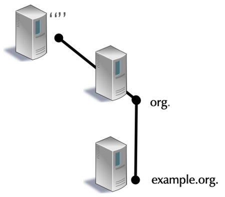
| Record | Function |
|---|---|
| www.example.org.A | IPv4 Address |
| www.example.org. RRSIG | signature ↑ |
| example.org. DNSKEY | public key |
| example.org. RRSIG | signature ↑ |
| example.org. DS | hash of public key |
| org. RRSIG | signature ↑ |
| org DNSKEY | public key |
| org RRSIG | signature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

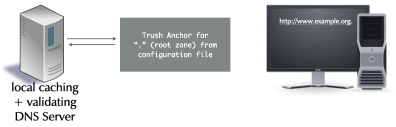




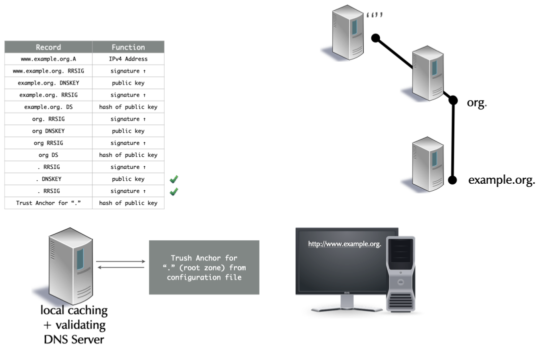
# DNSSEC Name Resolution

| 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 |





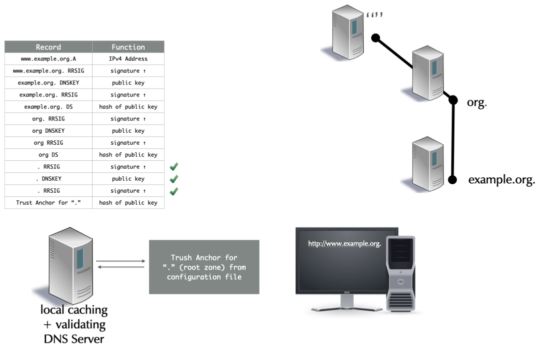
# DNSSEC Name Resolution



| 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

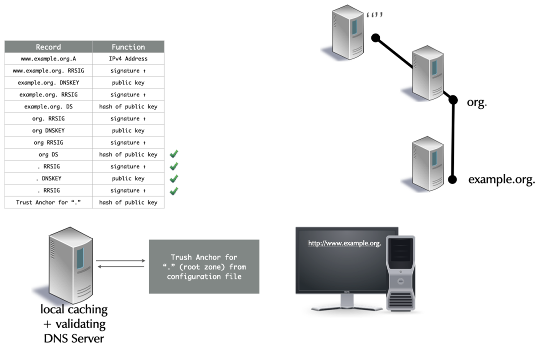
# DNSSEC Name Resolution



| 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

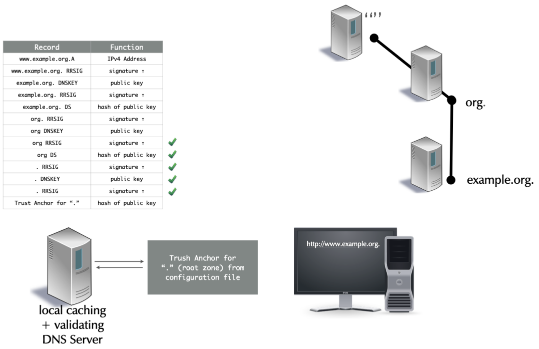
# DNSSEC Name Resolution



| 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

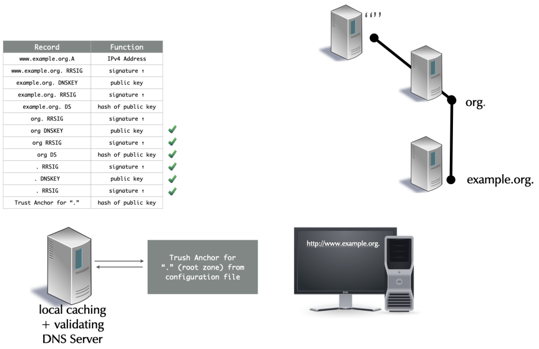
# DNSSEC Name Resolution



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

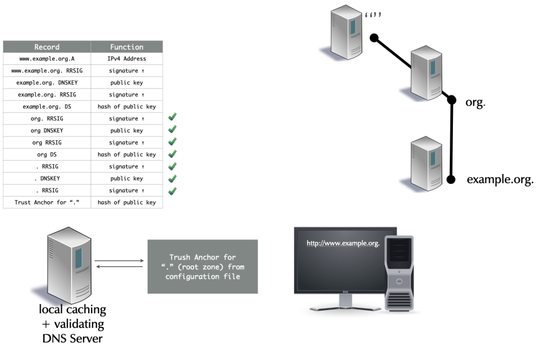
# DNSSEC Name Resolution



| 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

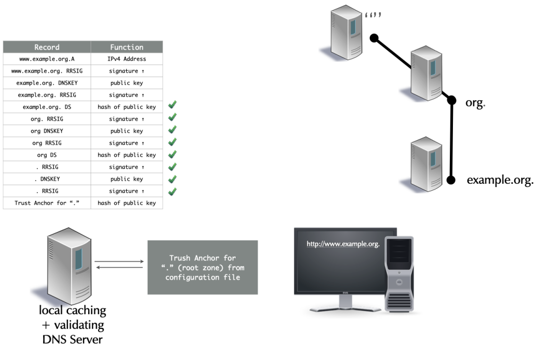
# DNSSEC Name Resolution



| 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

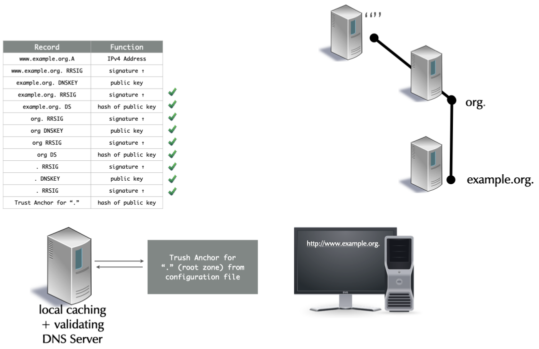
# DNSSEC Name Resolution



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

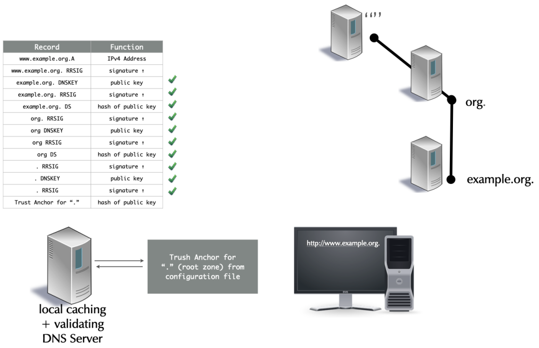
# DNSSEC Name Resolution



| 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

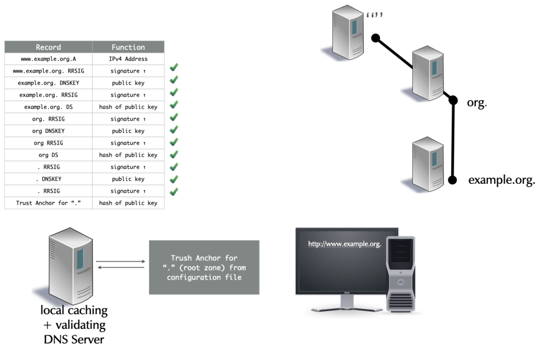
# DNSSEC Name Resolution



| 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

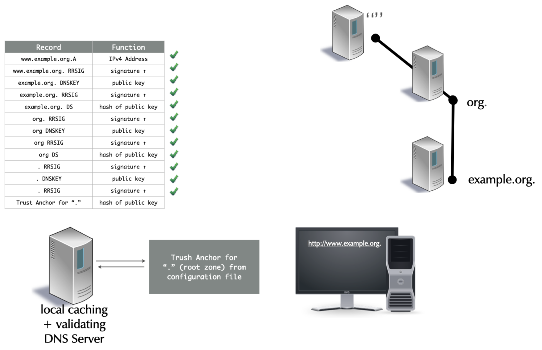
# DNSSEC Name Resolution



| 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 | |

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

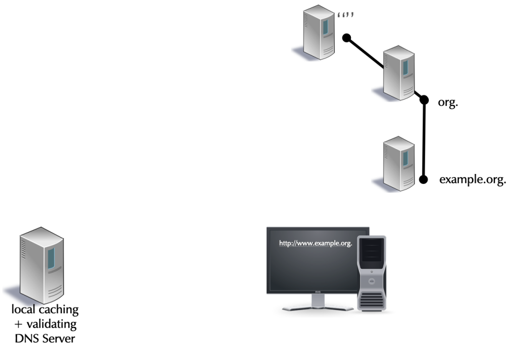
# DNSSEC Name Resolution



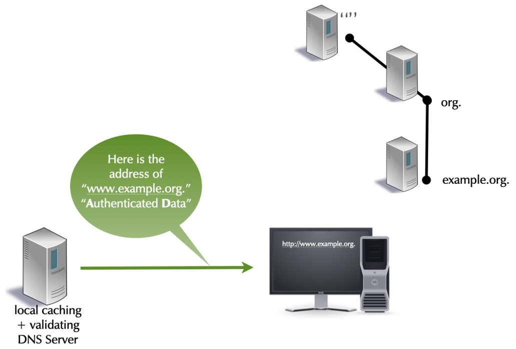
| Record | Function | |
|---|---|---|
| www.example.org.A | IPv4 Address | ✓ |
| www.example.org. RRSIG | signature ↑ | ✓ |
| example.org. DNSKEY | public key | ✓ |
| example.org. RRSIG | signature ↑ | ✓ |
| example.org. DS | hash of public key | ✓ |
| org. RRSIG | signature ↑ | ✓ |
| org DNSKEY | public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



DNSSEC Name Resolution


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

## 7.3. BUILDING A VALIDATING DNS RESOLVER

- Work on the `dnsrNN` virtual machine (DNS resolver server)
- Obtain *root* privileges

```
$ sudo -s
```

- Install BIND 9

```
% dnf install bind
```

- Enable and start BIND 9

```
% systemctl enable --now named
```

- Check that BIND 9 is running without errors

```
% rndc status
% systemctl status named
```

- Try if our BIND 9 works as a DNS resolver

```
$ dig @localhost switch.ch
```

## 7.4. OPTIMIZATIONS TO THE DEFAULT CONFIGURATION

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

  **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

### 7.5. EXTENDED DNS ERRORS (EDE)

- RFC 8914 - Extended DNS Errors [https://datatracker.ietf.org/doc/html/rfc8914] defines a way to deliver additional error information in an DNS response. It is implemented in `dig` since version BIND 9.16.4. BIND 9.18 and other open source DNS-Server support EDE-Messages, as well as some DNS resolver on the Internet (like Cloudflare):

```
$ dig lame.defaultroutes.org soa @1.1.1.1                         <

; <<>> DiG 9.18.41 <<>> lame.defaultroutes.org soa @1.1.1.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 29410
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; EDE: 22 (No Reachable Authority): (at delegation lame.defaultroutes.org.)
;; QUESTION SECTION:
;lame.defaultroutes.org.        IN  SOA

;; Query time: 705 msec
;; SERVER: 1.1.1.1#53(1.1.1.1) (UDP)
;; WHEN: Mon Nov 03 13:19:33 CET 2025
;; MSG SIZE  rcvd: 94
```

