



**RIPE NCC**  
RIPE NETWORK COORDINATION CENTRE

# DNSSEC

Training Course

# Schedule



09:00 - 09:30	<b>Coffee, Tea</b>
11:00 - 11:15	<b>Break</b>
13:00 - 14:00	<b>Lunch</b>
15:30 - 15:45	<b>Break</b>
17:30	<b>End</b>

# Introduction



- **Name**
- **Number on the list**
- **Experience**
  - DNS
  - DNSSEC
  - Cryptography
- **Goals**

# Overview



- 1. Introduction to DNS**
  - *Exercise A: Creating a Zone file*
- 2. DNS Vulnerabilities**
- 3. Introduction to Cryptography**
- 4. Introduction to DNSSEC**
- 5. New Resource Records for DNSSEC**
  - *Exercise B: Update the zone file in Bind*
  - *Exercise C: Using DIG to Find Information*
- 6. Delegating Signing Authority**
- 7. Setting up a Secure Zone**
  - *Exercise D: Configure DNSSEC for the Domain*
- 8. Flags and Scenarios**
- 9. Key Rollovers**
- 10. DANE**
- 11. Troubleshooting, Tips, Tricks**
  - *Exercise E: Check and Troubleshoot DNSSEC*





# Introduction to DNS

## Section 1

# In the Beginning...



- **The Internet was small**
  - fewer than 100 hosts
- **Everybody knew everybody else**
- **Centralised: **host** file distributed to everyone**
- **But it didn't scale**

# What is DNS ?



- **Domain Name System**
- **RFC1035**
- **Distributed database**
- **Translation**

name -> IP address

IP address -> name

# What is DNS ?



- System to convert names to IP addresses:

**www.ripe.net**



**193.0.6.139**

**www.ripe.net**



**2001:67c:2e8:22::c100:68b**

# What is DNS ?



Reverse DNS:

**139.6.0.193.in-addr.arpa**



**www.ripe.net**

or

**b.8.6.0.0.0.1.c.  
0.0.0.0.0.0.0.0.2.2.0.0.8.e.  
2.0.c.6.7.0.1.0.0.2.ip6.arpa**



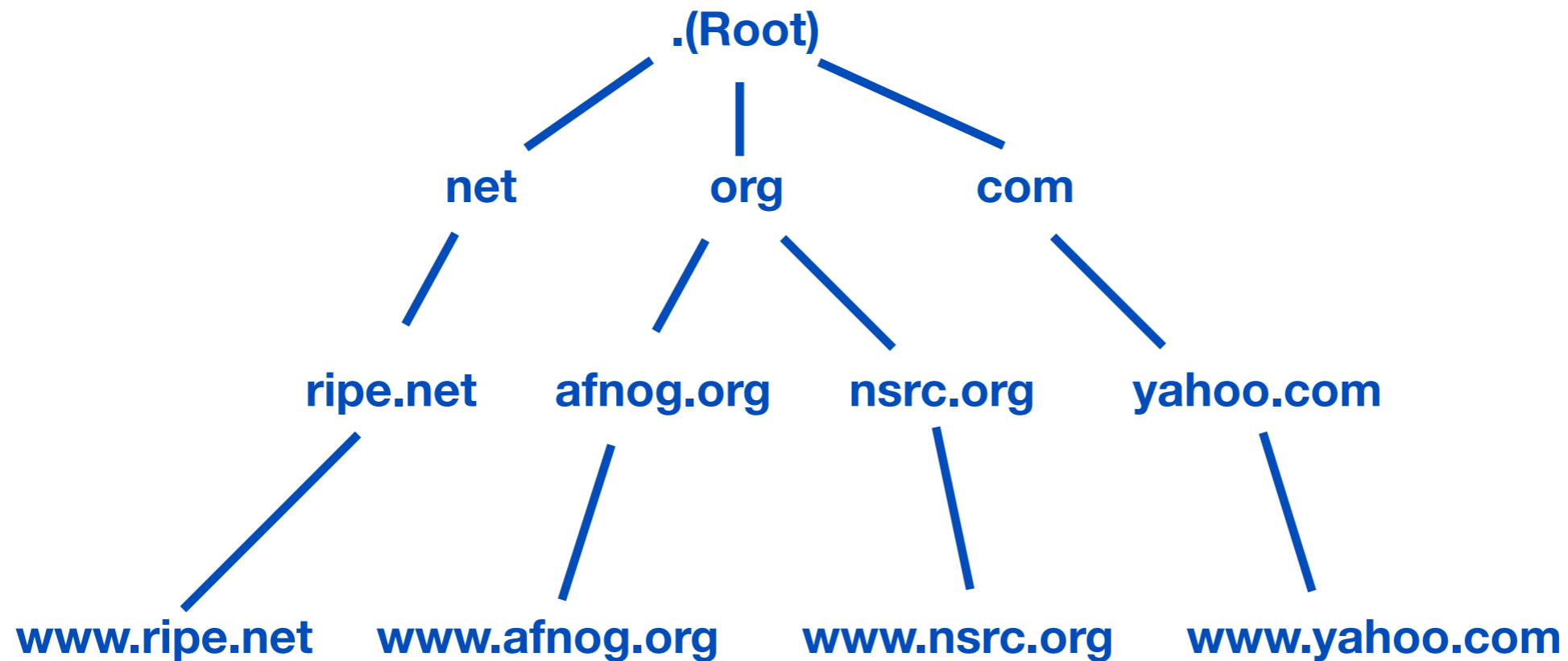
**www.ripe.net**

# DNS



- **Case insensitive**
- **Transport is either UDP or TCP on port 53**
- **Indexed by “domain names”**
  - A “domain name” is a sequence of labels
    - `www.ripe.net`
    - `emi.ac.ma`

# DNS is Hierarchical



- **DNS administration shared**
  - No single central entity administers all data
- **Delegation = distribution of administration**

# DNS is a Database



- **Contains different types of data:**
  - IP Addresses
  - Where to send email
  - Who is responsible
  - Geographical info
  - etc..