- Blog-Post: Extended DNS Errors used in DNS software and services [https://www.sidn.nl/en/news-and-blogs/extended-dns-errors-used-in-dns-software-and-services]

# CHAPTER 8. DNSSEC KEYS

## 8.1. 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/] |
| SM2SM3 | 17 | SM2 and SM3 are cryptographic algorithms that are national standards for China, as well as ISO/IEC standards RFC 9563 [https://www.rfc-editor.org/rfc/rfc9563.html] |
| GOST-2012 | 23 | GOST R 34.10-2012 and GOST R 34.11-2012 are Russian national standards. Their cryptographic properties haven't been independently verified. RFC 9558 [https://www.rfc-editor.org/rfc/rfc9558.html] |

## 8.2. 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

### 8.2.1. 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

## 8.3. 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)

### 8.3.1. 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

### 8.3.2. 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

## 8.4. 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;
};
```

### 8.5. 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)

### 8.6. 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 `secure-mail.example.com`) which is secured with *NSEC3-NARROW* scheme and keys that are online on every authoritative server

# 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. EXERCISE: AN AUTHORITATIVE DNS SERVER

- Use your virtual lab machine, `nsNNa.dnslab.org` (primary authoritative)
- Obtain *root* privileges

```
$ sudo -s
```

- Install BIND 9 and start the BIND 9 process

```
% dnf install bind
% systemctl enable --now named
```

- Open port 53 (DNS) in the firewall

```
% firewall-cmd --permanent --zone=public --add-service=dns
% firewall-cmd --reload
```

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

```
% named -V
```

- Open the BIND 9 configuration file `/etc/named.conf` with a text editor

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

- Disable recursion

```
options {
  [...]
```

```
    recursion no;
    [...]
  };
```

- Adjust the `listen-on` directives to listen to `any` interface

```
  options {
    [...]
    listen-on { any; };
    [...]
  };
```

- Permit queries from anywhere

```
  options {
    [...]
    allow-query { any; };
    [...]
  };
```

- Add a sensible logging configuration for an authoritative BIND 9 server (here's a template you can use [https://dnslab.org/dns/logging.conf]).
- Add a new primary zone definition at the end of the configuration file for the zone `zoneNN.dnslab.org`. Replace `NN` with your participant number

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

- Create a new zone file for the zone `zoneNN.dnslab.org` in the directory `/var/named`. It is important to use low Time-To-Live values in our lab zones (recommended: 60 seconds), else the key rollover of the DNSSEC keys later in the training will take too long time.

```
  $ORIGIN zoneNN.dnslab.org.
  $TTL 60
  @                  IN SOA nsNNa.dnslab.org. hostmaster 1001 1h 30m 41d 60
                     IN NS  nsNNa.dnslab.org.
                     IN TXT "Zone zoneNN.dnslab.org"
                     IN MX 10 mail.zoneNN.dnslab.org.
+
  mail               IN A xxx.xxx.xxx.xxx
                     IN AAAA xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx
```

- Replace `NN` with your participant number. Add the IPv4 and IPv6 address of your VM (see `hostname -I` for the addresses)

```
  % $EDITOR /var/named/zoneNN.dnslab.org
```

- Check the BIND 9 configuration file for errors

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

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

```
  % rndc reconfig
```

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

```
$ dig @127.0.0.1 zoneNN.dnslab.org SOA
```

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

```
$ dig @9.9.9.9 zoneNN.dnslab.org SOA
```

### 9.3. EXERCISE: SIGNING BIND 9.6 STYLE (MANUAL SIGNING)

- Create a Zone Signing Key (ZSK), RSASHA256 Algorithm and 1024 bit length

```
% mkdir -p /etc/bind/keys
% cd /etc/bind/keys
% dnssec-keygen -a RSASHA256 -b 1024 zoneNN.dnslab.org
```

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

```
% dnssec-keygen -a RSASHA256 -b 2048 -f KSK zoneNN.dnslab.org
```

- 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 KzoneNN.dnslab.org.+008+*.key >> /var/named/zoneNN.dnslab.org
% $EDITOR /var/named/zoneNN.dnslab.org
```

- DNSSEC sign the zone

```
% dnssec-signzone -o zoneNN.dnslab.org \
       -k KzoneNN.dnslab.org.+008+(KSK).private \
      /var/named/zoneNN.dnslab.org \
      KzoneNN.dnslab.org.+008+(ZSK).private
```

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

```
zone "zoneNN.dnslab.org" {
      type primary;
      file "zoneNN.dnslab.org.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 zoneNN.dnslab.org SOA +dnssec +multi
```

- The DS record for the zone can be found in the file `dsset-zoneNN.dnslab.org` in the directory where the zone has been signed. Copy the DS record onto the machine of the trainer (using `scp`, username `user` and password `DNSandBIND`. 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-zoneNN.dnslab.org.
% scp dsset-zoneNN.dnslab.org. user@ns01a.dnslab.org:.
```

- 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> zoneNN.dnslab.org SOA +dnssec +multi
```

- Query the DS record from the parent

```
$ dig @<IP-of-resolver> zoneNN.dnslab.org DS +dnssec
```

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

```
% named-checkzone zoneNN.dnslab.org /var/named/zoneNN.dnslab.org
```

- 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 zoneNN.dnslab.org
```

- 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.zoneNN.dnslab.org +dnssec +multi
```

# CHAPTER 10. DNSSEC KEY AND SIGNING POLICY

- BIND 9.16 has introduced a new `dnssec-policy` feature as a further step in automation
- Configuring a zone to use KASP (*Key And Signing Policy*) can be as easy as

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

- Requirements

One of the following

```
inline-signing yes;    // manual edited zone files
update-policy local;   // dynamic update zone files
```

- 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

### 10.1. DEFAULT POLICY

- Single CSK ("common signing key", also called a *SSK* - "single signing key")

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`

## 10.2. 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;
};
```

## 10.3. 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>;
};
```

## 10.4. A REAL WORLD CUSTOM DNSSEC POLICY

- A KSK with ECDSAP256SHA256 and no automatic key rollover
- A ZSK with ECDSAP256SHA256 and a rollover interval of 30 days
- Signatures are valid for 10 days
- Signatures will be refreshed after 8 days (2 day buffer)
- TTL of DNSKEY records is 2 hours
- Max TTL of other records in the zone is 1 day
- DNSSEC keys will be published one hour after they have become valid (publish-savety)
- DNSSEC keys will be kept in the zone one our after deactivation (retire-safety)
- Expired DNSSEC keys will be removed after 90 days
- Zone transfer between primary and secondary servers is 5 minutes (zone-propagation-delay) *

```
dnssec-policy "example.eu" {
       dnskey-ttl 2h;
       keys { ksk lifetime unlimited algorithm ECDSAP256SHA256;
              zsk lifetime P30D      algorithm ECDSAP256SHA256;
            };
       max-zone-ttl 1d;
       publish-safety 1h;
```

```
      purge-keys P90D;
      retire-safety 1h;
      signatures-refresh 8d;
      signatures-validity 10d;
      signatures-validity-dnskey 10d;
      zone-propagation-delay 300;
};
```

# CHAPTER 11. EASY DNSSEC WITH BIND "DEFAULT-POLICY"

- Using DNSSEC policy configuration (available since BIND 9.16) 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).

## 11.1. DNSSEC SIGNING THE ZONE

- We work on the primary authoritative server
- Create a new zone file with the name `p.zoneNN.dnslab.org` in `/var/named/`. The zone should contain `SOA`, `NS` and one `TXT` record. The `NS` record should point to `nsNNa.dnslab.org`

```
$ORIGIN p.zoneNN.dnslab.org.
$TTL 60
@        IN SOA nsNNa.dnslab.org. hostmaster.zoneNN.dnslab.org. 1001 1h 2h 41d 60
         IN NS  nsNNa.dnslab.org.
     IN TXT "This is a zone with BIND 9 default policy"
```

- Open the file `/etc/named.conf` in an editor and add a zone block

  **with `dnssec-policy default;` line to the configuration**

```
zone "p.zoneNN.dnslab.org" {
  type primary;
  file "p.zoneNN.dnslab.org";
  dnssec-policy default;
  inline-signing yes;
};
```

- Check the configuration and reload BIND 9

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

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

```
% ls -l /var/named
```

- Test if the DNS server returns DNSSEC resource records

```
$ dig @localhost p.zoneNN.dnslab.org +dnssec +multi
$ dig @localhost p.zoneNN.dnslab.org DNSKEY +dnssec +multi
```

- Resolve a domain name `p.zoneNN.dnslab.org` from your zone on your resolver machine. Does it show the `AD`-Flag? What might be missing?

## 11.2. ADD THE DS-RECORD FROM YOUR CHILD-ZONE TO YOUR PARENT ZONE

- Your zone `zoneNN.dnslab.org.` in the parent of `p.zoneNN.dnslab.org`
- In order to close the *chain of trust*, you must publish the DS-record for `p.zoneNN.dnslab.org` in your own zone `zone.dnslab.org`. The DS-record contains the hash of the main key of the zone (the KSK or

CSK/SSK). The CSK can be found in `/var/named/Kp.zoneNN.dnslab.org.013<CSK-ID>.key`, and the DS-Record can be created with (replace ZZZZZ with the key-id):

```
% dnssec-dsfromkey /var/named/Kp.zoneNN.dnslab.org.+013+ZZZZZ.key
```

- Add a delegation NS-Record into the parent zone `zoneNN.dnslab.org`

```
p.zoneNN.dnslab.org.     IN NS nsNNa.dnslab.org.
```

- Add the DS-Record to the zone `zoneNN.dnslab.org`, increment the SOA serial number, resign the `zoneNN.dnslab.org` (the parent zone that does not use automation) with `dnssec-signzone` (same command used in the previous exercises).
- Reload the parent zone

```
% rndc reload zoneNN.dnslab.org
```

- Try again to query a name from the zone `p.zoneNN.dnslab.org`, the `AD` flag should now appear.

# CHAPTER 12. NSEC VS. NSEC3

## 12.1. 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

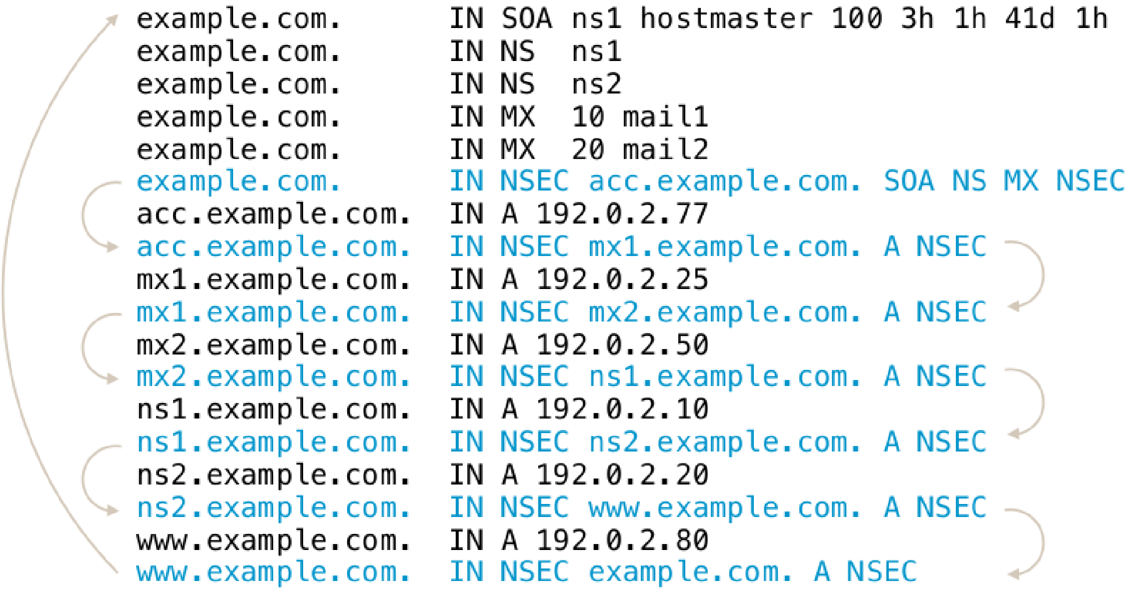
- DNS knows two kinds of negative answers: NXDOMAIN and NODATA/NXRRSET

  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
```

## 12.2. 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)

The new *compact authenticated denial of existance* standard (RFC 9824) solves this issue.

- Example of zone walking:

```
$ ldns-walk paypal.com
```

## 12.3. THE NSEC3 RECORD

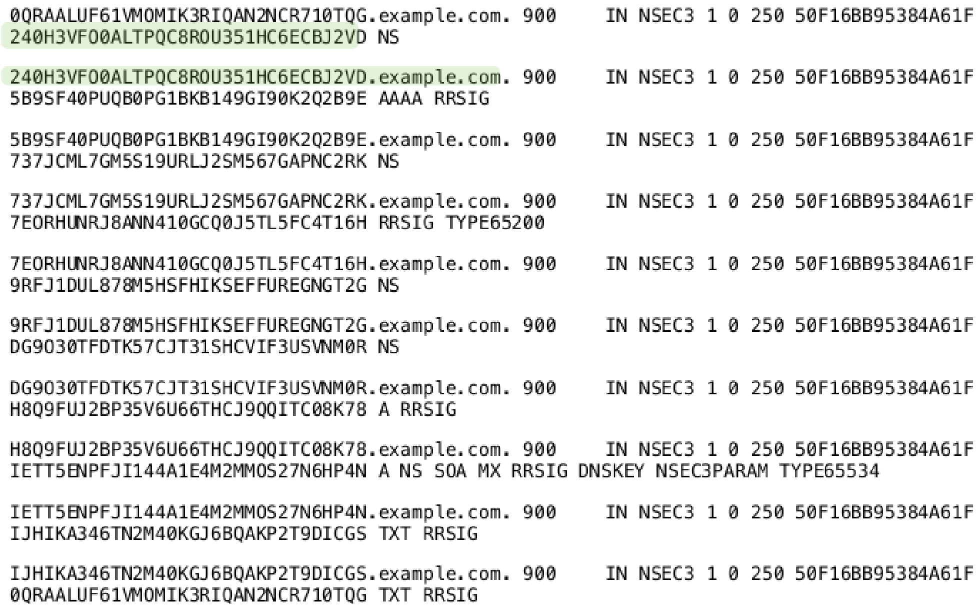
- RFC 5155 [https://tools.ietf.org/html/rfc5155] (2008) defines the NSEC3 record as an alternative to NSEC

  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

## 12.4. 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
```

### 12.5. 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

### 12.6. 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
```

### 12.7. 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

### 12.8. 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

### 12.9. 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)

### 12.10. 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

### 12.11. 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)

### 12.12. 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 …) or a single dash (-))

```
% dnssec-signzone -3 <salt> -H <iterations> -o zoneNN.dnslab.org \
       -k <KSK-private-file> zoneNN.dnslab.org \
       <ZSK-private-file>
```

# CHAPTER 13. DNSSEC APPLICATIONS

- Securing E-Mail with SPF/DKIM/DMARC: https://switch.dane.onl [https://switch.dane.onl]
- Securing Security with DANE (x509 trust from DNS): https://switch-lausanne.dane.onl [https://switch-lausanne.dane.onl]

# CHAPTER 14. DNSSEC KEY-ROLLOVER

## 14.1. WHY KEYROLLOVER

- 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)

## 14.2. 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

## 14.3. DNSSEC KEYROLLOVER DOCUMENTATION

- **RFC 6781** - DNSSEC Operational Practices, Version 2 [https://www.rfc-editor.org/rfc/rfc6781]
- **RFC 7583** - DNSSEC Key Rollover Timing Considerations [https://www.rfc-editor.org/rfc/rfc7583]

## 14.4. 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

### 14.5. 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

#### 14.5.1. 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

#### 14.5.2. 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)

#### 14.5.3. 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)

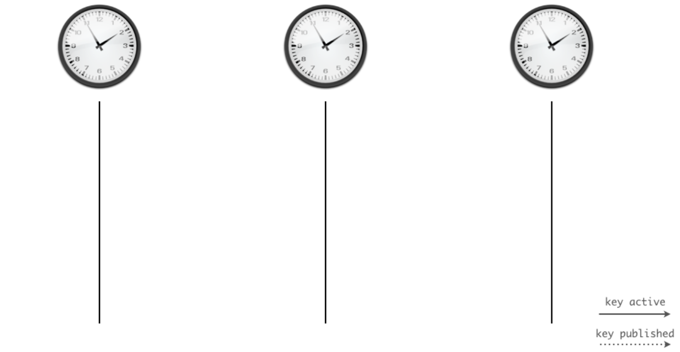