Clients use **Stub Resolvers**



Stub resolvers ask **Recursive resolver**

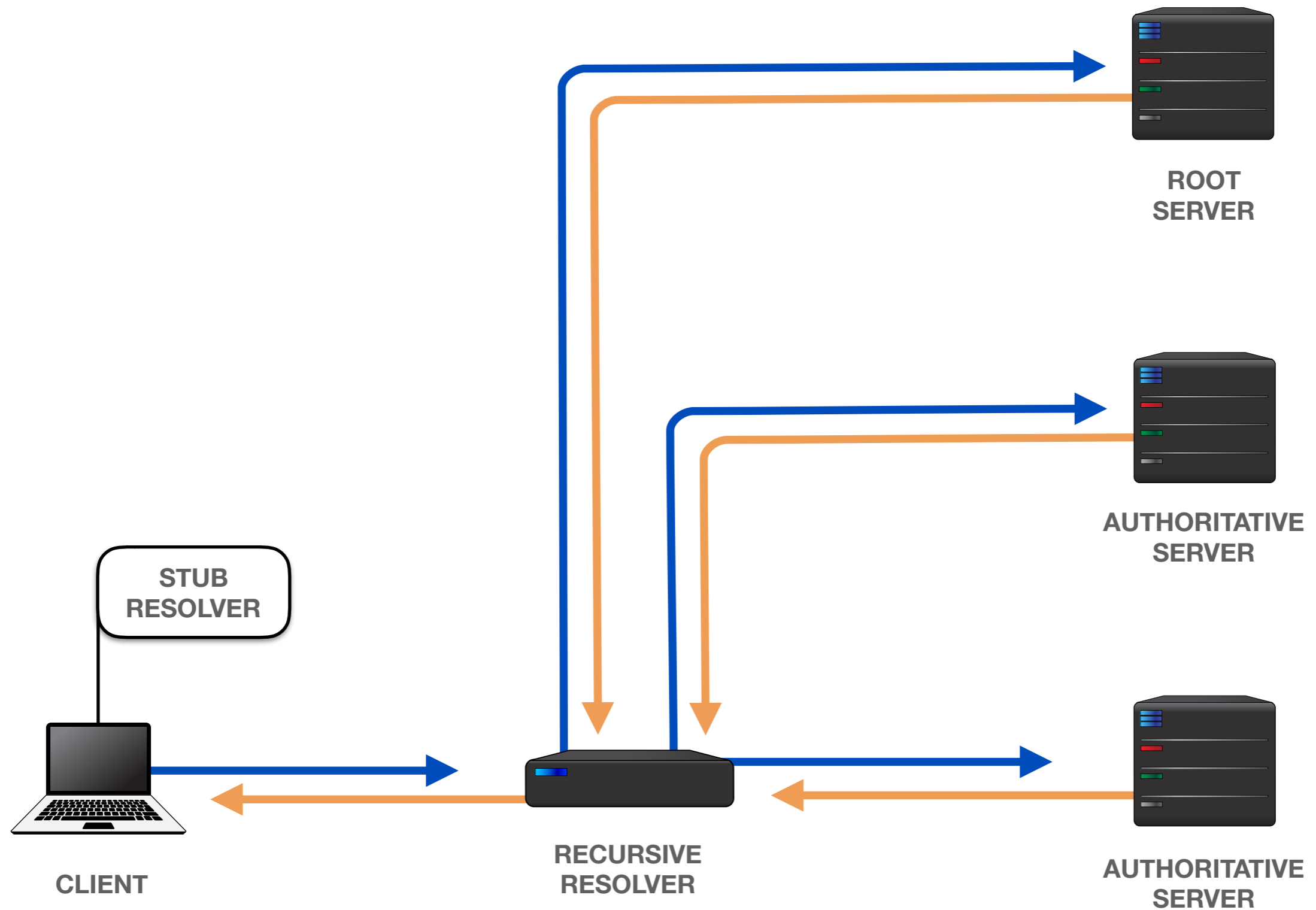


**Recursive resolver** will find answer on behalf of client

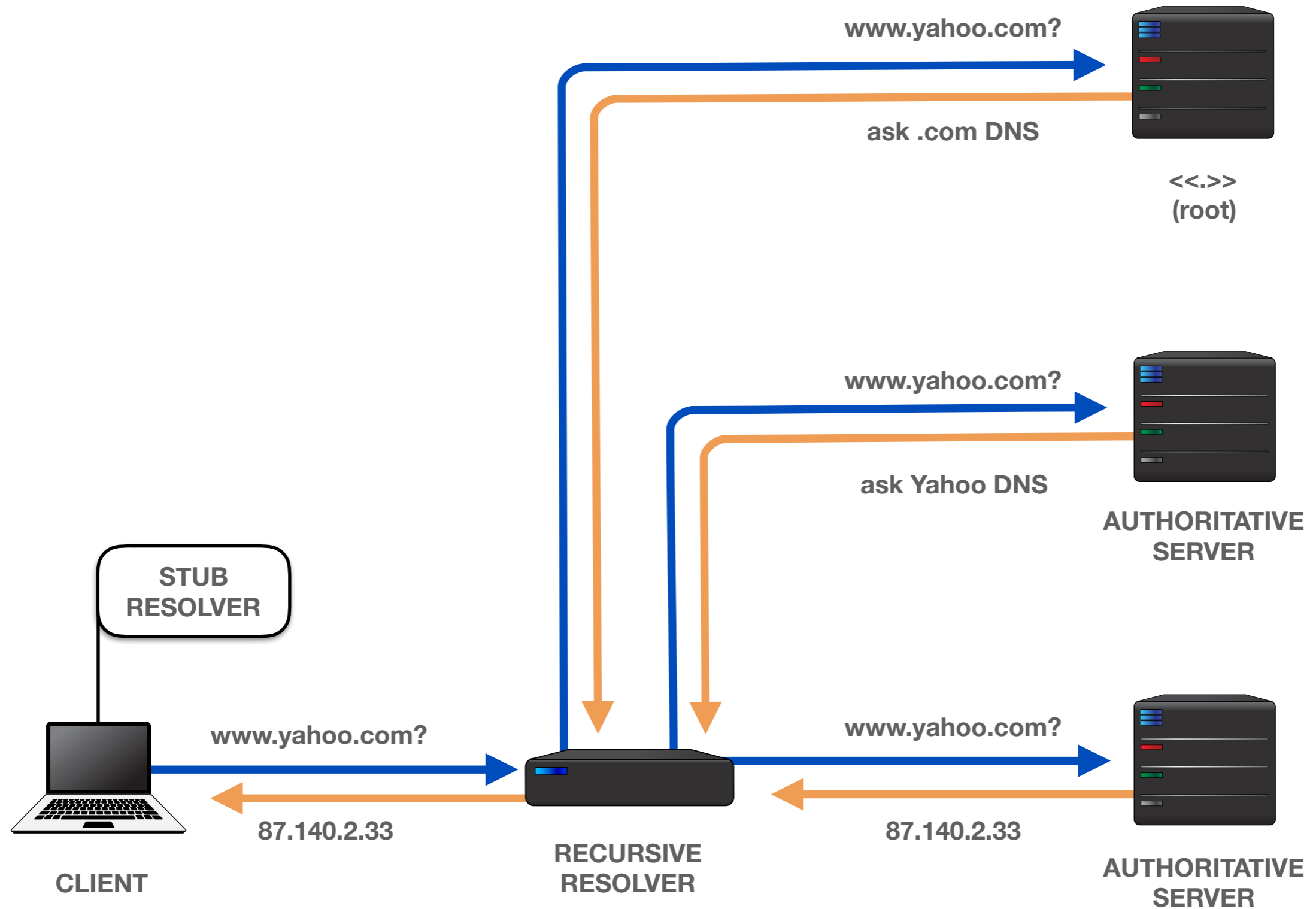


**Recursive resolver** keeps asking **Servers** top (root) to bottom  
until it finds the answer

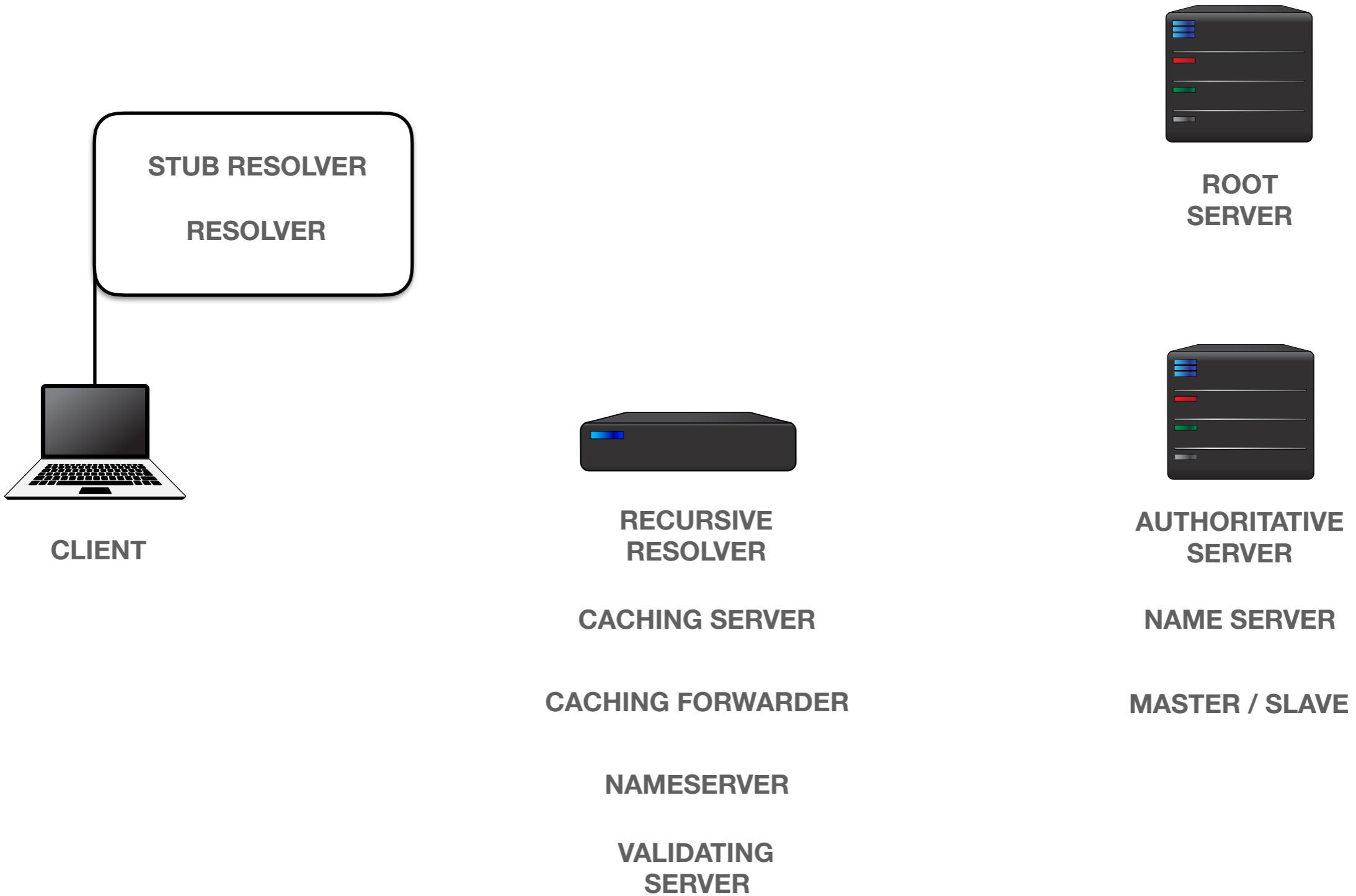
# How Does DNS Work?



# Example of a DNS query



# Terminology





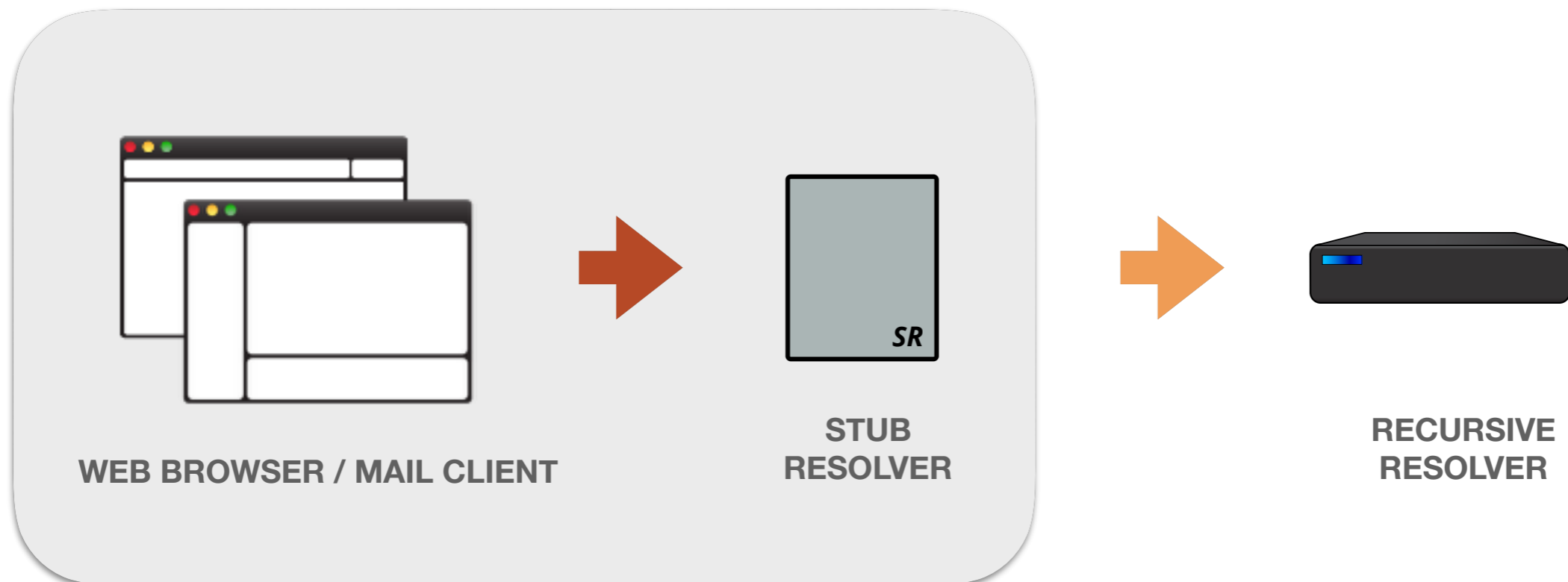
# **Recursion is Important**

**No single machine can have all the  
information in the world**

# How the Client Finds the Recursive Resolver?



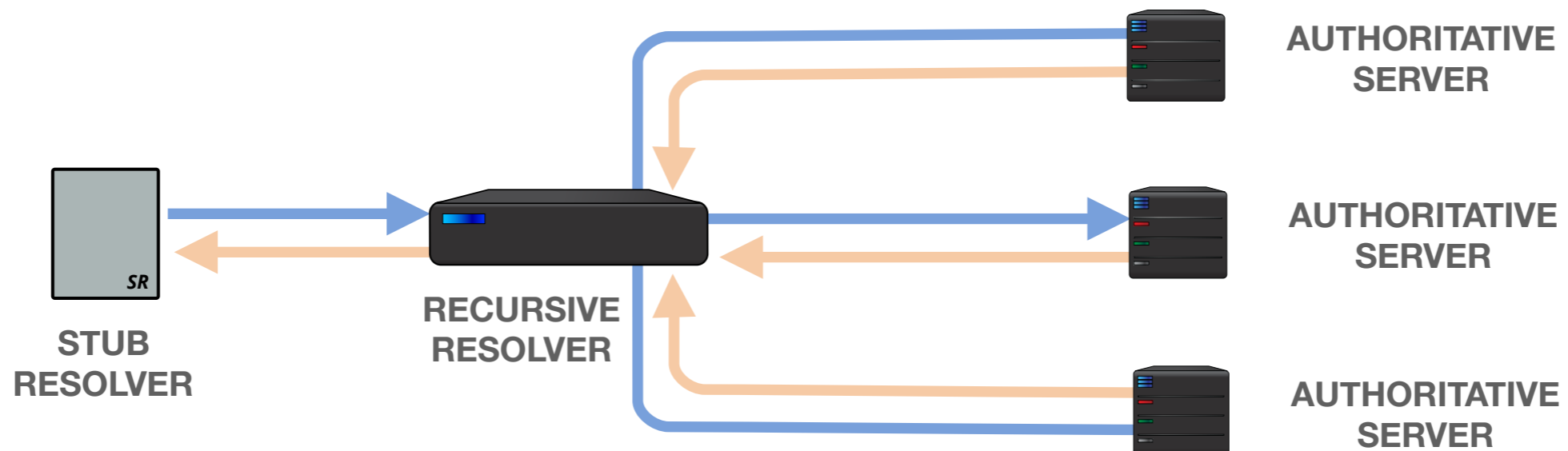
- **Client** uses OS's **stub resolver** to find **recursive resolver's** IP address
- The address of the resolver can be configured manually, or received via DHCP



# Recursive Resolver



- Queried by **stub resolvers** to resolve names
- They query the **authoritative servers** for the answer and serve it back
- Results are cached based on the Time To Live (TTL) in the zone





**MOST FAMOUS RESOLVER?**

**8.8.8.8**



# How Does a Recursive Resolver Find the Root?



- Each **recursive resolver** has a list of **ROOT NAMESERVERS** and their IPs  
(A-M.ROOT-SERVERS.NET)



- **First query to 192.112.36.4 (G.ROOT-SERVERS.NET.)**

**In BIND: `named.cache`**

# Root servers



- Top-bottom approach
- 13 Root servers
- “Empty label” covers the “.” zone



**ROOT  
SERVER**

# Top Level Domains (TLDs)



**gTLD** - Generic Top-Level Domain

**.com**                      **.net**                      **.org**                      ...

**ccTLD** - Country-Code Top-Level Domain

**.it**                                      **.nl**                                      **.pt**                                      ...

**New TLDs**

**.tourism**                      **.newyork**                      **.museum**                      ...

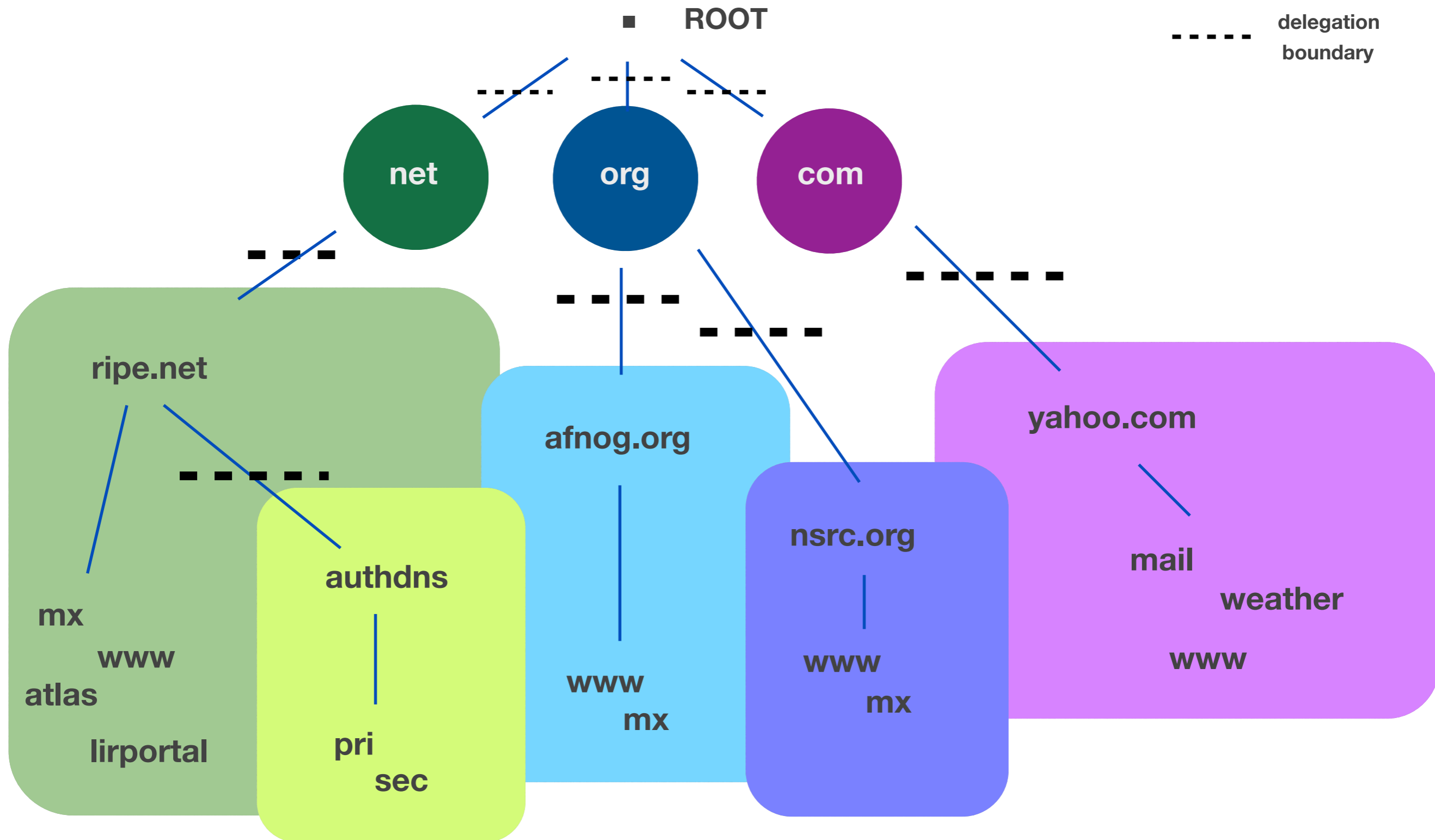
**IDN** - Internationalised Domain Names