#### 14.5.4. ZSK - pre-publication - Step 4

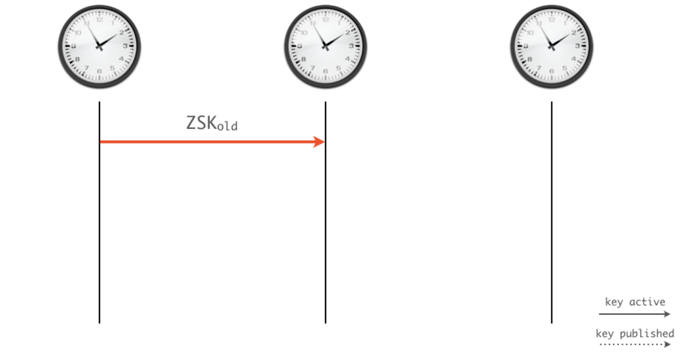
- Wait for the new zone version with the signatures from the new ZSK to be visible on all authoritative DNS servers of the zone (zone transfer)
- Wait for the largest TTL in the zone (plus some buffer)
- Now the signatures created by the old ZSK are expired from the caches, and the new signatures are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

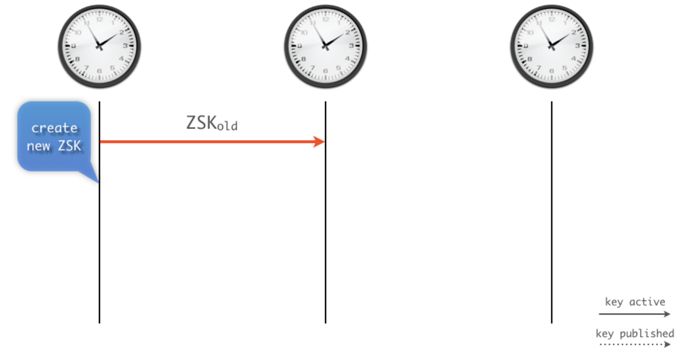
#### 14.5.5. ZSK - pre-publication - Step 5

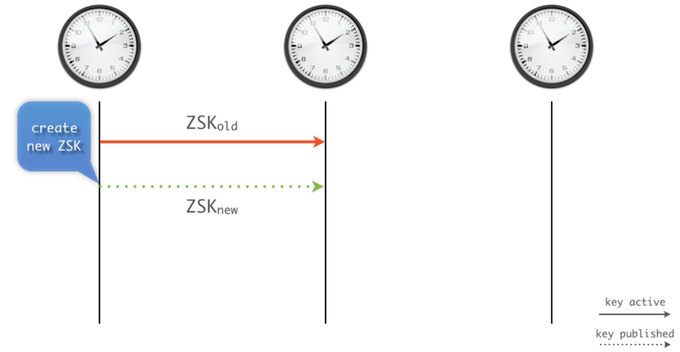
- Remove the old ZSK from the DNSKEY record set of the zone
- Continue signing the zone with the new ZSK

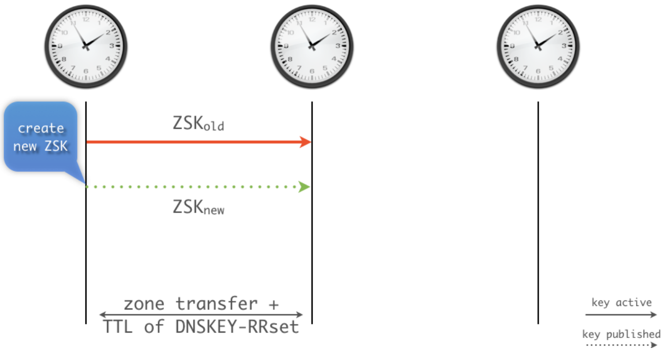
### 14.5.6. ZSK - pre-publication in pictures

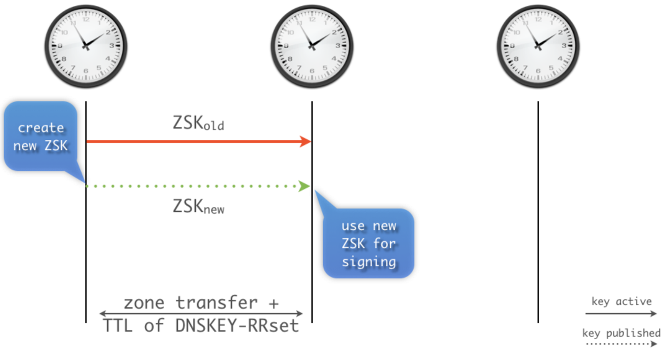


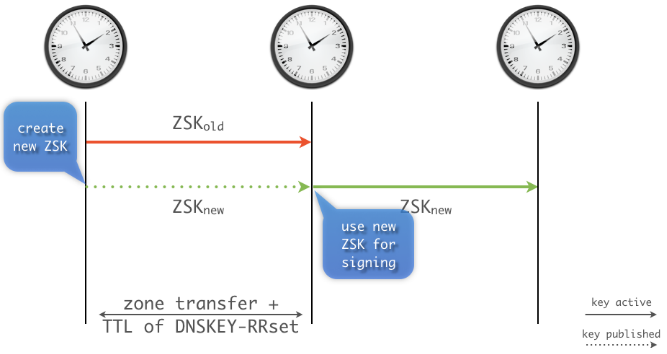


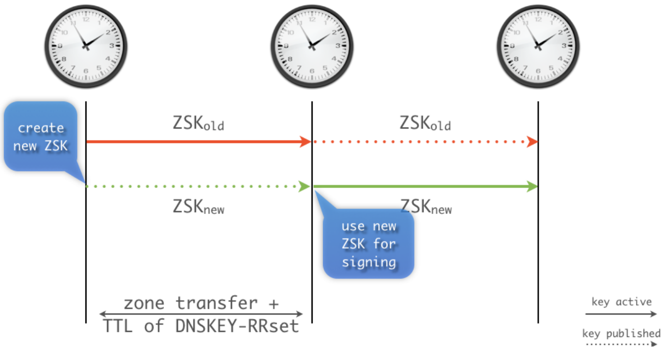


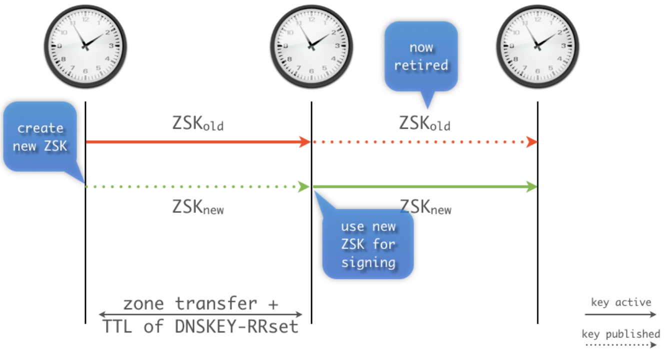


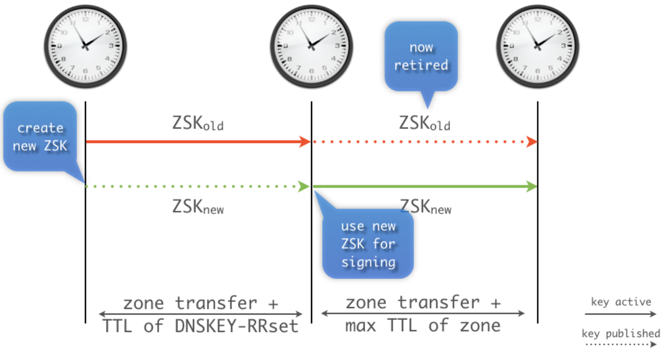

create new ZSK
ZSKold
ZSKnew
zone transfer + TTL of DNSKEY-RRset
key active
key published

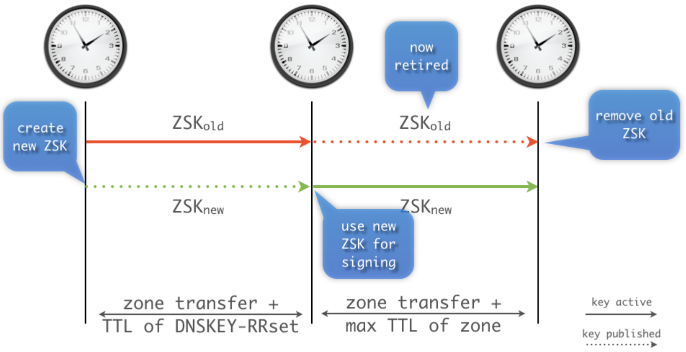

create new ZSK
ZSKold
ZSKnew
use new ZSK for signing
zone transfer + TTL of DNSKEY-RRset
key active
key published

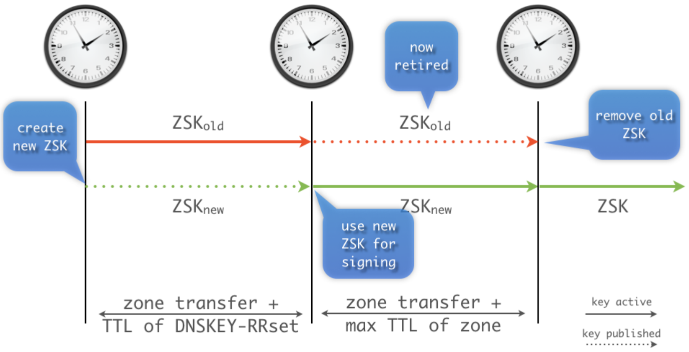

create new ZSK
ZSKold
ZSKnew
use new ZSK for signing
ZSKnew
zone transfer + TTL of DNSKEY-RRset
key active
key published


create new ZSK
ZSKold
ZSKold
ZSKnew
use new ZSK for signing
ZSKnew
zone transfer + TTL of DNSKEY-RRset
key active
key published


create new ZSK
$\text{ZSK}_{\text{old}}$
now retired
$\text{ZSK}_{\text{old}}$
$\text{ZSK}_{\text{new}}$
$\text{ZSK}_{\text{new}}$
use new ZSK for signing
zone transfer + TTL of DNSKEY-RRset
key active
key published


create new ZSK
$\text{ZSK}_{\text{old}}$
now retired
$\text{ZSK}_{\text{old}}$
$\text{ZSK}_{\text{new}}$
$\text{ZSK}_{\text{new}}$
use new ZSK for signing
zone transfer + TTL of DNSKEY-RRset
zone transfer + max TTL of zone
key active
key published


create new ZSK
$\text{ZSK}_{\text{old}}$
now retired
$\text{ZSK}_{\text{old}}$
remove old ZSK
$\text{ZSK}_{\text{new}}$
$\text{ZSK}_{\text{new}}$
use new ZSK for signing
zone transfer + TTL of DNSKEY-RRset
zone transfer + max TTL of zone
key active
key published


create new ZSK
$\text{ZSK}_{\text{old}}$
now retired
$\text{ZSK}_{\text{old}}$
remove old ZSK
$\text{ZSK}_{\text{new}}$
$\text{ZSK}_{\text{new}}$
ZSK
use new ZSK for signing
zone transfer + TTL of DNSKEY-RRset
zone transfer + max TTL of zone
key active
key published

### 14.6. EXERCISE: ZSK ROLLOVER

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