**.ایران**                                      **.MOCKBA**                                      ...

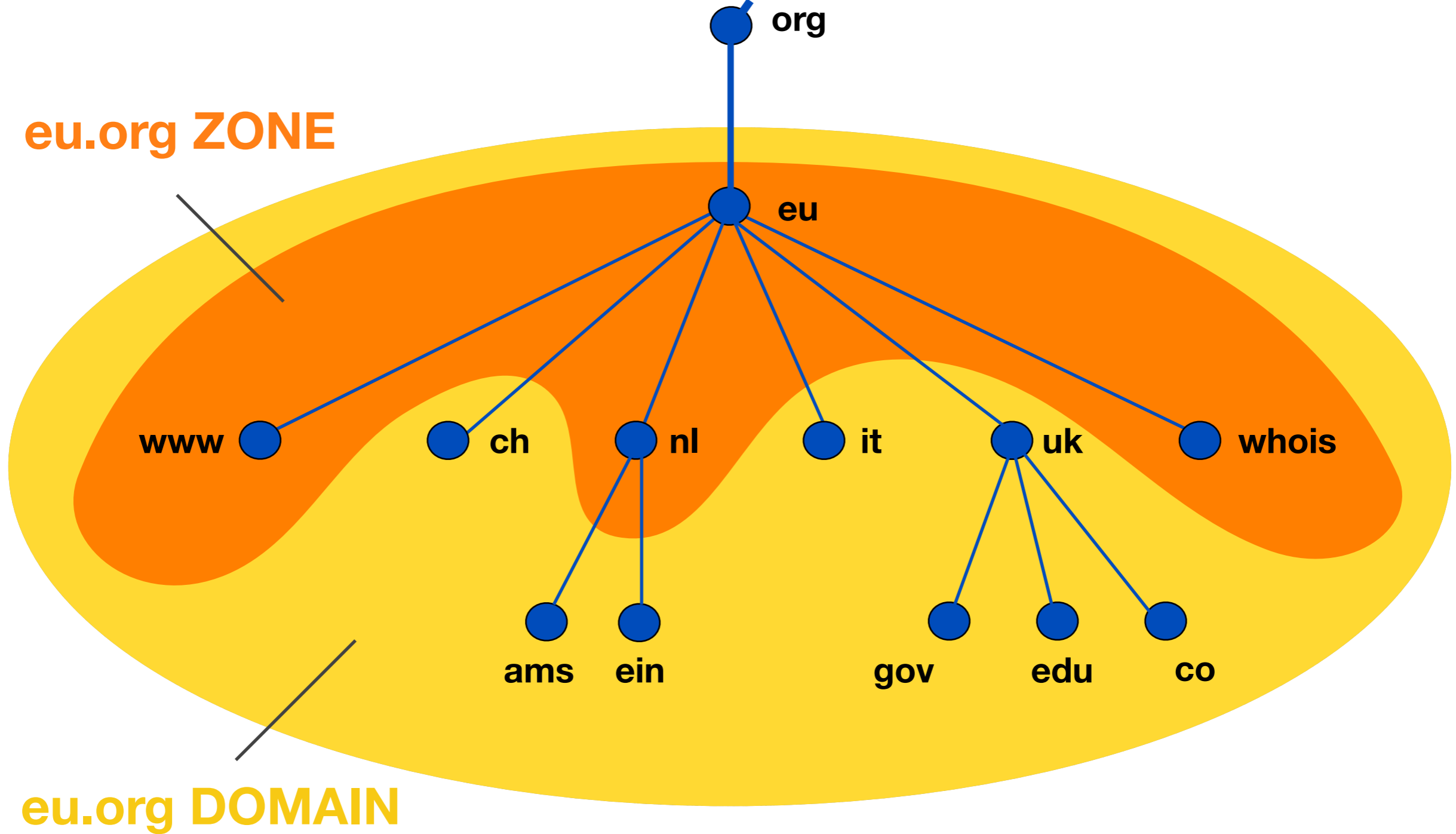
# Delegation



.....  
delegation  
boundary



# Delegation: Domain vs Zone



# DNS Query



- **Every DNS Query consists of:**
  - qname: a domain name (i.e. www.ripe.net)
  - qtype: A, AAAA, MX, CNAME, PTR, SRV, TXT, NS
  - qclass: IN, CH, HS (only IN used today)
  - Flags: QR, RD, EDNS Opt, DO, AD, etc.



# Resource record types



- **SOA: Start of Authority**
- **NS: Name Server**
- **A: IPv4 address record**
- **AAAA: IPv6 address record**
- **CNAME: Canonical Name (i.e. Alias)**
- **MX: Mail Exchanger**
- **PTR: Pointer (for reverse DNS)**

# NS Record



- Name Server record
- Delegates a DNS subtree from parent
  - Lists the authoritative servers for the zone
- Appears in both parent and child zones
- rdata contains hostname of the DNS server

ripe.net. IN NS pri.authdns.ripe.net.

“owner”,  
child’s domain

child’s DNS server



# A Record



- IPv4 Address Record
- rdata contains an IPv4 address

www.ripe.net. IN A 193.0.6.139

“owner”

host or domain

IPv4 address

# AAAA Record



- IPv6 Address Record
- rdata contains an IPv6 address

www.ripe.net. IN AAAA 2001:67c:2e8:22::c100:68b

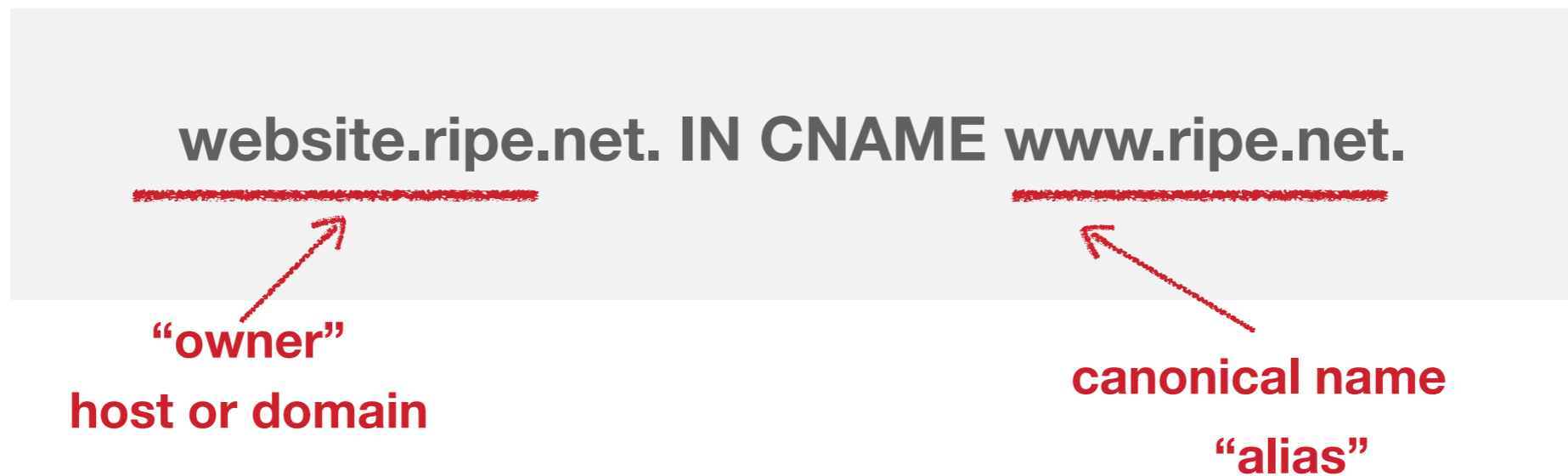
“owner”  
host or domain

IPv6 address

# CNAME Record



- An “alias”
  - maps one name to another (regardless of type)
- rdata contains the mapped domain name (canonical name)



# MX Record



- **Mail Exchanger: defines the host receiving mail**

```
ripe.net. IN MX 10 mail1.ripe.net.  
ripe.net. IN MX 20 mail2.ripe.net.
```

↑  
preference

↑  
host receiving email

**lower preference = higher priority**

# SOA Record (Start Of Authority)



- **Defines the start of a new zone**
  - also, important parameters for the zone
- **Always appears at the beginning of the zone**
- **Serial number should be incremented on zone content updates**

serial number



```
ripe.net. IN SOA pri.authdns.ripe.net. dns.ripe.net. 1399456383 3600  
600 864000 300
```

# Authority: Who Owns This Data



- Query the SOA (Start of Authority) for a domain:

```
# dig SOA <domain>
```

```
...
```

```
;; AUTHORITY SECTION:
```

```
<domain>. 860 IN SOA ns.<domain>. root.<domain>.
```

label

For replication between nameservers

How often slave server checks master for new data

If refresh fails, how often retry

If master failed to answer for this long, don't hand out this data to clients

If you got a negative answer (record doesn't exist) when you query the zone file, cache it for so long

SOA  
name of master  
server

RP  
(responsible person's  
email)  
Replace first "." by "@"

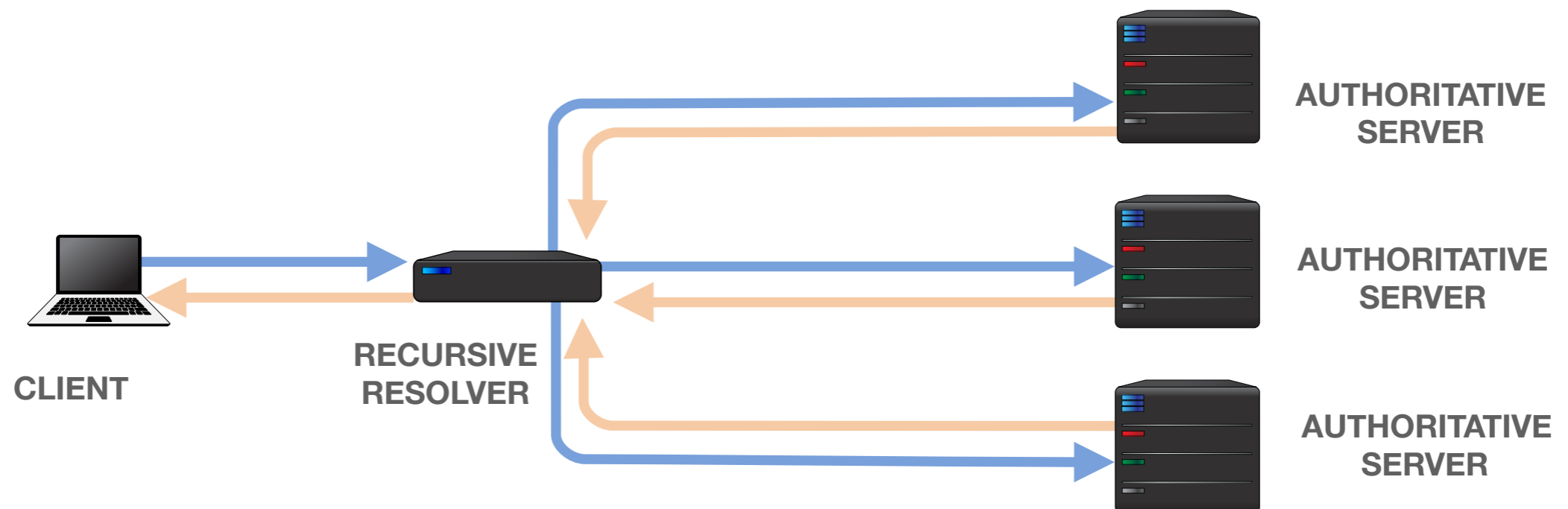
```
200702270 ; serial  
28800 ; refresh  
14400 ; retry  
3600000 ; expire  
86400 ; neg ttl
```

```
...
```

# Authoritative Server



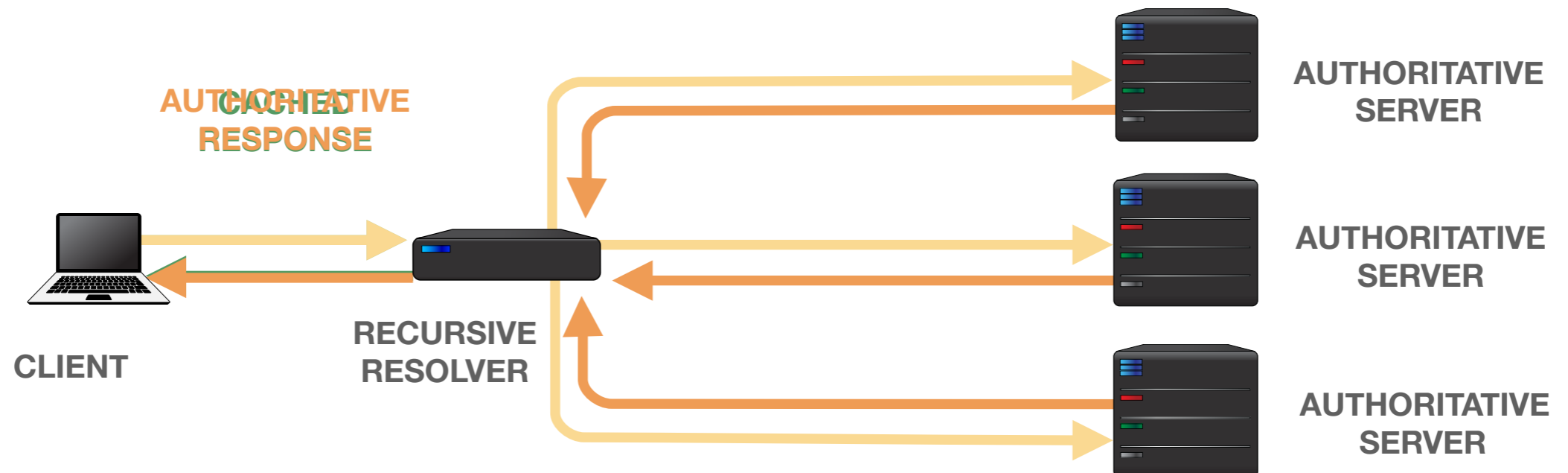
- Contains records in its **zone file**
- Only answer queries for data under **its authority**
- If can't answer, it points to authority
  - but doesn't query recursively.