```
 % cd /etc/bind/keys
 % dnssec-keygen -a RSASHA256 -b 1024 zoneNN.dnslab.org
```

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

```
 % cat KzoneNN.dnslab.org.+008+<ID-of-the-new-ZSK>.key >> /var/named/zoneNN.dnslab.org
```

- 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 zoneNN.dnslab.org \
   -k <KSK-private-file> \
   /var/named/zoneNN.dnslab.org  \
   <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 zoneNN.dnslab.org +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 zoneNN.dnslab.org \
       -k <KSK-private-file> \
       /var/named/zoneNN.dnslab.org  \
   <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 zoneNN.dnslab.org +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 `/var/named/zoneNN.dnslab.org`, increment the SOA serial and sign the zone with the new ZSK:

```
   % dnssec-signzone -N unixtime -o zoneNN.dnslab.org \
         -k <KSK-private-file> \
         /var/named/zoneNN.dnslab.org  \
     <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 zoneNN.dnslab.org +nssearch
```

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

```
$ dig zoneNN.dnslab.org SOA +dnssec +multi
```

- This is the end of the ZSK rollover

## 14.7. 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)

### 14.7.1. 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)

### 14.7.2. 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)

### 14.7.3. KSK - double-signing - Step 3

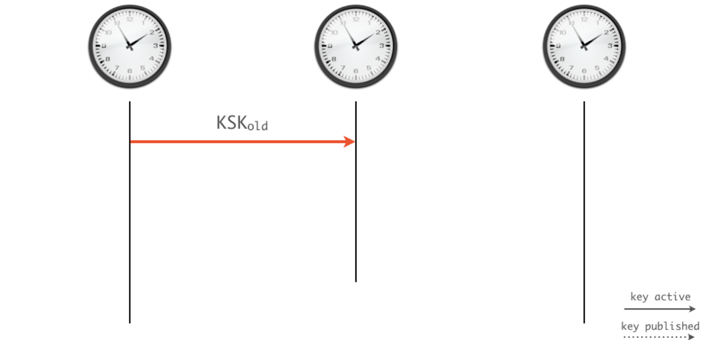
- Send the new DS record to the operator of the parent DNS zone (usually through an API or through an Web-Interface)
- Wait for the DS record to be updated in the parent zone
- Wait for the TTL of the DS record in the parent zone (+ some buffer)
- The new DS record is now visible for all DNS clients

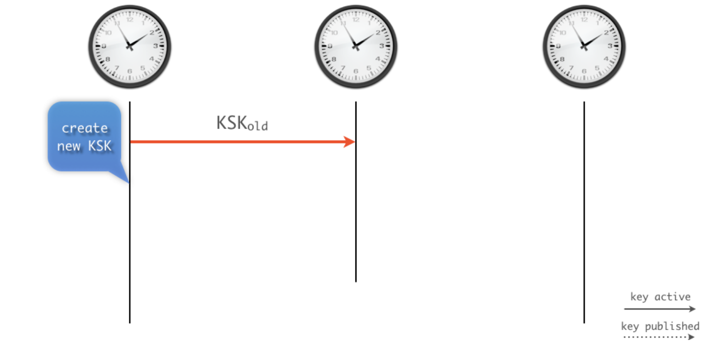
### 14.7.4. KSK - double-signing - Step 4

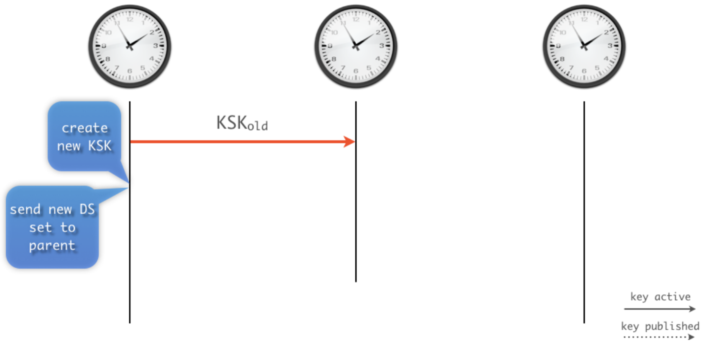
- Remove the old KSK from the DNSKEY record set of the zone
- Sign the zone with only the new KSK
- Wait and remove the old DS from the parent zone

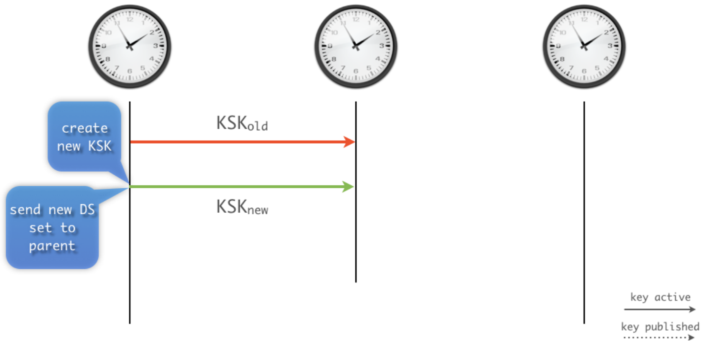
### 14.7.5. KSK - double-signing in pictures

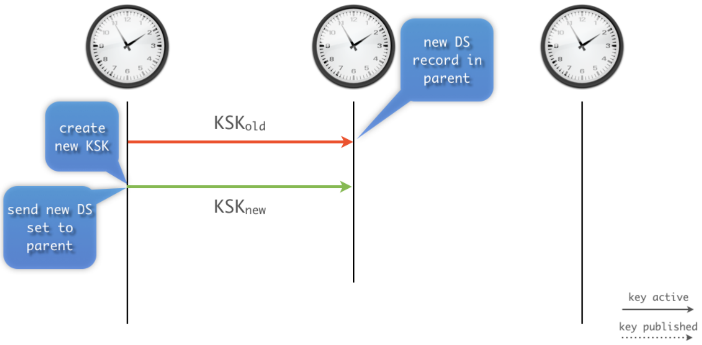
```
   [sidebar]
image::./img/ksk-roll-00.png[scaledwidth=65%]
```

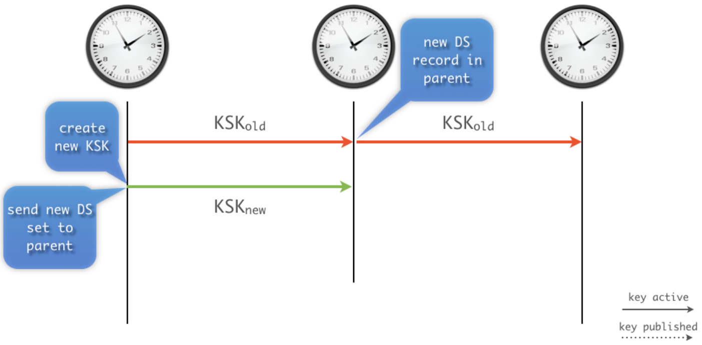


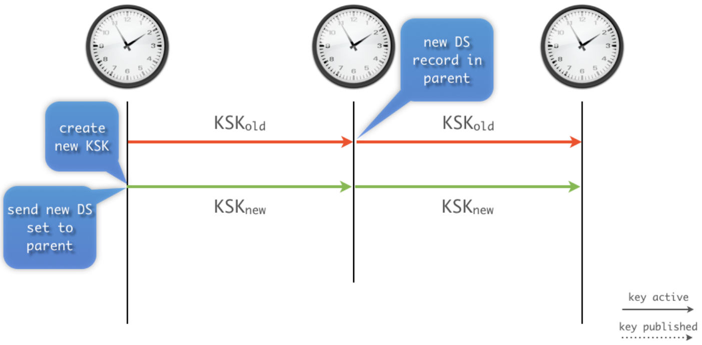


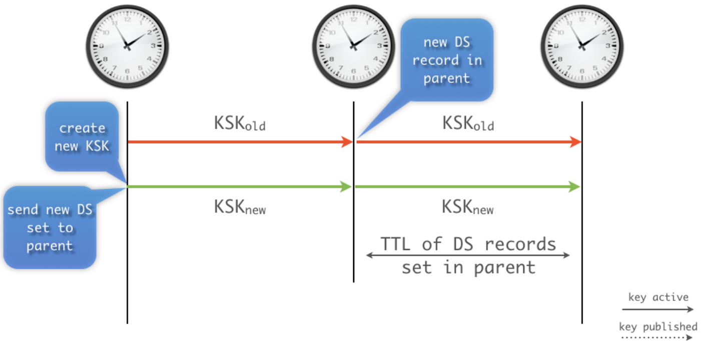


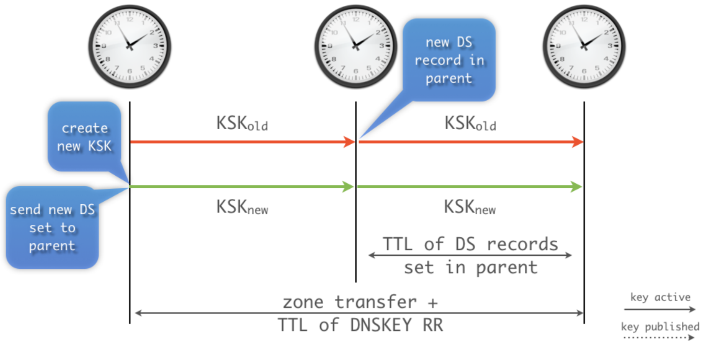


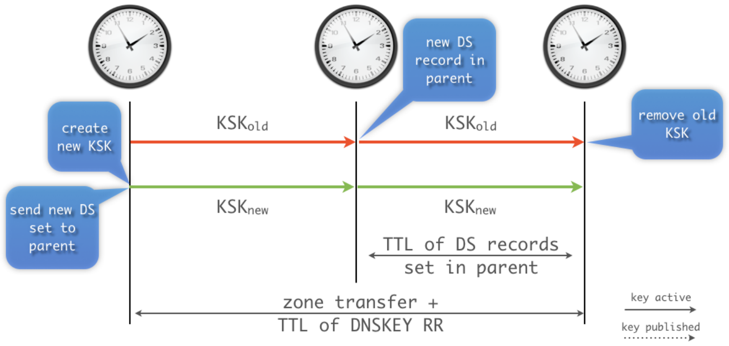

new DS record in parent
create new KSK
$KSK_{old}$
send new DS set to parent
$KSK_{new}$
key active
key published

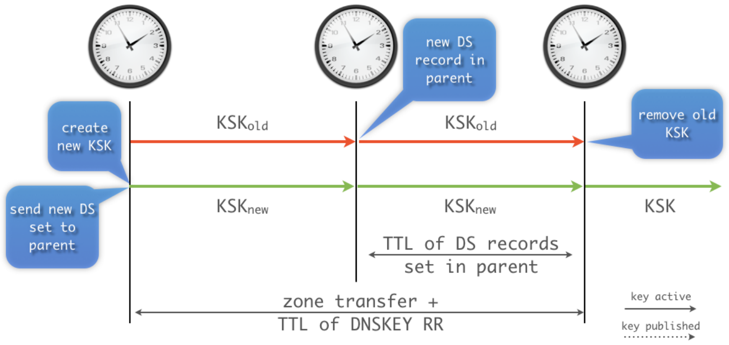

new DS record in parent
create new KSK
$KSK_{old}$
$KSK_{old}$
send new DS set to parent
$KSK_{new}$
key active
key published


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


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







## 14.8. EXERCISE: KSK ROLLOVER

- Use your virtual lab machine, `nsNNa.dnslab.org` (primary authoritative)
- Create a new KSK key pair

```
% cd /etc/bind/keys
% dnssec-keygen -a RSASHA256 -b 2048 -f KSK zoneNN.dnslab.org
```

- 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 KzoneNN.dnslab.org.+008+<ID-of-new-KSK>.key >> /var/named/zoneNN.dnslab.org
% dnssec-signzone -N unixtime -o zoneNN.dnslab.org \
        -k <OLD-KSK-private-file> \
        -k <NEW-KSK-private-file> \
        /var/named/zoneNN.dnslab.org \
        <Current-ZSK-private-file>
```

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

```
% rndc reload
$ dig zoneNN.dnslab.org SOA +dnssec +multi
```

- Submit the new DS to the parent zone

```
% scp dsset-zoneNN.dnslab.org. user@ns01a.dnslab.org:.
```

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

```
$ dig zoneNN.dnslab.org 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 zoneNN.dnslab.org \
            -k <NEW-KSK-private-file> \
            /var/named/zoneNN.dnslab.org \
            <Current-ZSK-private-file>
```

- Load the signed zone into the BIND 9 server

```
% rndc reload
```

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

```
$ dig zoneNN.dnslab.org SOA +dnssec +multi
```

### 14.9. ALGORITHM ROLLOVER

- An algorithm rollover is used when the DNSSEC key algorithm of the zone needs to be changed

    e.g. when switching from RSASHA256 to ECDSASHA256

- During an algorithm rollover, the KSK and ZSK will be changed at the same time using a *double-signing* rollover scheme

    During such a roll over, the zone should be monitored for failures and over sized DNS answer messages

    Link: DNSSEC algorithm rollover HOWTO [https://www.dns.cam.ac.uk/news/2020-01-15-rollover.html] by Tony Finch (Univ. of Cambridge)

### 14.10. EMERGENCY ROLLOVER KEY

- In an emergency, time is at an essence
- To save time on an KSK rollover, step one (publication) can be done ahead of time

    This published extra key is called the *standby* key

    It saves time, but makes the DNSKEY RRSet larger

- The *standby* key is not used during *normal* DNSSEC signing operation
- If there is a need to change the KSK in an emergency, the zone can almost immediately switch over to the *standby* key
- The *standby* key should be replaced (rolled) whenever the production KSK is rolled

## CHAPTER 15. KSK-2024 ROOT ZONE KSK-ROLLOVER

- The Key-Signing-Key of the Internet Root Zone will be rolled in October 2026 (for the 2nd time in since the root has been DNSSEC signed).

  Because this key is the trust-anchor for all of DNSSEC, and the public part or hash of that key has to be present in *all* DNS-resolver **before** the roll, special care must be taken by DNS resolver operators.

  The new KSK2024 key (Key-ID 38696) has been created in April 2024 and has been published in January 2025 inside the root DNS zone:

```
$ dig dnskey . +multi
+
; <<>> DiG 9.18.41 <<>> dnskey . +multi
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33764
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1
+
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;.          IN DNSKEY
+
;; ANSWER SECTION:
.           77796 IN DNSKEY 257 3 8 (
                AwEAAaz/tAm8yTn4Mfeh5eyI96WSVexTBAvkMgJzkKTO
                iW1vkIbzxeF3+/4RgWOq7HrxRixHlFlExOLAJr5emLvN
                7SWXgnLh4+B5xQlNVz8Og8kvArMtNROxVQuCaSnIDdD5
                LKyWbRd2n9WGe2R8PzgCmr3EgVLrjyBxWezF0jLHwVN8
                efS3rCj/EWgvIWgb9tarpVUDK/b58Da+sqqls3eNbuv7
                pr+eoZG+SrDK6nWeL3c6H5Apxz7LjVc1uTIdsIXxuOLY
                A4/ilBmSVIzuDWfdRUfhHdY6+cn8HFRm+2hM8AnXGXws
                9555KrUB5qihylGa8subX2Nn6UwNR1AkUTV74bU=
                ) ; KSK; alg = RSASHA256 ; key id = 20326
.           77796 IN DNSKEY 257 3 8 (
                AwEAAa96jeuknZlaeSrvyAJj6ZHv28hhOKkx3rLGXVaC
                6rXTsDc449/cidltpkyGwCJNnOAlFNKF2jBosZBU5eeH
                spaQWOmOElZsjICMQMC3aeHbGiShvZsx4wMYSjH8e7Vr
                hbu6irwCzVBApESjbUdpWWmEnhathWu1jo+siFUiRAAx
                m9qyJNg/wOZqqzL/dL/q8PkcRU5oUKEpUge71M3ej2/7
                CPqpdVwuMoTvoB+ZOT4YeGyxMvHmbrxlFzGOHOijtzN+
                u1TQNatX2XBuzZNQ1K+s2CXkPIZo7s6JgZyvaBevYtxP
                vYLw4z9mR7K2vaF18UYH9Z9GNUUeayffKC73PYc=
                ) ; KSK; alg = RSASHA256 ; key id = 38696
.           77796 IN DNSKEY 256 3 8 (
                AwEAAeuS7hMRZ7muj1c/ew2DoavxkBw3jUG5R79pKVDI
                39fxvlD1HfJYGJERnXuV4SrQfUPzWw/lt5Axb0EqXL/s
                Q2ZyntVmwQwoSXi2l9smlIKh2UrkTmmozRCPe7GS4fZT
                E4Ew7AV4YprTLgVmxiGP4unRuXkYOgQJXCIBpVCmEdUl
                i6X2sm0i5ZilU/Q+rX3RDw+eYPP7K0cJnJ+ZnvQDy14+
                BFtreWl9enNQljgGpBp26lEp1z7AbvUfzkQNVCVGaM8j
                EaCSALXiROlwCQMOdj+tpLXFf99kdJnTQw2cpEr1uGs/
                yENAJpoGhbjZAoIkXzDUBSlfFeYz7FWzH6gIe7M=
                ) ; ZSK; alg = RSASHA256 ; key id = 61809
+
;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1) (UDP)
;; WHEN: Mon Nov 03 13:57:31 CET 2025
;; MSG SIZE  rcvd: 853
```