# Caching vs Authoritative



- Answers are **cached** by the recursive resolver
- **Recursive Resolver** returns the same answer to a repeated query
- this way the response time is **lower**





# Time to Live (TTL)



## *How fresh is your data?*

- Maximum time that an answer can be cached
- TTL values decrement and expire

Try asking for A record repeatedly

```
# dig www.yahoo.com
```



# ***Creating a Zone File***

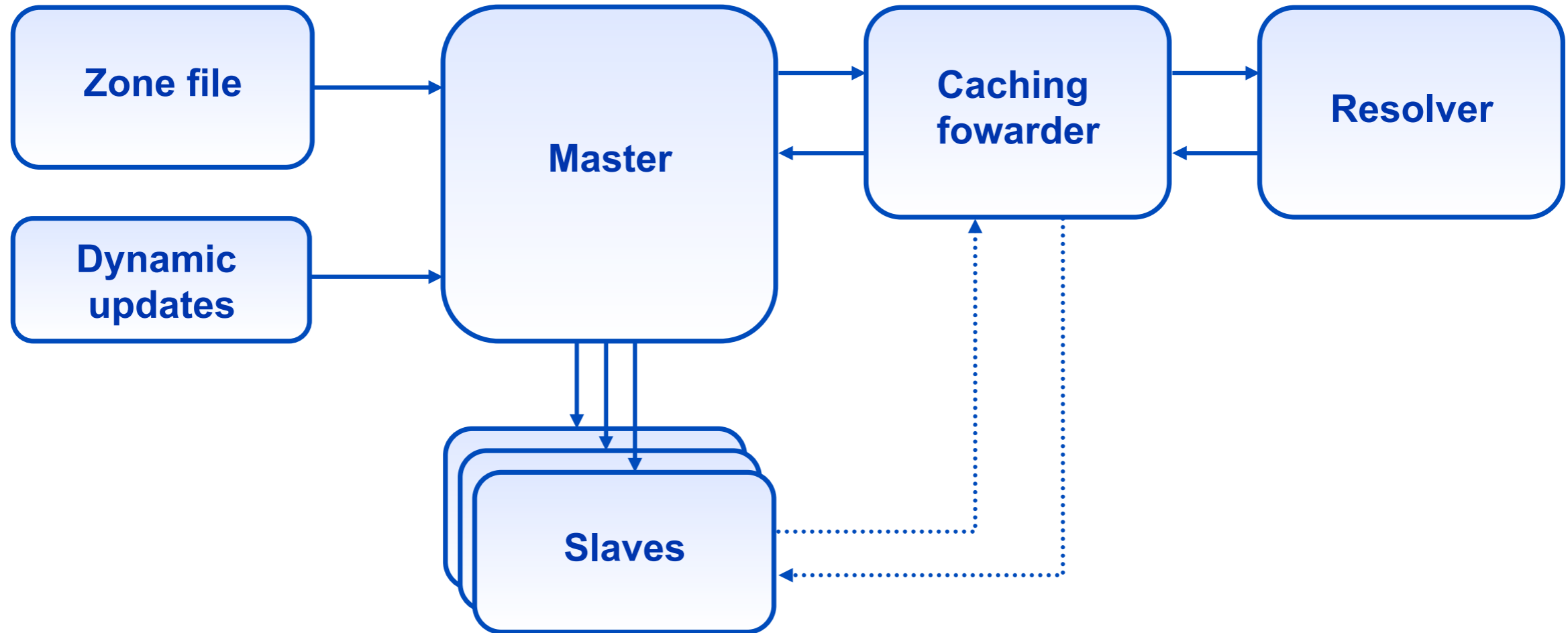
## *Exercise A*



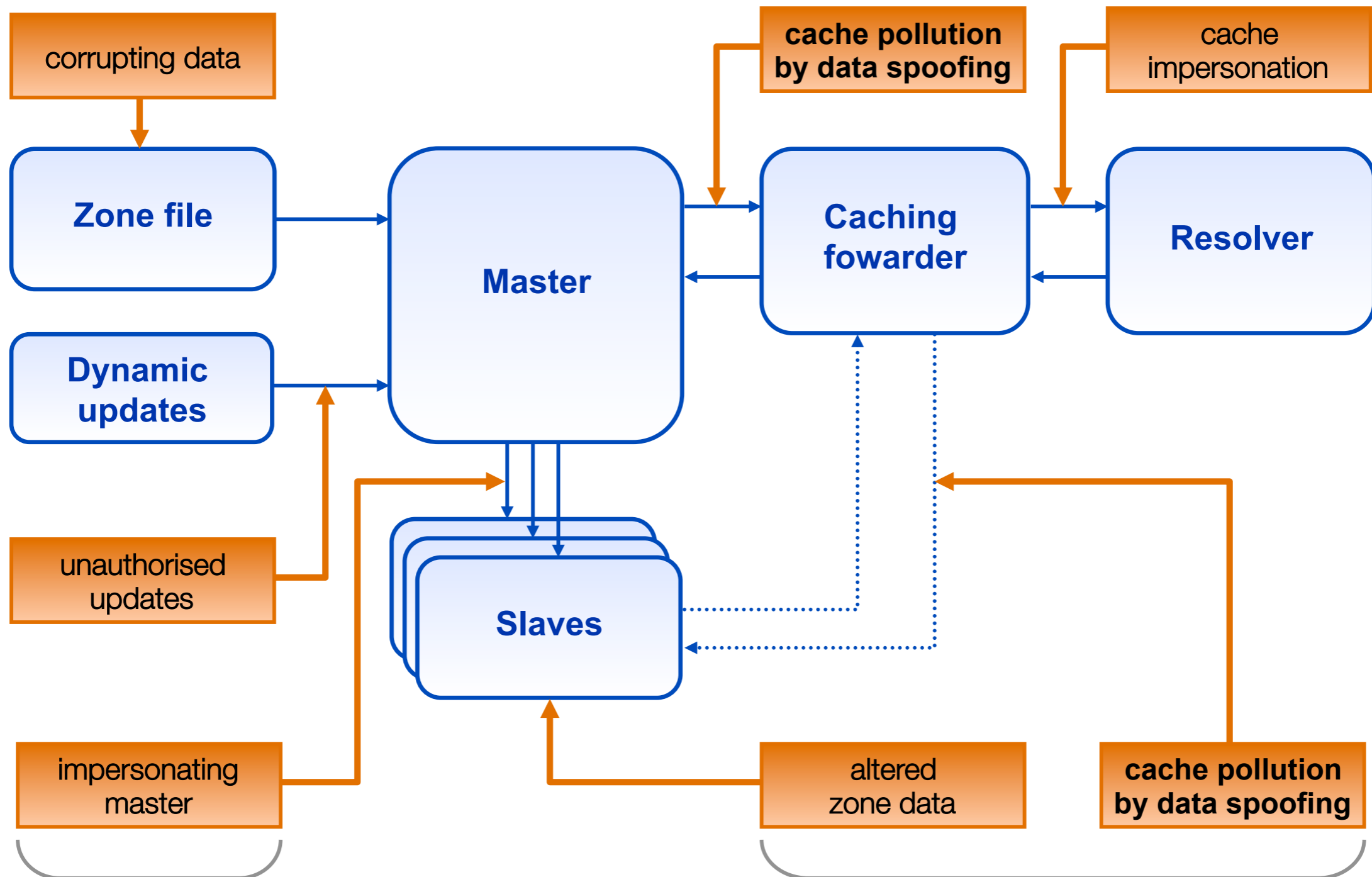
# DNS Vulnerabilities

## Section 2

# DNS Data Flow



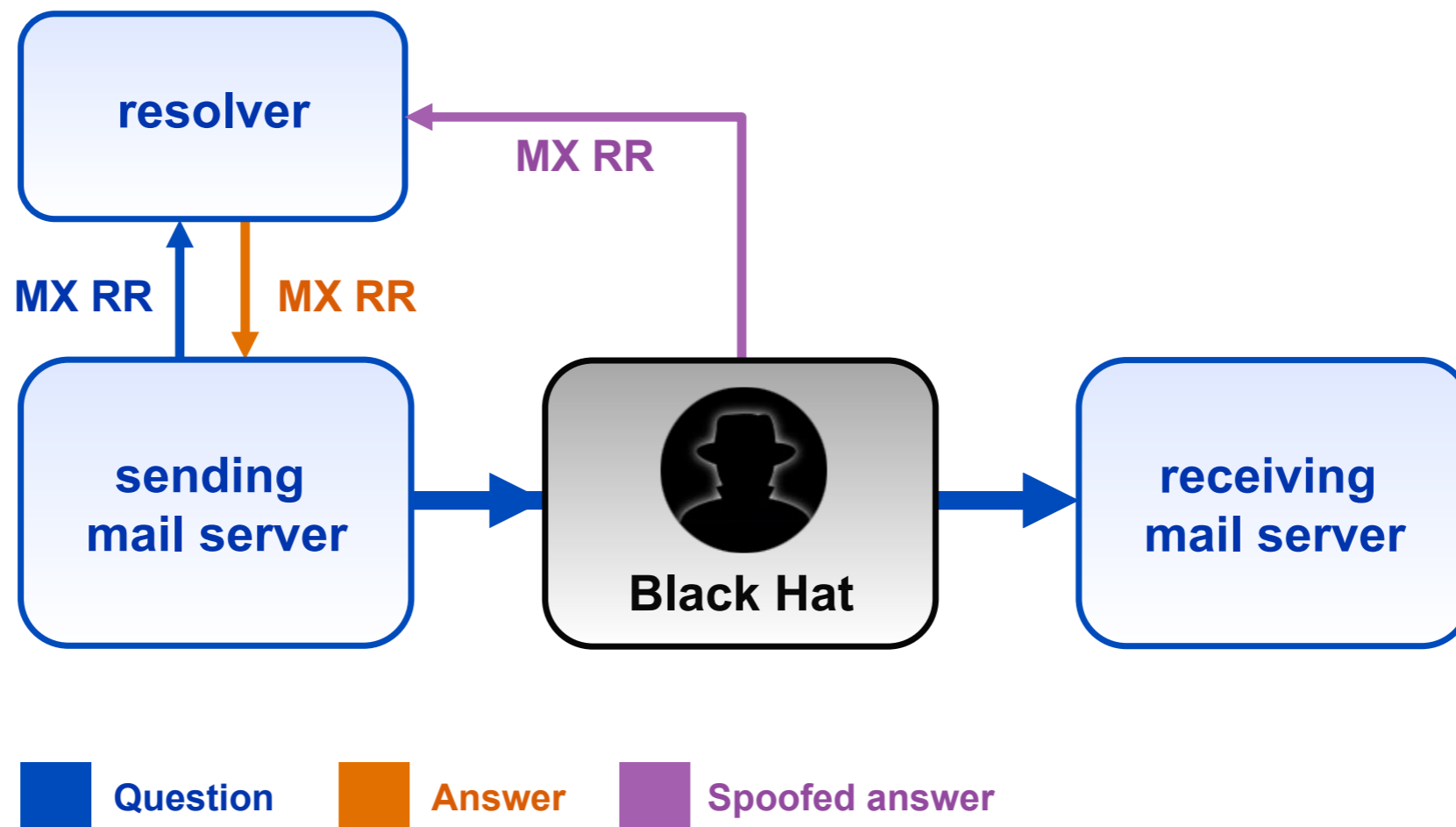
# DNS Vulnerabilities



# DNS Exploit Example



- Mail goes to the server in the MX resource record
- Path only visible in the email headers





# Introduction to Cryptography

Section 3

# Cryptography



- **A way to encrypt or hash some content**
  - Make it “secure” and/or verifiable
- **Intent is not always to hide the message**
  - For DNSSEC, goal is to verify the content
- **Different methods and keys**



# Hashes



- Turns a string into a different series of characters
- Fixed length

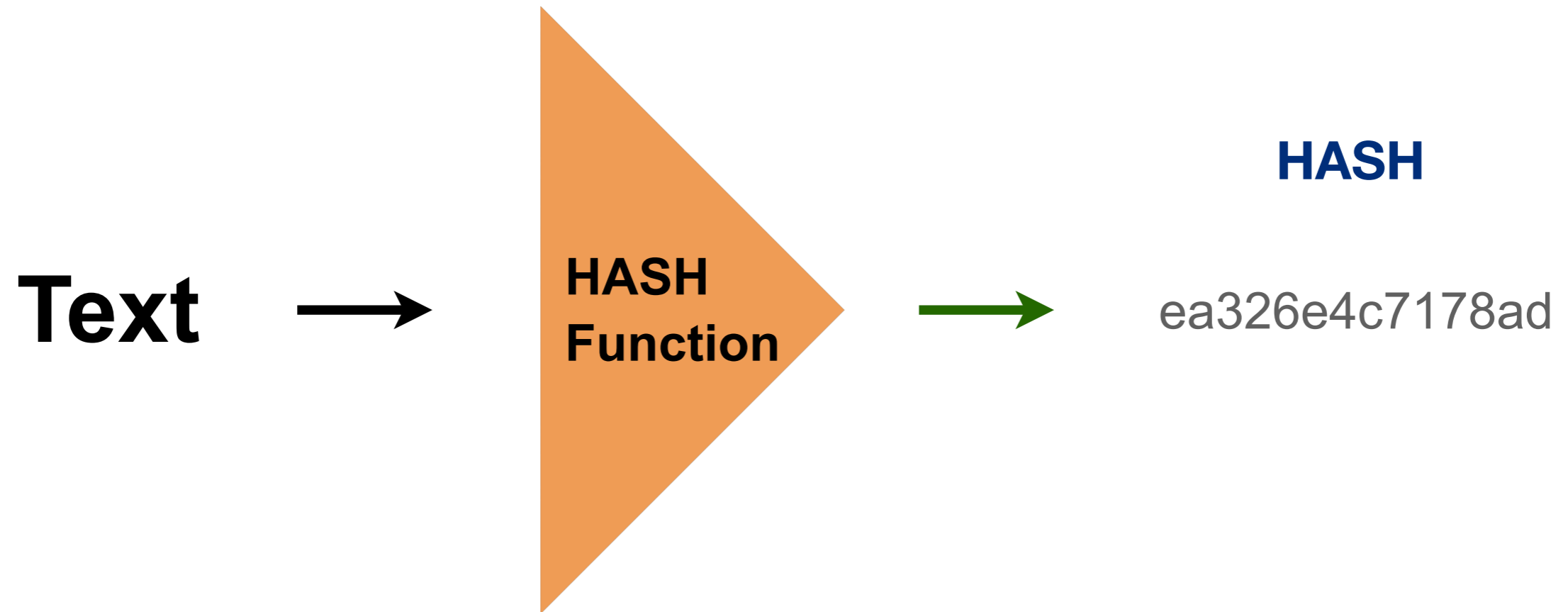
**SHA256 (“This is the DNSSEC Course”)**

a8feb4dd098d86d1ea326e4c7178ad5dcbacacabb4df421c  
0f4bbe04f28077a2

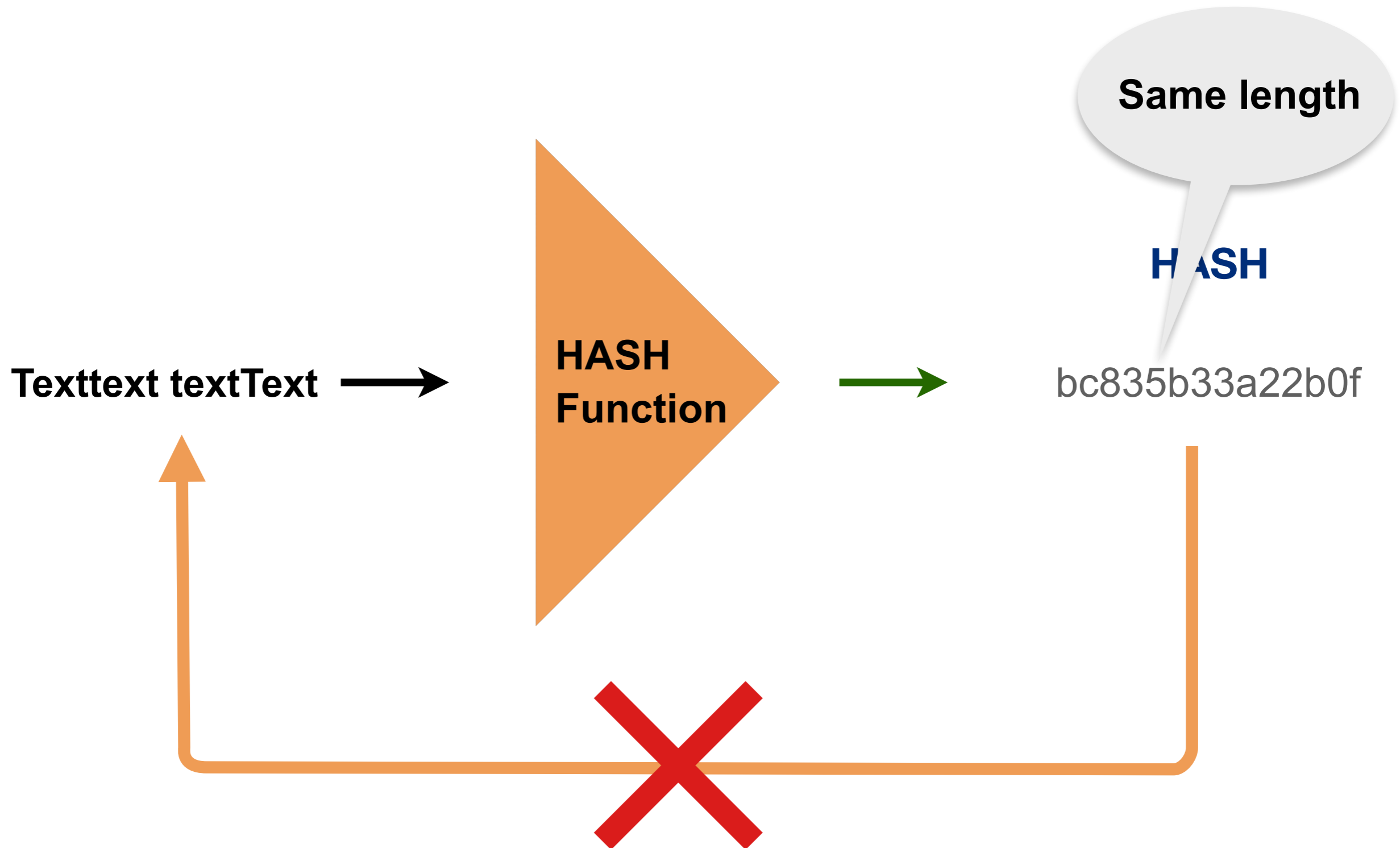
**SHA256 (“This is the DNSSEC Course for LIRs”)**

74fda40946cb6bc835b3322bc0b0a6643aca1ce38af4f88ca  
114edec859bec68

# Hashes



# Hashes



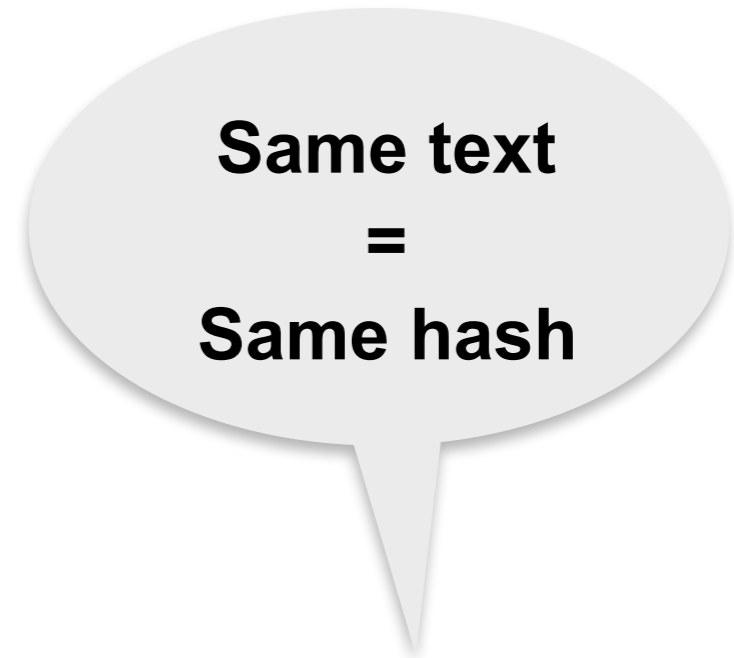
# Hashes



**Same Text**  
**Same Text**  
**Same Text**



ea326e4c7178ad  
ea326e4c7178ad  
ea326e4c7178ad



# Public Key Cryptography



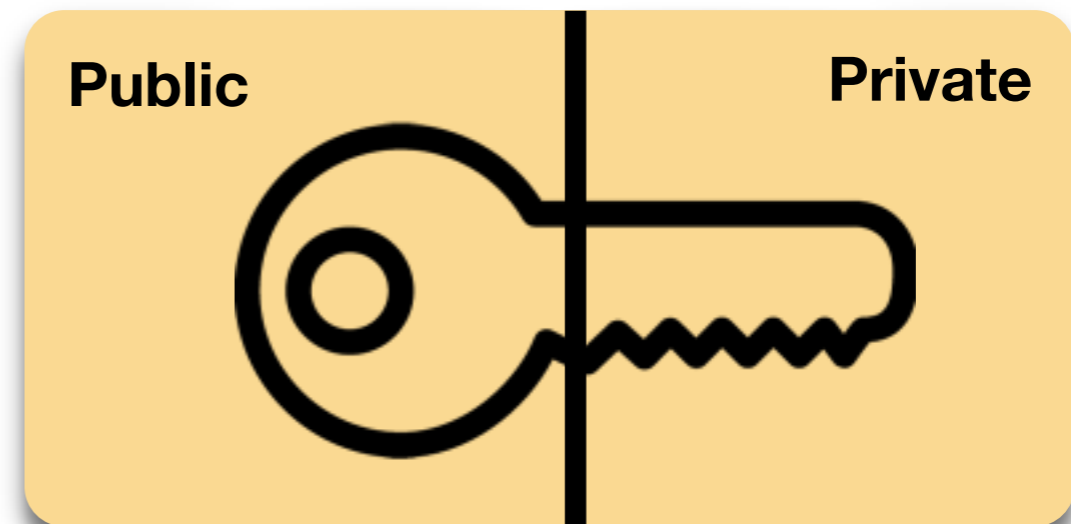
- **Most commonly used cryptographic system**
- **Can guarantee security and authentication**
  - Via encryption
  - Via signatures
  - Or both

# Encryption: Keys



- **Key pair**

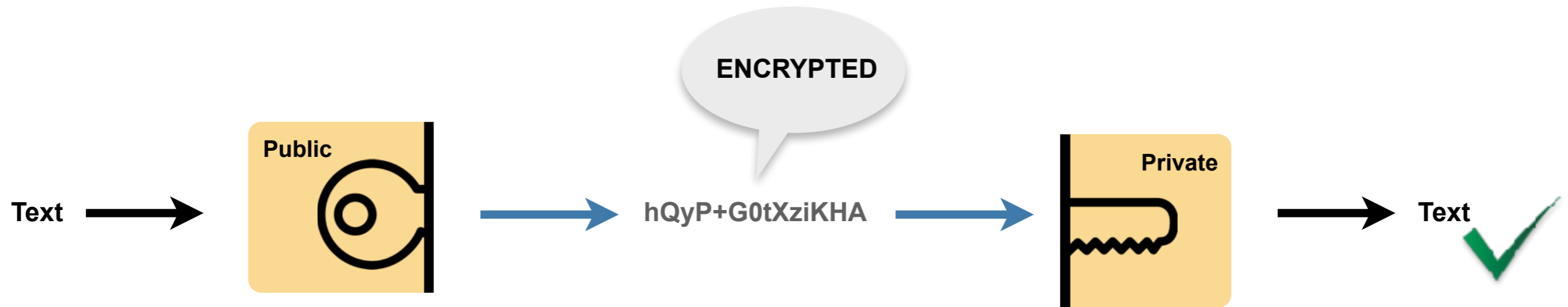
- One private
- One public



- **Content encrypted with one key, can only be decrypted with the other one**

- A public key can “open” content encrypted with the private key, and viceversa

# Encryption with Key Pair



# Digital Signatures



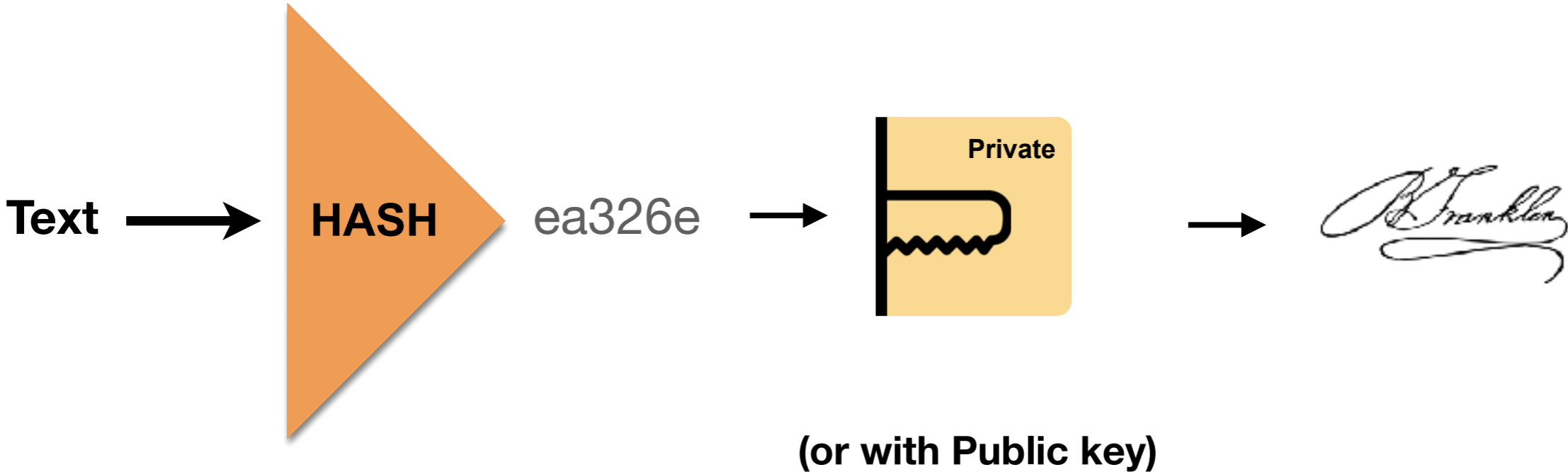
- If we combine **hashes** and public key **encryption**, we get a **digital signature**
- We generate a hash, then encrypt it with a key



# Signature



**Hashing + Encryption = Signature**



# Checking Authenticity of Signatures



- **Decrypt it,**
  - you get the hash
- **Hash original message again**
- **Compare it with the hash received**
- **If 2 hashes match, nobody tampered with the message**

# Key Rollovers



- **Keys have to be changed regularly**
  - For security reasons
- **Key rollover = scheduled changing of keys**



# Introduction to DNSSEC

## Section 4

# Basic DNS problems



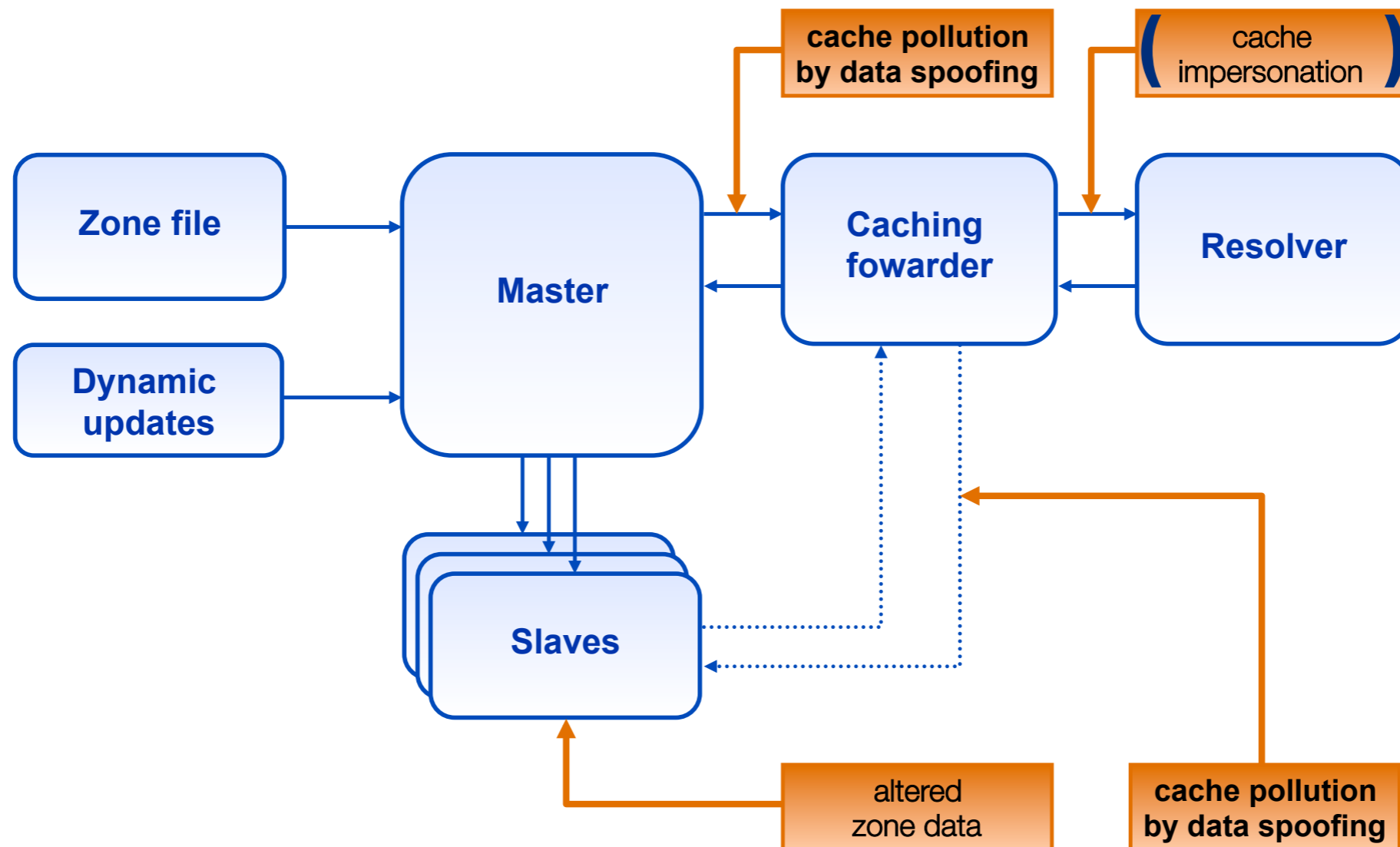
- **DNS is plain text**
- **Simple UDP, no sessions**
- **Tree structure with delegations**
  - **Each entity is responsible for a limited part of it**
- **Resolvers victims of attacks, hijacks and mistakes**
- **Trust is needed**

# DNSSEC





- **DNS Security Extensions**
- **RFC4033**
- **Adds layers on top of DNS to make it verifiable**
  - **Adds new record types**
  - **Adds PKI**
- **Chain of trust to validate data**

# DNSSEC Protected Vulnerabilities



# DNSSEC Summary



- Data authenticity and integrity by signing the Resource Records Sets with **private DNSKEY**
- You need **Public DNSKEYs** to verify the **RRSIGs**  signature
- Children sign their zones with their **private key**
  - Parent guarantees authenticity of child's key by signing the hash of it (**DS**)  Delegation Signer
- Repeat for parent ...
  - ...and grandparent
- Ideal case: one **public DNSKEY** distributed



# DNSSEC Summary



*ripe.net.*

www.ripe.net IN A 193.0.0.214

www.ripe.net IN **RRSIG** A ... 26523 ripe.net.

ripe.net IN **DNSKEY** 256 26523 ... ripe.net.

ripe.net IN **RRSIG** DNSKEY 32987 ... ripe.net.

ripe.net IN **DNSKEY** 257 32987 ... ripe.net.