- Blog-Post: The 2024-2026 Root Zone KSK Rollover: Initial Observations and Early Trends [https://blog.verisign.com/security/2024-2026-root-zone-ksk-rollover-initial-observations/]

- ICANN KSK Rollover Files https://www.iana.org/dnssec/files [https://www.iana.org/dnssec/files]
- RFC 9718 - DNSSEC Trust Anchor Publication for the Root Zone [https://www.rfc-editor.org/rfc/rfc9718.html]

| Event | Date | Description |
|---|---|---|
| Publication | 11 January 2025 | The successor key was introduced in the DNS root zone. |
| - | 10 February 2025 | The successor key should begin to be trusted by resolvers that follow the mechanisms described in RFC5011. |
| Rollover | 11 October 2026 | The successor key is scheduled to sign the zone; the current key will not sign the zone. Validating resolvers must have updated trust anchors to continue validating the root zone. |

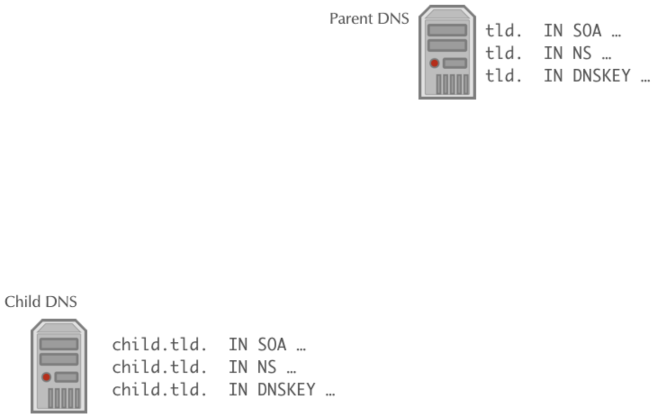
# CHAPTER 16. DNSSEC KEY-ROLLOVER AUTOMATION

- 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)

## 16.1. 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)

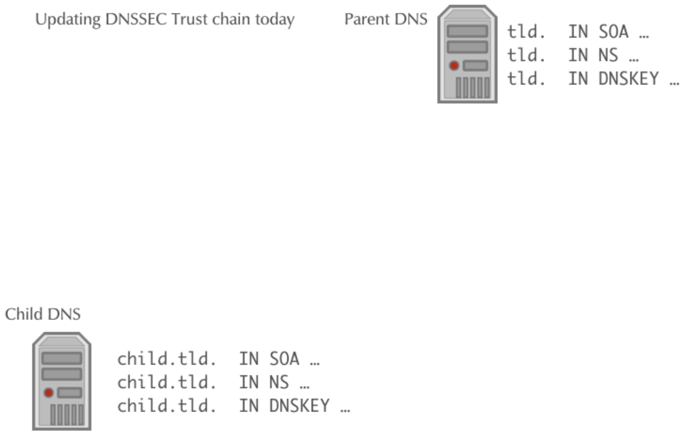
  If a new CDNSKEY record is found in the child zone, the authoritative DNS server of the parent zone will validate the record, calculate the hash to get a DS record, and will then replace the DS record with the newly calculated DS record
- BIND 9 supports CDS and CDNSKEY since version 9.11
- The `dnssec-cds` 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



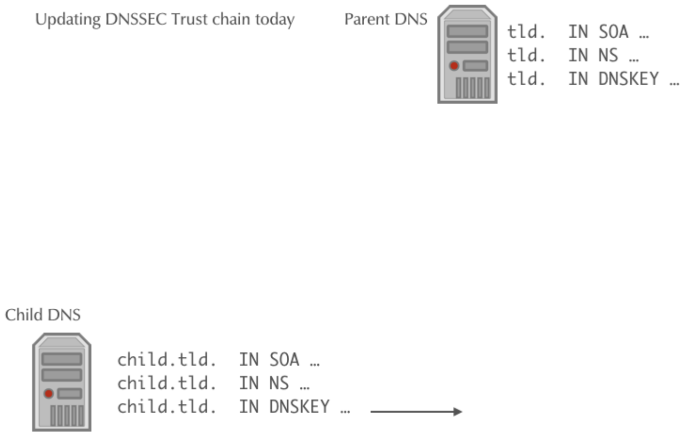
Parent DNS

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

Child DNS

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

Updating DNSSEC Trust chain today



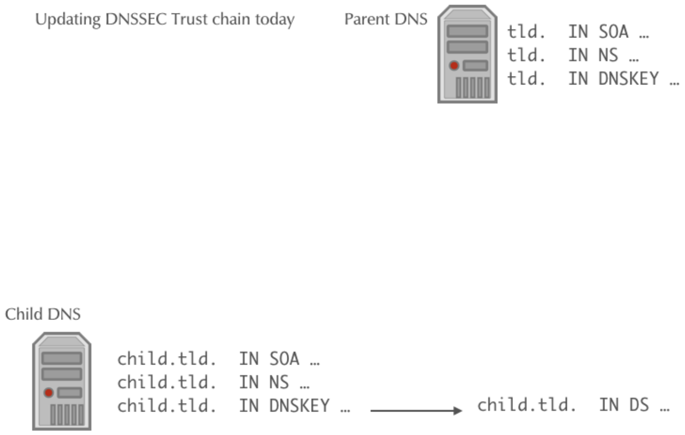
Parent DNS

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

Child DNS

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

Updating DNSSEC Trust chain today

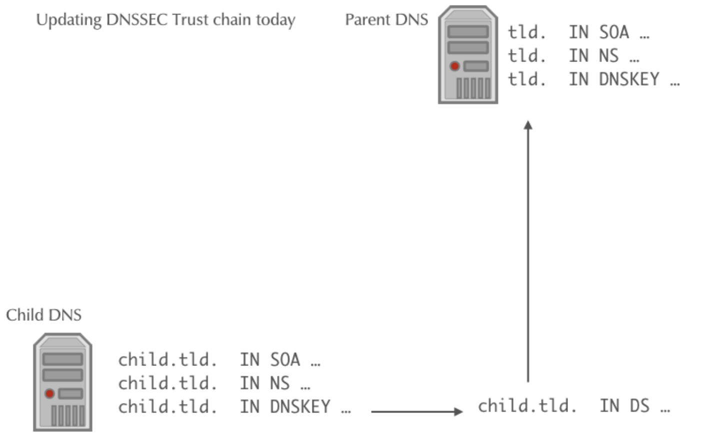


Parent DNS

```
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 DS …
```



Updating DNSSEC Trust chain today

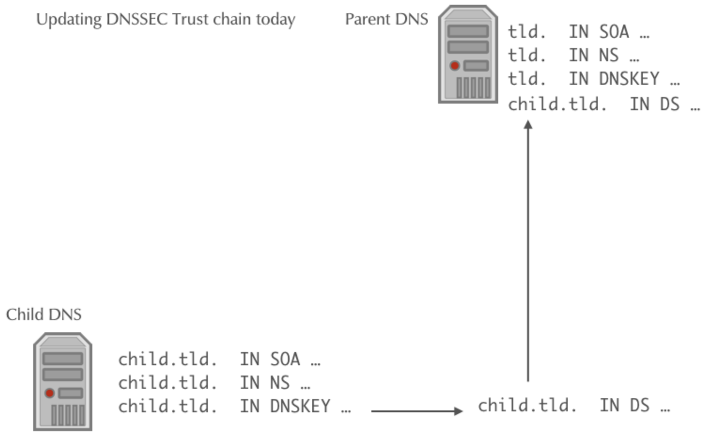
Parent DNS

```
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 DS …
```



Updating DNSSEC Trust chain today

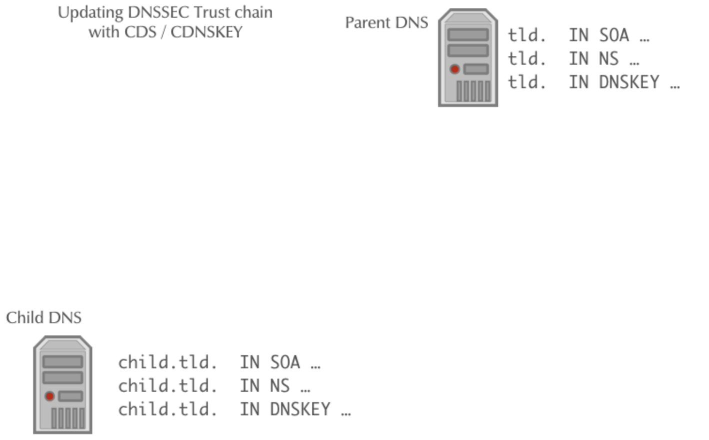
Parent DNS

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

Child DNS

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

```
child.tld.  IN DS …
```



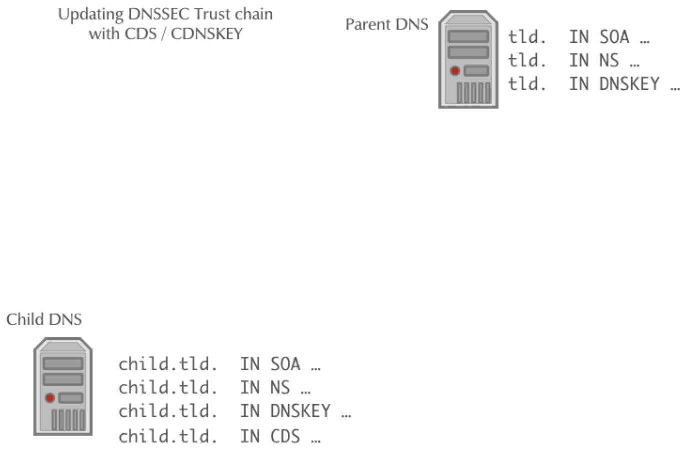
Updating DNSSEC Trust chain
with CDS / CDNSKEY