*net.*

ripe.net IN **DS** 26523 8 1 ...

ripe.net IN **RRSIG** DS ... 43249 net.

net IN **DNSKEY** 256 43249 ... net.

# The Recursive Resolver's View



- So far we talked about authoritative servers
- Recursive resolver will query them for records and for authentication of records
- **DNSSEC happens between server and resolver**
  - Security status of records
  - Security status determines what client gets to see

# Security Status of Data



- **Secure**
  - Resolver can build chain of signed DNSKEY and DS RRs from trusted anchor to RRset
- **Insecure**
  - Resolver knows it has no chain of signed DNSKEY and DS RRs from any trusted starting point to RRset
- **Bogus**
  - Resolver thinks it can build a chain of trust but it is unable to do so
  - May indicate attack or configuration error or data corruption
- **Indeterminate**
  - Resolver cannot determine whether the RRset should be signed



***Update the zone file in  
BIND***

***Exercise B***



# ***Using Dig to find Information***

*Exercise C*



# **DNSSEC: New Resource Records in DNS**

Section 5

# RRs and RRSets



- **Resource Record:**

name	TTL	class	type	rdata
<code>www.ripe.net.</code>	7200	IN	A	<code>192.168.10.3</code>

- **RRset: RRs with same name, class and type:**

<code>www.ripe.net.</code>	7200	IN	A	<code>192.168.10.3</code>
<code>www.ripe.net.</code>	7200	IN	A	<code>10.0.0.3</code>
<code>www.ripe.net.</code>	7200	IN	A	<code>172.25.215.2</code>

- **RRSets are signed, not the individual RRs**

# New resource records



**RRSIG**

Signature over RRset

**DNSKEY**

Public key(s)

**DS**

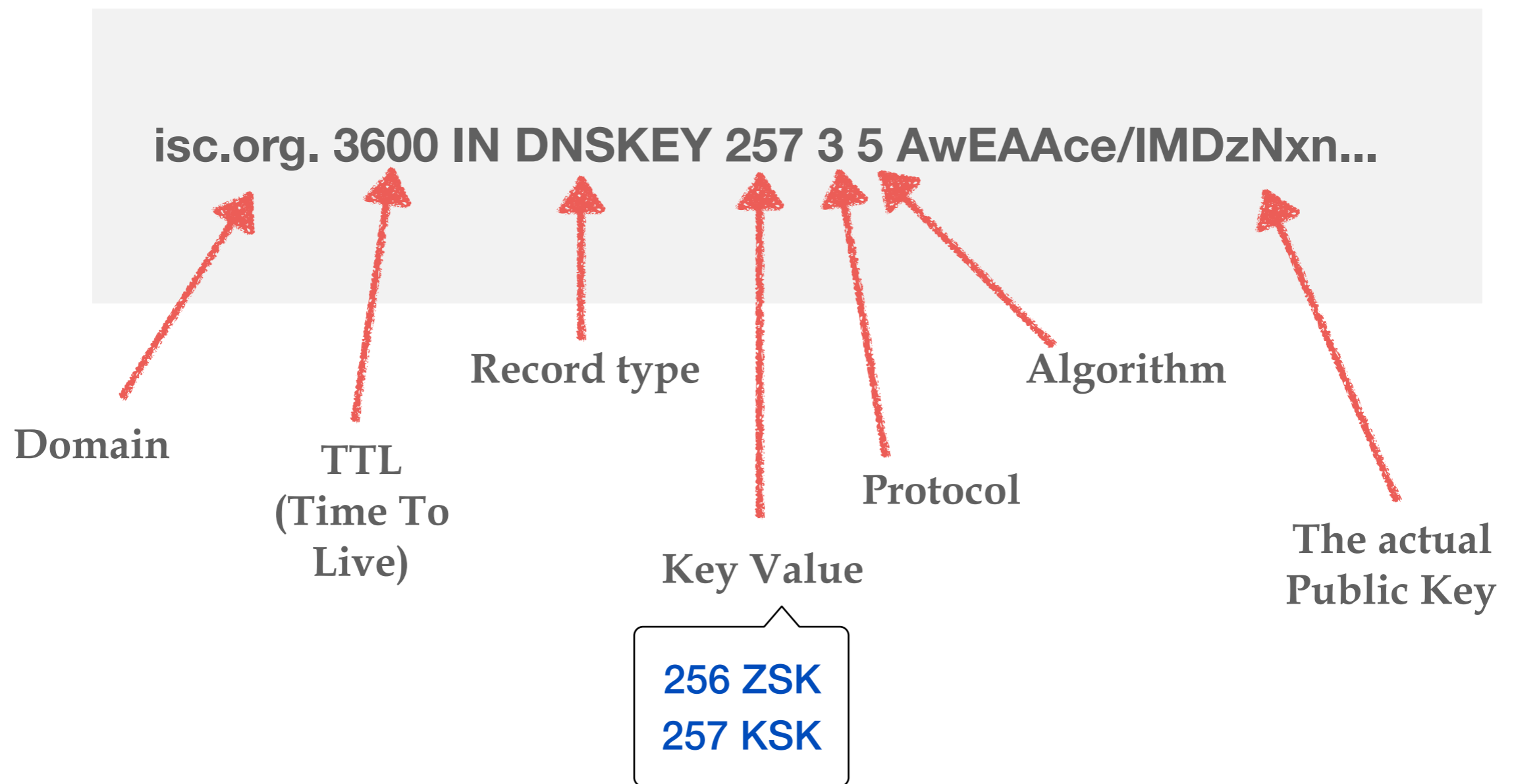
Delegation Signer (hash of DNSKEY)



# DNSKEY Record



- Contains Zone's public key(s)



# DNSKEY Record (cont.)



OWNER                      TYPE                      FLAGS                      PROTOCOL                      ALGORITHM  
MYZONE.                      600                      DNSKEY                      256                      3                      5                      (

AwEAAdevJXb4NxFnDFT0Jg9d/jRhJwzM/YTu  
PJqpvjRl14WabhabS6vioBX8Vz6XvnCzhlAx

... ) ; key id = 5538                      KEY ID

PUBLIC KEY  
(BASE64)

- FLAGS determines the usage of the key (more on this...)
- PROTOCOL is always 3 (DNSSEC)
- ALGORITHM can be:

0 – reserved

1 – RSA/MD5 (deprecated)

2 – Diffie/Hellman

3 – DSA/SHA-1 (optional)

4 – reserved

5 – RSA/SHA-1 (mandatory in validator)

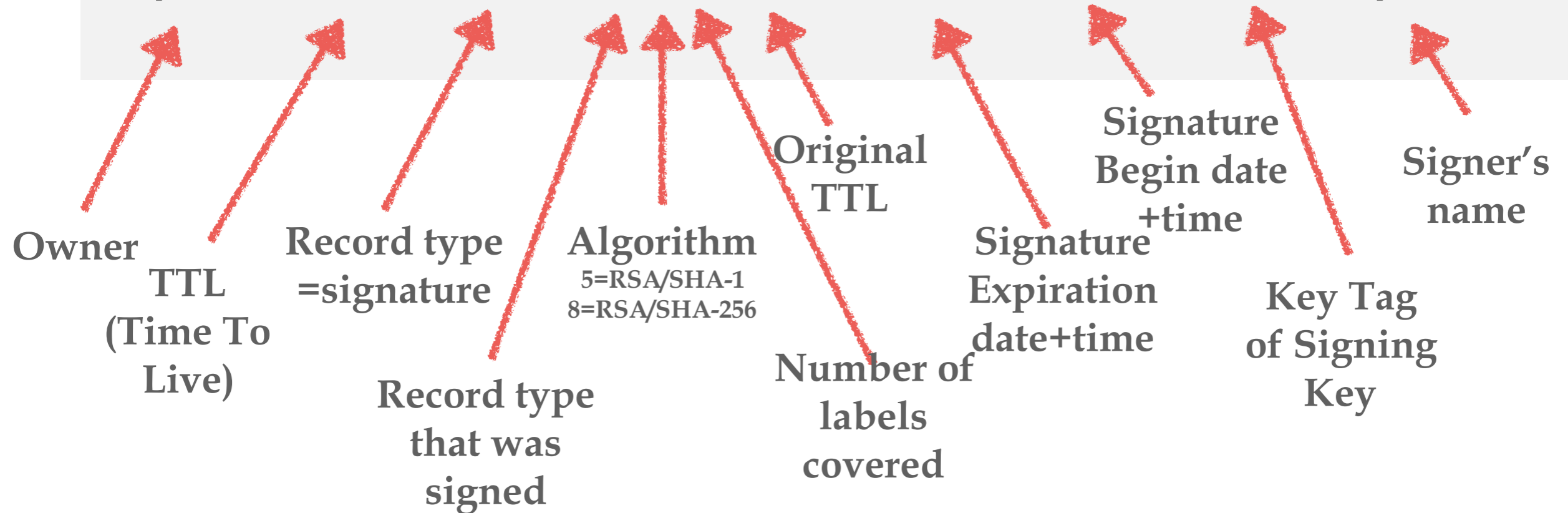
8 – RSA/SHA-256

# RRSIG



- Resource Record SIGNature
- Digital signature of a set of records

ripe.net. 3600 IN RRSIG A 5 2 3600 20140201 20140101 65306 ripe.net



# RRSIG (cont.)



RR set

```
test.myzone. 600 A 1.2.3.4  
test.myzone. 600 A 2.3.4.5
```

2      1

```
test.myzone. 600 RRSIG A 5 2 600 20090317182441 (
```

TYPE COVERED ALGO # LABELS ORIG. TTL SIG. EXPIR.

```
20090215182441 5538 myzone.
```

KEY ID      SIGNER NAME

```
rOXjsOwdIr576VRAoIBfbk0TPtxvp+1PI0XH  
p1mVwfR3u+ZuLBGxkaJkorEngXuvThV9egBC  
...  
)
```

SIG. START

RRSIG

# Delegation Signer Record



- **The child's DNSKEY is hashed**
- **The hash of the key is signed by the parent's DNSKEY**
  - and included in the parent's zone file
- **Repeat for grandchild**
  
- **Chain of trust**

# Delegation Signer (DS)

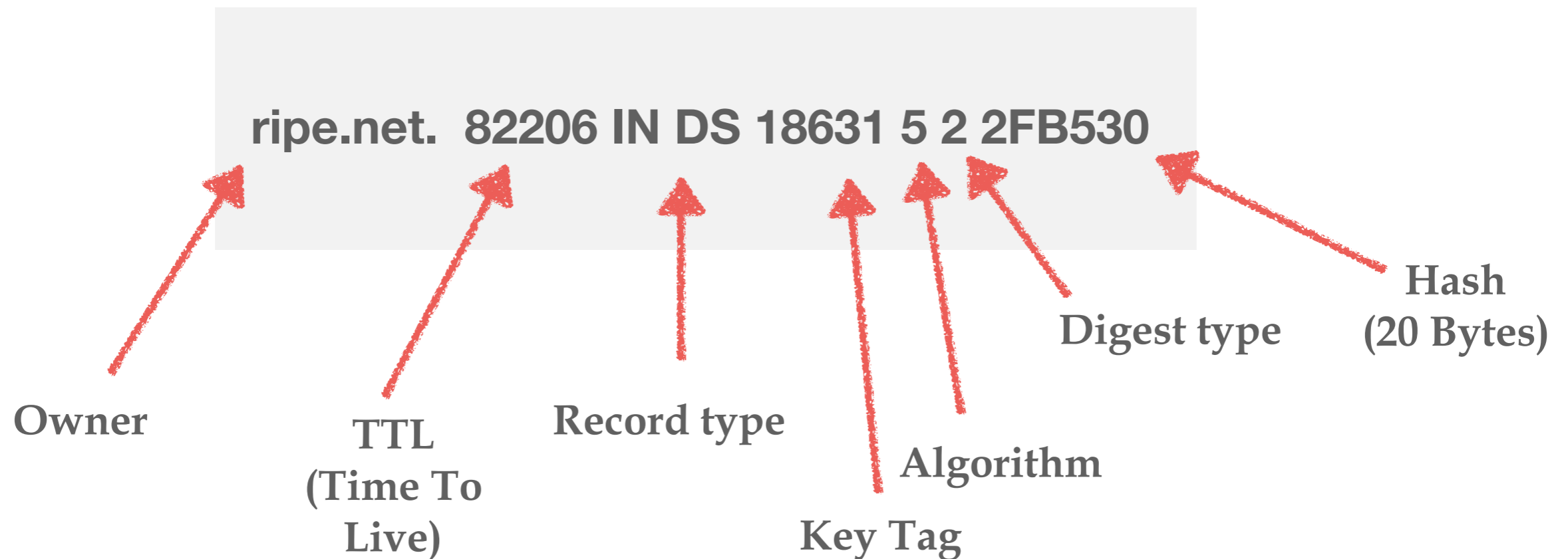


- **Delegation Signer (DS) RR shows that:**
  - child's zone is digitally signed
  - hashed key is used for the child's zone
- **Parent is authoritative for the DS of the child's zone**
  - DS should be in the parent's , not the child's zone

# DS



- Delegation Signer
- Contains hash of the (KSK) DNSKEY
- To be published in the parent zone of DNS chain



# NSEC Record



- **“Next SECure” record**
- **Authenticates non-existence of data**
- **Side Effect: allows discovery of zone contents**



# NSEC Example 1



## ZONE FILE

```
ant.ripe.net NSEC baby.ripe.net A AAAA NSEC RRSIG  
baby.ripe.net NSEC cat.ripe.net A NSEC RRSIG  
cat.ripe.net NSEC dodo.ripe.net A AAAA NSEC RRSIG  
dodo.ripe.net NSEC mouse.ripe net A NSEC RRSIG  
mouse.ripe.net NSEC ripe.net A AAAA NSEC RRSIG  
ripe.net NSEC www.ripe.net A AAAA MX NSEC RRSIG  
www.ripe.net NSEC ant.ripe.net A AAA NSEC RRSIG
```

**Q:** A for fruit.ripe.net ?

Doesn't exist! There is nothing between **dodo** and **mouse** !

**A:** **dodo.ripe.net** NSEC mouse.ripe net A NSEC RRSIG

RRSIG over NSEC

# NSEC Example 2



**ZONE FILE**

```
ant.ripe.net NSEC baby.ripe.net A AAAA NSEC RRSIG
→ baby.ripe.net NSEC cat.ripe.net A NSEC RRSIG
cat.ripe.net NSEC dodo.ripe.net A AAAA NSEC RRSIG
dodo.ripe.net NSEC mouse.ripe.net A NSEC RRSIG
mouse.ripe.net NSEC ripe.net A AAAA NSEC RRSIG
ripe.net NSEC www.ripe.net A AAAA MX NSEC RRSIG
www.ripe.net NSEC ant.ripe.net A AAA NSEC RRSIG
```

**Q:** AAAA for baby.ripe.net ?

Doesn't exist! Its not in the list in the NSEC record

**A:** baby.ripe.net NSEC cat.ripe.net A NSEC RRSIG

RRSIG over NSEC

# NSEC Record



- **Points to the next domain name in the zone**
  - also lists what are all the existing RRs for “owner”
  - NSEC record for last name “wraps around” to first name in zone
- **Used for authenticated denial-of-existence of data**
  - authenticated non-existence of TYPEs and labels

www.ripe.net. 3600 IN NSEC ant.ripe.net. A RRSIG NSEC

“owner”  
next owner in zone file  
Existing Resource Record types for [www.ripe.net](http://www.ripe.net)

# Problem: NSEC Walk



- **NSEC records allow for zone “re-construction”**
- **Causes privacy issues**
- **It’s a deployment barrier**

# Solution: NSEC3 Record



- Same as NSEC
- But hashes all names to avoid zone discovery
- Hashed names are ordered

```
DRVR6JA3E4VO5UIPOFAO5OEEVV2U4T1K.dnssec-course.net. 3600 IN  
NSEC3 1 0 10 03F92714 GJPS66MS4J1N6TIIJ4CL58TS9GQ2KRJ0 A RRSIG
```

# NSEC3 Example



## ZONE FILE

```
ant.ripe.net NSEC baby.ripe.net A AAAA NSEC RRSIG  
baby.ripe.net NSEC cat.ripe.net A NSEC RRSIG  
cat.ripe.net NSEC dodo.ripe.net A AAAA NSEC RRSIG  
dodo.ripe.net NSEC mouse.ripe.net A NSEC RRSIG  
mouse.ripe.net NSEC A AAAA NSEC RRSIG  
ripe.net NSEC www.ripe.net A AAAA MX NSEC RRSIG  
www.ripe.net NSEC ant.ripe.net A AAA NSEC RRSIG
```

## ZONE FILE

```
df67wer9x1 NSEC3 8d5g8rt69v A AAAA NSEC3 RRSIG  
8d5g8rt69v NSEC3 5tyro47f75 A NSEC3 RRSIG  
5tyro47f75 NSEC3 h3aq475y76q A AAAA NSEC3 RRSIG  
h3aq475y76q NSEC3 1z45wt6P3d A NSEC3 RRSIG  
1z45wt6P3d NSEC3 gf8r8yt64j A AAAA NSEC3 RRSIG  
gf8r8yt64j NSEC3 9t8y0gur9a A AAAA MX NSEC3 RRSIG  
9t8y0gur9a NSEC3 df67wer9x1 A AAAA NSEC3 RRSIG
```

**Q:** A for fruit.ripe.net ?

Doesn't exist! There is nothing between **h3aq475y76** and **1z45wt6P3q** !

**A:** **h3aq475y76** NSEC3 1z45wt6P3q net A NSEC3 RRSIG

RRSIG over NSEC

# New Resource Records



- **Three Public key crypto related RRs**
  - **RRSIG** Signature over RRset using private key
  - **DNSKEY** Public key, needed for verifying an RRSIG
  - **DS** Delegation Signer; 'Pointer' for building chains of authentication
- **One RR for internal consistency**
  - **NSEC** shows which name is the next one in the zone and which types exist for the name queried
  - authenticated non-existence of data



# **Delegating Signing Authority Chains of Trust**

Section 6



# What if There Was No DS ?



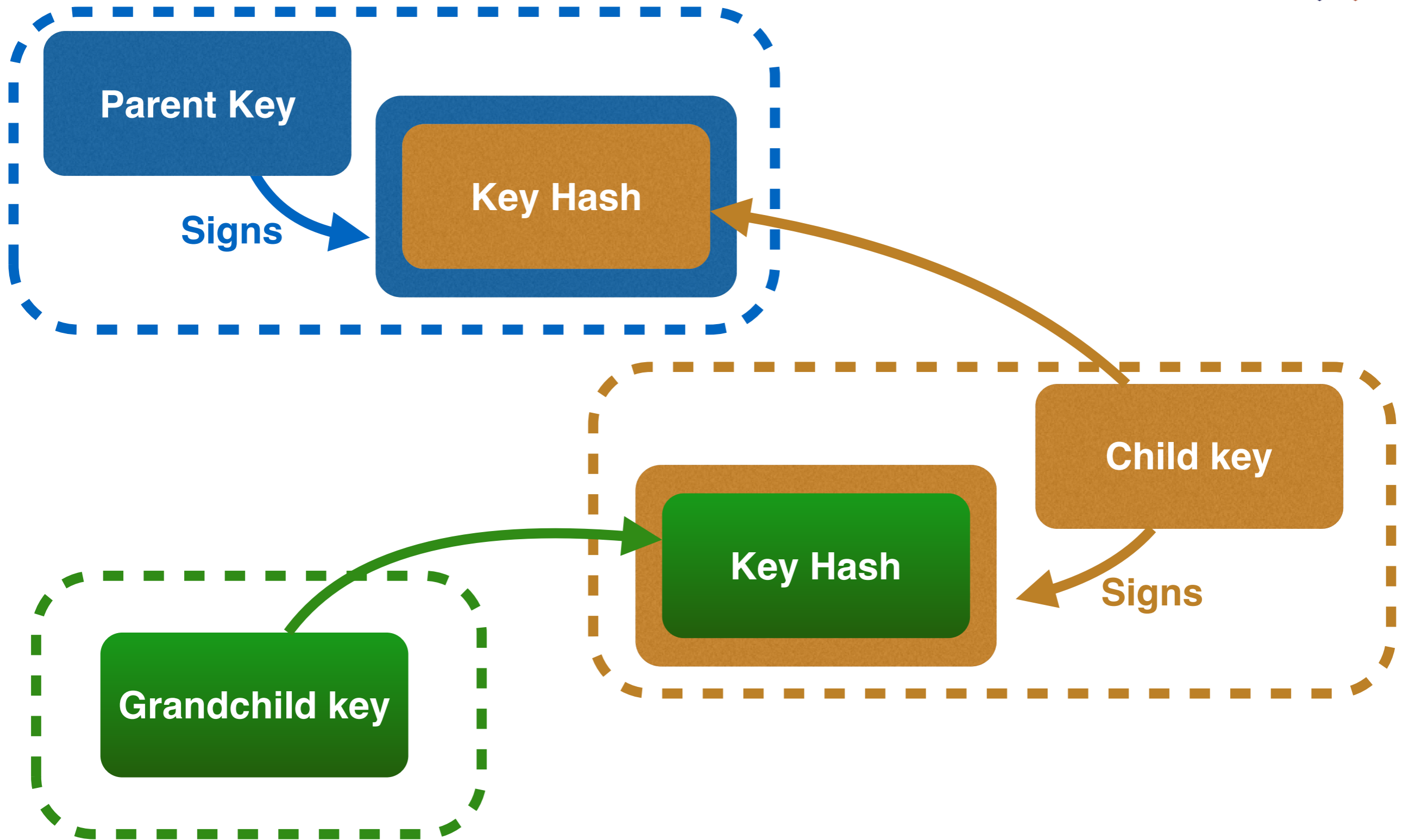
- **Without delegating signing authority (DS) the resolver would need to store millions of public keys**
  
- **But with DS only one key is needed: the root key**

# DNS and Keys



- **DNS is made of islands of trust, with delegations**
- **A parent needs to have pointers to child keys**
  - in order to sign/verify them
  - DS Records are used for this
- **You want to keep interaction between parent and children at a minimum**

# DNSSEC Made simple



# Key Problem



- **Interaction with parent administratively expensive**
  - Should only be done when needed
  - Bigger keys are better
- **Signing zones should be fast**
  - Memory restrictions
  - Space and time concerns
  - Smaller keys with short lifetimes are better

# Key Functions



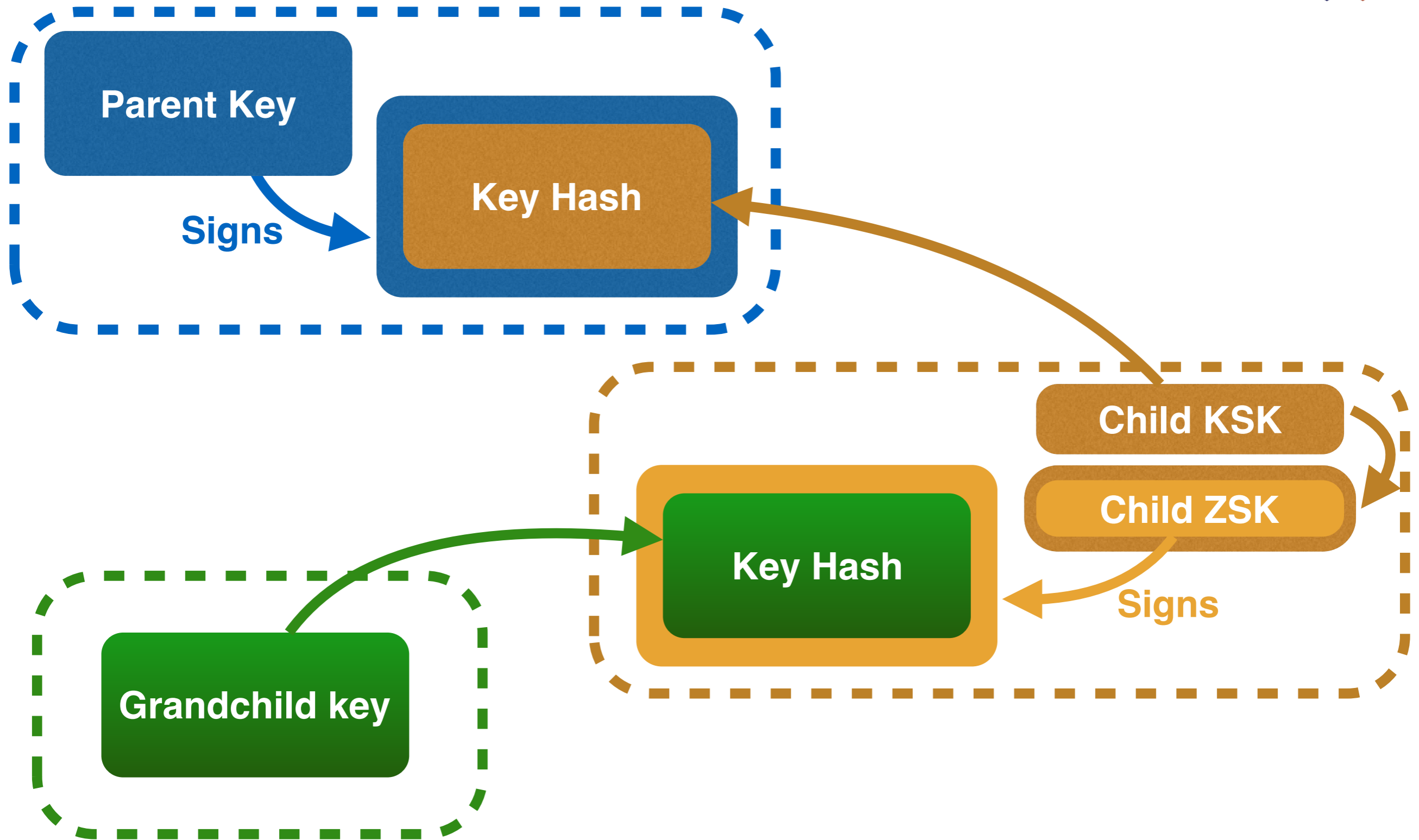
- **Large keys are more secure**
  - Can be used longer ✓
  - Large signatures => large zonefiles ✗
  - Signing and verifying computationally expensive ✗
- **Small keys are fast**
  - Small signatures ✓
  - Signing and verifying less expensive ✓
  - Short lifetime ✗

# Key Solution: More Than One Key



- **Key Signing Key (KSK)** only signs DNSKEY RRset
- **Zone Signing Key (ZSK)** signs all RRset-s in zone
  
- RRsets are signed, not RRs
- DS points to child's KSK
  - Parent's ZSK signs DS
  - Signature transfers trust from parent key to child key

# Key split - ZSK and KSK



# Zone Signing Key - ZSK



- **Used to sign a zone**
- **Can be lower strength than the KSK**
- **No need to coordinate with parent zone if you want to change it**