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 …
```



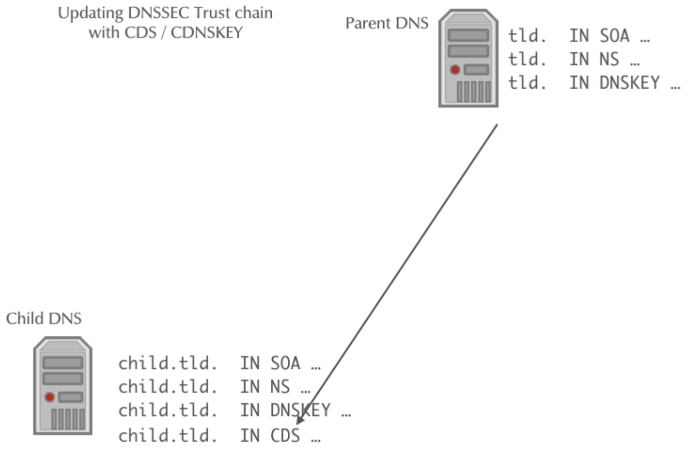
Updating DNSSEC Trust chain
with CDS / CDNSKEY

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 …
```



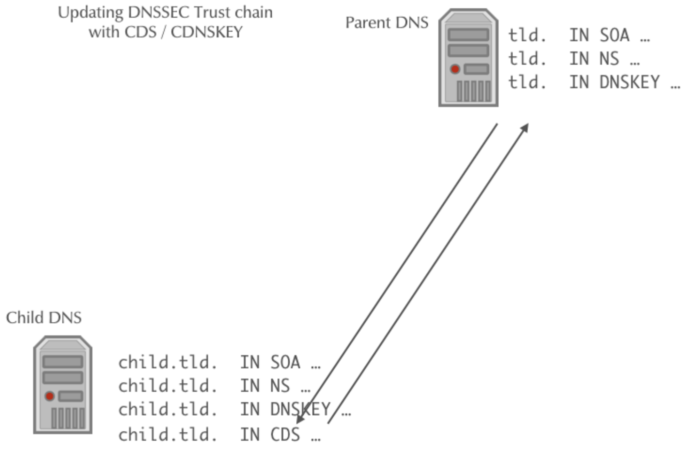
Updating DNSSEC Trust chain
with CDS / CDNSKEY

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 …
```



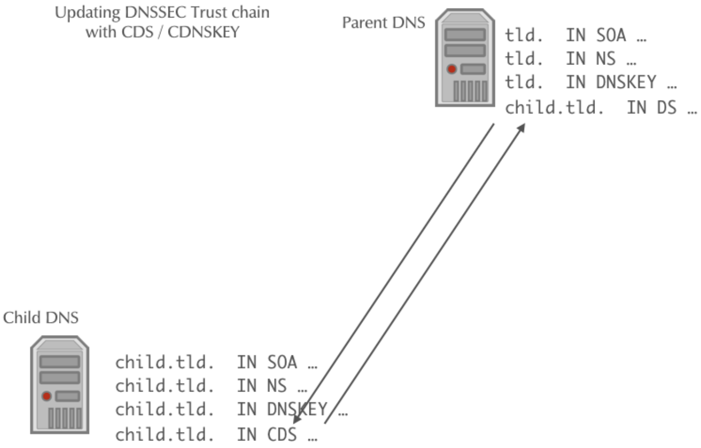
Updating DNSSEC Trust chain
with CDS / CDNSKEY

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 …
```



### 16.1.1. 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/]

### 16.1.2. Bootstrapping DNSSEC trust (RFC 9615)

- The challenge with bootstrapping DNSSEC: there is no established cryptographic trust between the zone and the parent zone DNS server
- This draft 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.
- RFC 9615 "DNSSEC bootstrapping" [https://www.rfc-editor.org/rfc/rfc9615.html] has all the details.
- This RFC is already supported by the `.ch` and `.li` TLDs

### 16.1.3. 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/]

## 16.2. RFC 9859 - GENERALIZED DNS NOTIFICATIONS

- With automated DNSSEC delegation trust maintenance (RFC 8078), the parent domain operator (most often the TLD operator) will "scan" all child domains for new CDS/CDNSKEYs

  This is resource intensive

  It is done is intervals, for example 24 hours - changes to the CDS/CDNSKEY will propagate based on that interval - child zone operators need to wait multiple hours for the change to appear in the parent zone
- RFC 9859 - Generalized DNS Notifications [https://www.rfc-editor.org/rfc/rfc9859.html] (September 2025) extends the use of DNS notification (used for ages in DNS to inform secondary server of a new zone version to trigger a zone transfer) for other purposes
- One is to send a notify from the primary server of a child zone to the primary server of the parent zone to inform the parent zone of a new CDS/CDNSKEY

  No need for the parent to scan all child domains

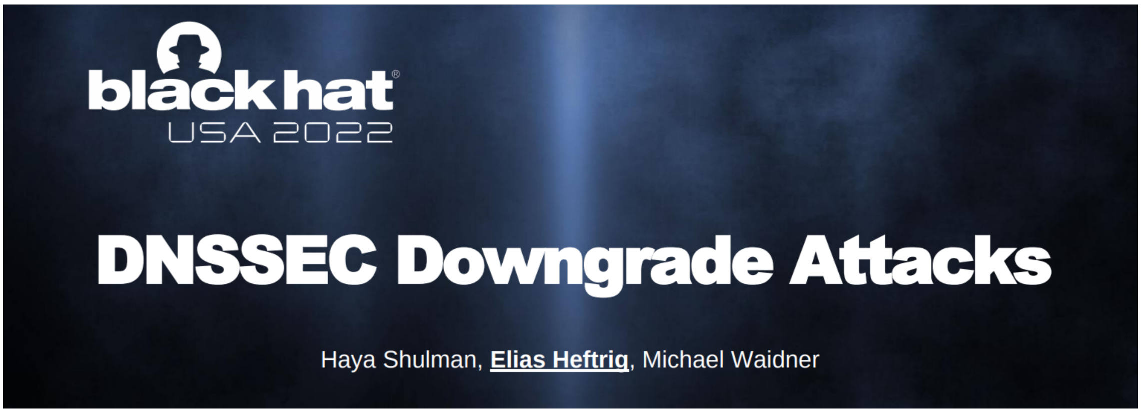
  Faster propagation times (seconds instead of hours)
- Support for "Gerneralized DNS Notifications" is planned for BIND 9.22 (Q2/2026)

# CHAPTER 17. BIND 9 REPOSITORIES FOR UP-TO-DATE BIND 9 VERSIONS

- https://kb.isc.org/docs/isc-packages-for-bind-9 [https://kb.isc.org/docs/isc-packages-for-bind-9]

# CHAPTER 18. DNSSEC BEST-PRACTICES

## 18.1. 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]

## 18.2. SHA1 PHASE OUT IN RED HAT EL 9

[RHEL SHA1 deprecated] | *./img/RHEL-SHA1-deprecated.png*

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]

## 18.3. 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

## 18.4. 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]

### 18.5. 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

### 18.6. 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)
- RFC 9715 - IP Fragmentation Avoidance in DNS over UDP [https://www.rfc-editor.org/rfc/rfc9715.html]

# CHAPTER 19. DNSSEC TROUBLESHOOTING

## 19.1. CHECKING DNS RESOLUTION ISSUES

### 19.1.1. 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?

#### 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).

#### 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?

#### 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).

### 19.1.2. 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.

## 19.2. 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.

### 19.2.1. 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.

### 19.2.2. 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
```

### 19.2.3. 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
```

## 19.3. 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

# CHAPTER 20. DEALING WITH DNSSEC RESOLVER ISSUES

## 20.1. NEGATIVE TRUST ANCHOR

- Often DNSSEC validation issues are caused by operational issues (expired DNSSEC signatures, DS record and DNSKEY KSK mismatch etc)

  Negative Trust Anchors can be used to disable DNSSEC validation for mis-configured domains

- Negative Trust Anchors are defined in RFC 7646 Definition and Use of DNSSEC Negative Trust Anchors [https://tools.ietf.org/html/rfc7646.html]

- Negative trust anchors (nta) disable DNSSEC validation for a specific domain for a certain amount of time

  NTA can be used by operators in case a misconfiguration for a remote DNSSEC signed zone is detected. Care should be take to check that the DNSSEC validation failure is indeed a misconfiguration and not attack

- Domains with an NTA are processed as if there is no trust-anchor for that domain

- NTAs are stored and are persistent across BIND 9 restarts

- BIND 9 checks the domain periodically. Once the domain starts validating again, the NTA for the domain is removed

- NTAs have a lifetime (maximum one week) and expire automatically

- Example: adding an NTA (for 60 seconds):

```
% rndc nta -l 60 fail01.dnssec.works
Negative trust anchor added: fail01.dnssec.works/_default, expires 18-Aug-2016 13:52:19.000
% rndc nta -dump
fail01.dnssec.works: expired 18-Aug-2016 13:52:19.000
% ls -l /var/named/_default.nta
-rw-r--r--. 1 root root 44 Aug 18 13:51 /var/named/_default.nta
% cat /var/named/_default.nta
fail01.dnssec.works. regular 20160818115219
```

- Example: removing an NTA

```
% rndc nta -l 86400 fail02.dnssec.works  # add a NTA for 1 day
Negative trust anchor added: fail02.dnssec.works/_default, expires 19-Aug-2016 13:56:22.000

% rndc nta -dump
fail02.dnssec.works: expiry 19-Aug-2016 13:56:22.000

% rndc nta -r fail02.dnssec.works # remove the NTA
Negative trust anchor removed: fail02.dnssec.works/_default

% rndc nta -dump    # NTA is now gone
```

## 20.2. EXERCISE: NEGATIVE TRUST ANCHOR FOR FAILNN.DNSSEC.WORKS

- In the chapter on DNSSEC troubleshooting we've learned about the broken DNS zones `fail01.dnssec.works` to `fail05.dnssec.works`

- Create negative trust anchor (NTA) for `fail01` to `fail03` in your DNSSEC validating resolver machine `dnsrNN`

- Check that a client can now retrieve DNS data from these zones

```
$ dig @127.0.0.1 fail01.dnssec.works A
```

- Remove the NTA for `fail01.dnssec.works`. Check that the zone now does not validate and return `SERVFAIL`

## 20.3. RECOMMENDATIONS FOR DNS RESOLVER OPERATORS

- Make sure your DNSSEC validating resolver supports DNSSEC negative trust anchor
- Test the negative trust anchor function
- Document how to add a negative trust anchor
- If your software does not support automatic removal of NTAs, implement an automation (connected to your DNS resolver monitoring)
- Do not implement automatic NTA additions, activating an NTA should only occur after human inspection of the DNSSEC validation problem
- Read the Internet Draft Recommendations for DNSSEC Resolvers Operators [https://www.ietf.org/archive/id/draft-ietf-dnsop-dnssec-validator-requirements-01.html]

## 20.4. DNSSEC RESOLVER AND TIME

- DNSSEC validation is time sensitive

  The DNSSEC signatures have expire and inception (*become valid*) times that the DNS resolver must check

- Without a correct time on the DNSSEC resolver machine, DNSSEC validation might fail
- Recommendation: Implement automatic time synchronization for the DNSSEC resolver (NTP, PTP)
- Make sure the time synchronization has no dependency on (DNSSEC) DNS name resolution (no circular dependecy or *chicken-and-egg-problem*)

  A great usecase for Precision Time Protocol (PTP https://en.wikipedia.org/wiki/Precision_Time_Protocol [https://en.wikipedia.org/wiki/Precision_Time_Protocol]) if available in the network

# CHAPTER 21. DNSSEC RISKS

## 21.1. RISK: OPERATING A DNSSEC VALIDATING RESOLVER

- A DNSSEC validating resolver will respond with the DNS error `SERVFAIL` to the DNS client whenever it detects a mismatch in the DNSSEC chain of trust or a bogus or expired DNSSEC signature

  `SERVFAIL` is not DNSSEC specific, there are many error situations that can result in an `SERVFAIL` response

  The end user cannot detect that the DNS name resolution fails because of DNSSEC security and will blame the operator of the DNS resolver for the outage image::./img/dnssec-stub-resolver6.png[scaled-width=100%]

### 21.1.1. Solutions to mitigate this issue

- Open communication with the end user about DNSSEC issues
- Active monitoring for DNSSEC validation issues on the DNS resolver
- Prepare for the use of *Negative Trust Anchor* (NTA)

## 21.2. RISK: DNSSEC SIGNED ZONE WITH NSEC3

- An authoritative DNS server with an NSEC3 DNSSEC signed zone must calculate one or more SHA1 hash(es) for every negative DNS response (NXDOMAIN or NODATA)

  This yields the risk of Denial-of-Service attacks against the authoritative server due to CPU resource exhaustion

### 21.2.1. Solutions: DNSSEC signed zones with NSEC3

- Consider employing NSEC instead of NSEC3. "zone walking" might be no or the smaller issue
- Provision enough CPU resources for the maximum amount or queries that can reach the DNS server (network capacity)

  Test the server capacity
- Enable response rate limiting on the authoritative DNS server(s)

## 21.3. RISK: DNSSEC SIGNED ZONE

- Errors during the signing process
- Error while publishing/replacing the DS record in the parent zone
- Error while refreshing the DNSSEC signatures (RRSIG records)
- Errors during key rollover
- Operational issues because of too large DNS response packets (>1232 byte)

- DNSSEC validation issues because of messy DNS resolution path (CNAME redirections between DNSSEC signed and unsigned zones)
- DNS errors because of Parent/Child zone disagreement (NS records in parent zone delegation do not match NS records in the zone)

#### 21.3.1. Solutions: DNSSEC signed zones

- Errors during the signing process:

  Learn and test on *toy* domains, get practice

  Get external know how for the introduction of DNSSEC in your zones

  Get DNSSEC training for the DNS administrators

  Adjust the TTL values of the zone (should be below 24 hours, better below 2 hours)

  Adjust the SOA values of the zone (Refresh/Retry/Expire/NegTTL). Too large values can be problematic.

- Errors while publishing/replacing the DS record in the parent zone

  Test the new DS record locally (for example with `unbound-host`) before submitting to the operator of the parent zone

  Test with a locally created root DNS server

  Monitor the status of the DS record and the KSK in the zone

  Automate the update of the DS record with an API

- Errors while refreshing the DNSSEC signatures (RRSIG records)

  Automate the refresh of signatures (BIND 9, PowerDNS, KnotDNS, Windows 2012/2016, OpenDNSSEC …)

  Monitor the expiry of signatures in the zone

- Errors during key rollover

  Become familiar with DNSSEC key rollovers; practice them

  Get external help, DNSSEC training

  Automate key rollovers

  Monitor key rollovers

  Create a written key rollover journal (helps when working as a team)

- Operational issues because of too large DNS response packets (>1232 byte)

  Adjust the EDNS0 buffer size on all authoritative DNS servers of the DNSSEC zone to 1232 byte

  Carefully select the domain names to make use of label compression

  Prevent large DNS resource record sets

  Stay away from large RSA DNSSEC keys and signatures, use EC algorithms

    - Prevent CNAME chains
    - Enable *minimal-responses*
    - Enable *minimal-any*
- DNSSEC validation issues because of messy DNS resolution path (CNAME redirections between DNSSEC signed and unsigned zones)
    - Balance the name resolution path between redundancy and DNS resolution steps (always a good idea)
    - Prevent CNAME redirection from DNSSEC signed zones to in-secure DNS zones
- DNS errors because of Parent-Child-Zone Disagreement (NS records in parent zone delegation do not match NS records in the zone)
    - Make sure the delegation NS records in the parent zone match the NS records in the zone
    - Monitor for parent-child NS record mismatch

### 21.4. RISK: DNSSEC

- Is DNSSEC a risk for the network?
    - Yes, but it is a risk that can be mitigated with planning, automation, and careful work
    - The DNSSEC benefits outweighs the risks

### 21.5. SEE ALSO

- DNS and DNSSEC Monitoring scripts: https://github.com/menandmice-services/dns-monitoring-scripts [https://github.com/menandmice-services/dns-monitoring-scripts]
- When parents and children disagree: Diving into DNS delegation inconsistency https://www.caida.org/publications/papers/2020/when_parents_children_disagree/ [https://www.caida.org/publications/papers/2020/when_parents_children_disagree/]

# CHAPTER 22. DNSSEC AND SPLIT-HORIZON DNS

- Definition of *split-horizon DNS*: a disjunct namespace where multiple copies of the same DNS names exist with different data
- Classic example: having the main DNS domain both external (Internet) and internal (LAN) with different content
- DNS (and DNSSEC) assumes a unified namespace — there can only be a single version of a RRSet

    *split-horizon* breaks this assumtion

- *split-horizon DNS* is often a *legacy* configuration — it creates all kinds of problems and the same properties can be build with a unified namespace

    … but it is very hard to migrate away from a *split-horizon DNS* (so many environments don't do it

    Recommendation:

        start a new namespace branch (new domain for internal or external DNS) and inplement new services on the new namespace

        over time the *split-horizon DNS* will dry out and can be removed

        if you don't have *split-horizon DNS*, don't start using it!

## 22.1. CHALLENGES WITH DNSSEC AND SPLIT-HORIZON DNS

- A DNS zone having a DS-Record **must** be DNSSEC signed with **that** key
- In a *split-horizon DNS*, the interal DNS only covers the second-level domain (SLD), not the TLD

    but the DS-Record will be looked up in top-level-domain (in the Internet)

- If a DNS resolver has a DS-record in the cache, and it receives DNS records from this zone without signatures (RRSig), it will not resolve these domain names (SERVFAIL error)

## 22.2. POSSIBLE SOLUTIONS FOR DNSSEC IN A SPLIT-HORIZON DNS ENVIRONMENT

### 22.2.1. Migrate away from split-horizon DNS

- More stable
- More secure

    *split-horizon DNS* is prone to leak internal DNS data to the Internet

### 22.2.2. Disable DNSSEC validation on all internal DNS resolver

- Weakens the internal network security — enables DNS cache poisioning
- Modern operating systems implement DNSSEC validation functions (Apple macOS/iOS, Linux systemd-resolved, OpenBSD "unwind", FreeBSD "unbound") — extra work to disable DNSSEC validation on OS level

#### 22.2.3. Sign internal and external zones with the same DNSSEC keys

- DNSSEC keys must be synced between multiple DNSSEC signer (brittle)
- Difficult with different DNS server products (MS DNS Server internally, Unix based DNS or Appliance for external DNS)
- Keys and DS-Records needs to be in sync

#### 22.2.4. Augmented internal local root zone

- Fetch the Internet root zone

  needs to be done in intervals — whenever the root zone changes — must be automated
- Remove all DNSSEC signtures from the root zone
- Remove all DNSSEC keys from the root zone
- Add own public DNSSEC keys to the root zone
- Make changes to the root zone (create new, internal delegations, internal top-level-domains etc)
- Sign the root zone with the own DNSSEC keys
- publish the root zone on internal DNS-resolver or internal authoritative DNS-server and configure internal root-hints on all resolvers

  Modern operating systems implement DNSSEC validation functions (Apple macOS/iOS, Linux systemd-resolved, OpenBSD "unwind", FreeBSD "unbound") — extra work to distribute root-hints or DNSSEC trust-anchors on OS level

# CHAPTER 23. UPCOMING DNSSEC CHANGES

## 23.1. "DRY-RUN" DNSSEC

- Switching on DNSSEC (by placing a DS record in the parent zone) can be both exciting and frightening

  Even with good preparation and testing on the authoritative server side, you can never sure what *brokenness* is out there in the Internet on the resolve side (misbehaving Firewalls, broken DNS load balancer, weird endpoint security, …)

- The "dry-run" IETF draft proposes a way to signal a DNSSEC test by using a special "dry-run" flag in the DS record placed into the parent zone

- DNS resolver supporting this extension that see the "dry-run" flag will try to validate the DNSSEC signed zone, but will not fail to resolve the zone but treat it insecure

  All DNSSEC validation errors will be reported back to the operator of the DNS zone using the "DNS error reporting" function (see chapter above).

  The idea is to first deploy a new DNSSEC signed zone in this "dry-run" mode, collect error messages (and positive feedback) over a period of time, and then remove the "dry-run" flag from the DS record to convert the zone into a validating DNSSEC zone

  The current proposal is to have the flag in the digest/hash-algorithm field of the DS record

- A client can indicate (possibly via an EDNS flag) towards a DNSSEC validating resolver to have a "dry run" DNSSEC zone validated (with normal DNSSEC error reporting = SERVFAIL and extended DNS error)

- DNS resolver that does not support the "dry run" DNSSEC DS record will ignore it and will treat the zone as insecure (no DNSSEC validation)

- The "dry run" DNSSEC DS record can also be used to test drive a DNSSEC KSK key rollover without risks to break the zone

- Draft "dry run DNSSEC" (A draft is not yet a standard, and might never be): https://datatracker.ietf.org/doc/draft-yorgos-dnsop-dry-run-dnssec/ [https://datatracker.ietf.org/doc/draft-yorgos-dnsop-dry-run-dnssec/]

- This draft has not yet been implemented in production DNS server software

- This draft has not yet been adopted by the IETF working DNSOP working group, but people at the IETF liked the idea

## 23.2. BIND 9.21.12 "MANUAL-MODE" DNSSEC-POLICY

- A new option *manual-mode* has been added to dnssec-policy.

  The intended use is that if it is enabled, it will not automatically move to the next state transition, but instead the transition is logged.

  Only after manual confirmation with `rndc dnssec -step` the transition is made

- This helps with migrations from older `auto-dnssec maintain;` configurations to DNSSEC policy based configurations

The administrator can check every step of the transition before it becomes active

- `manual-mode` DNSSEC will be availble in BIND 9.22.0 (Q2/Q3 2026)