# Key Signing Key - KSK



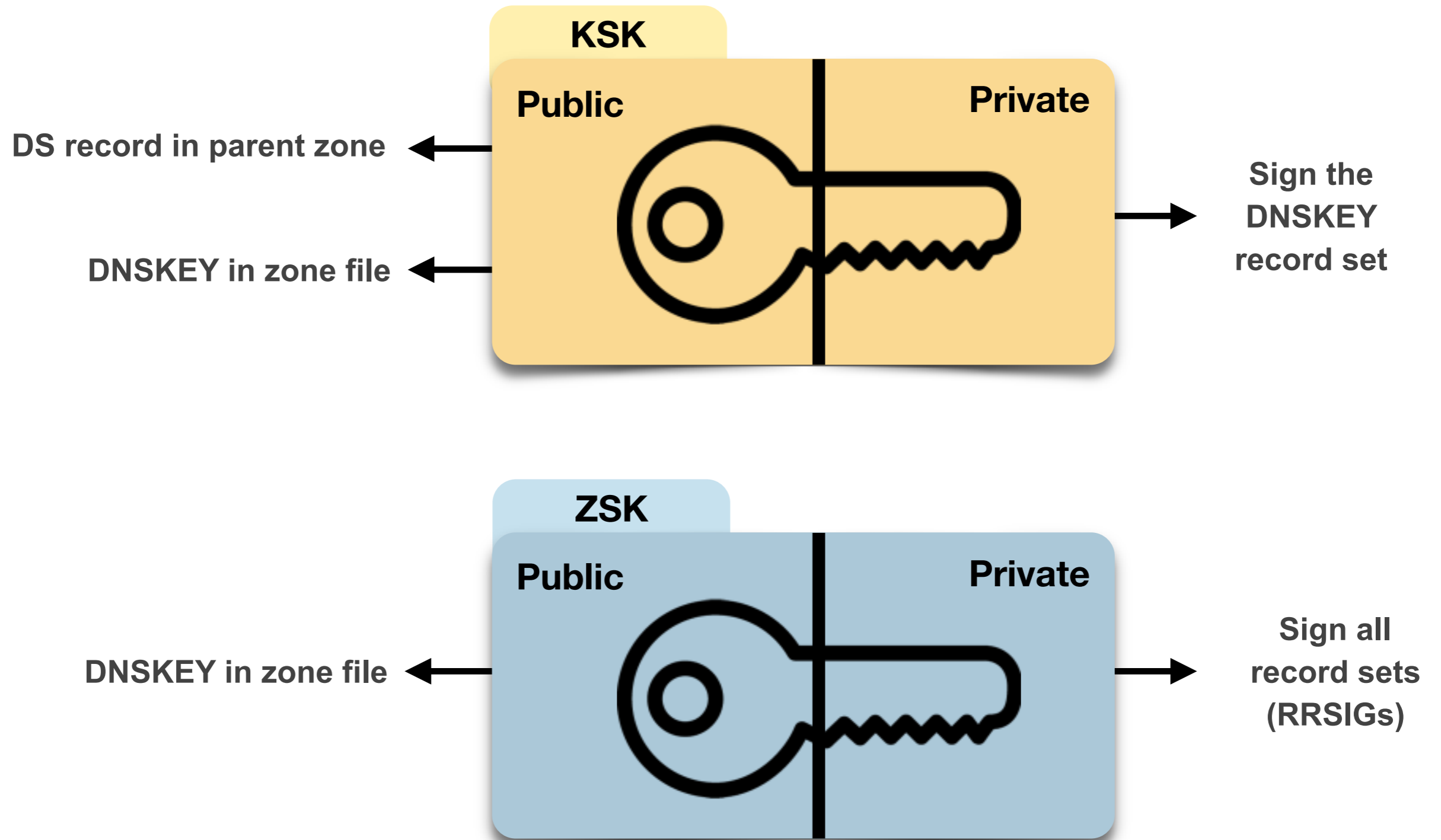
- **Only signs the Resource Record Set containing DNSKEYs for a zone**
- **Used as the trust anchor**
- **Needs to be specified in the parent zone using DS (Delegation Signature) records**

# Initial Key Exchange

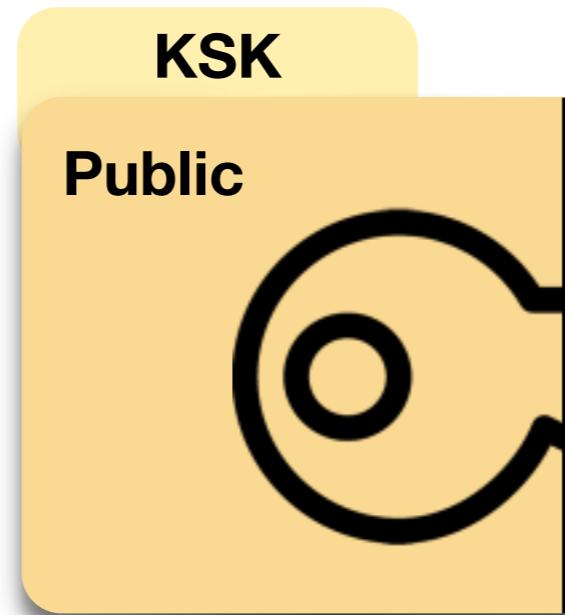


- **Child needs to:**
  - **Send key signing keyset to parent**
  
- **Parent needs to:**
  - **Check child's zone**
    - for DNSKEY & RRSIGs
  - **Verify if key can be trusted**
  - **Generate DS RR**

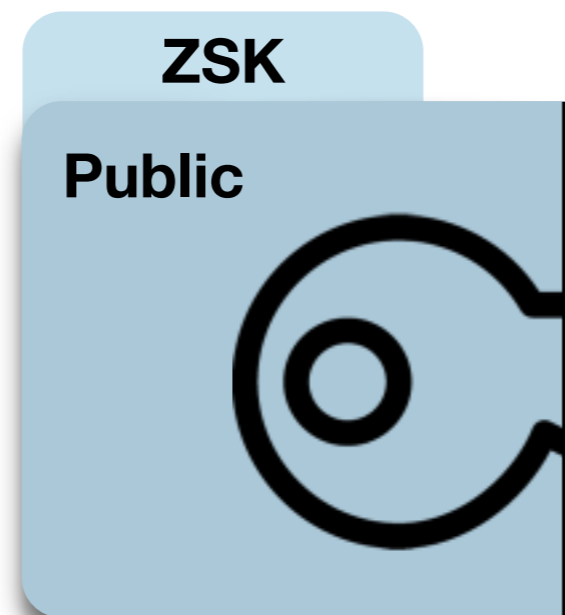
# Keys



# Keys



Used to decrypt RRSIG  
and verify signatures





# PARENT

DNSKEY (KSK)

DNSKEY (ZSK)

DS

RRSIG DS

← hash of child's (public) KSK

← signed by Parent's (private) ZSK

# CHILD

MX  
MX  
MX

Record Set

RRSIG MX

← signed by (private) ZSK

A  
A  
A

Record Set

RRSIG A

← signed by (private) ZSK

DNSKEY (KSK)

← (public) KSK

DNSKEY (ZSK)

← (public) ZSK

RRSIG DNSKEY

← signed by (private) ZSK

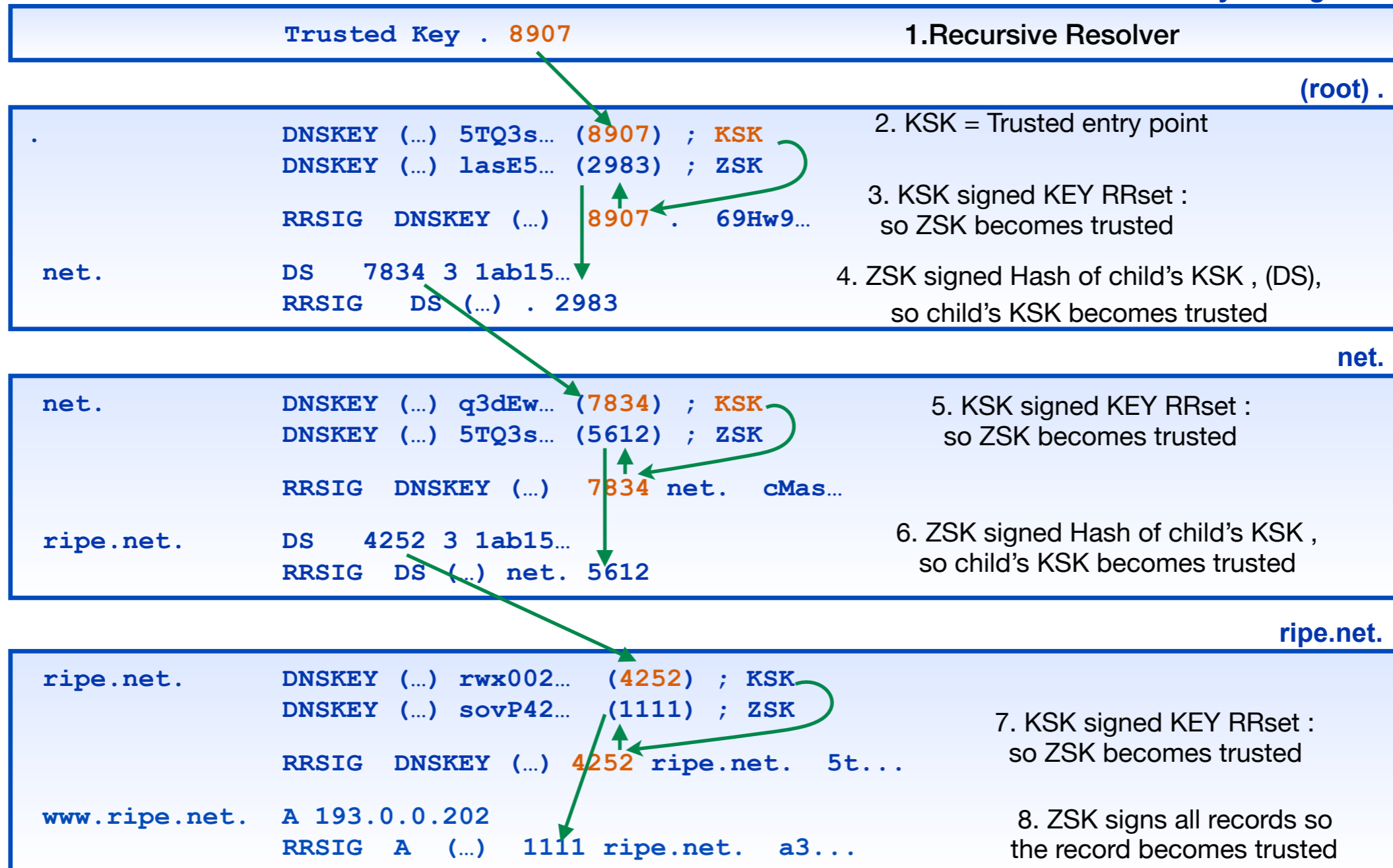
RRSIG DNSKEY

← signed by (private) KSK

# Walking the Chain of Trust



Locally Configured





# **Setting Up a Secure Zone Step by Step**

Section 7

# DNSSEC Step-by-Step



**1. Generate the key pair**

**2. Sign and publish the zone(s)**

DNSSEC **NOT** active



DNSSEC active

**3. Create DS Record on parent**



# Step 1 : Generate the Key Pair



```
dnssec-keygen -a alg -b bits -f KSK -n type [options] name
```

- **algorithm:** RSA-SHA1
- **Bitsize:** depends on key function & paranoia level
- **type:** zone
- **name:** zone you want to sign
  - **key type:** either null or KSK
- **'-r /dev/urandom'** might be needed

# 1. Creating the Key Pair



```
$ dnssec-keygen -a RSASHA1 -b 1024 -n zone example.net.  
$ kexample.net.+005+20704
```

- **2 files are created:**
  - **Kexample.net.+005+20704.key**
    - contains the public key
    - should go into the zone file
  - **Kexample.net.+005+20704.private**
    - contains the private key

# 1. Generate Keys



- in `/etc/bind/keys/example.com`:

Directory where keys are stored

```
# mkdir -p /etc/bind/keys/example.com
# cd /etc/bind/keys/example.com
# dnssec-keygen -a RSASHA256 -b 1024 example.com
Generating key pair.....+++++
Kexample.com.+008+17694
# dnssec-keygen -a RSASHA256 -b 2048 -f KSK example.com
Generating key pair.....+++
Kexample.com.+008+06817
```

ZSK key

KSK key

Algorithm

Number of bits

# 1. Generate Keys (cont.)



- 4 files in `/etc/bind/keys/example.com`:

- `Kexample.com.+008+06817.key`
- `Kexample.com.+008+06817.private`
- `Kexample.com.+008+17694.key`
- `Kexample.com.+008+17694.private`

- looking inside the key file you can tell if ZSK or KSK

# 1. Generate Keys



```
# cat Kexample.com.+008+06817.key
; This is a key-signing key keyid 6817, for example.com.
; Created: 20141120094612 (Thu Nov 20 17:46:12 2014)
; Publish: 20141120094612 (Thu Nov 20 17:46:12 2014)
; Activate: 20141120094612 (Thu Nov 20 17:46:12 2014)
example.com. IN DNSKEY 257 3 8 AwEAAcWDps...lM3NRn/G/R
# cat Kexample.com.+008+17694.key
; This is a zone-signing key keyid 17694, for example.com.
; Created: 20141120094536 (Thu Nov 20 17:45:36 2014)
; Publish: 20141120094536 (Thu Nov 20 17:45:36 2014)
; Activate: 20141120094536 (Thu Nov 20 17:45:36 2014)
example.com. IN DNSKEY 256 3 8 AwEAAcjGaU...zUU55If5
```



## 2. Signing by Reconfiguring BIND



- Add extra lines to 'named.conf' file
  - /etc/bind/named.conf

```
options {  
    directory "/etc/bind";  
    recursion no;  
    minimal-responses yes;  
};
```

```
zone "example.com" IN {  
    type master;  
    file "db/example.com.db";  
    key-directory "keys/example.com";  
    inline-signing yes;  
    auto-dnssec maintain;  
};
```

created a subfolder  
'example.com' for that  
zone's keys

where named should look  
for the public and private  
DNSSEC key files

BIND keeps unsigned zone and creates signed zone

next slide

## 2. Reconfigure BIND (cont)



- **auto-dnssec ...**
  - **off** default. Key management manually
  - **allow** allows uploading keys and resigning the zone  
when user runs `rndc -sign [zone-name]`
  - **maintain** same as “allow” +automatically adjusts the keys on schedule ( key’s timing metadata)

## 2. Reload named.conf



```
# rndc reload  
server reload successful
```



# 2 : What Does Signing the Zone Do?



- Sort the Zone
- Insert:
  - **NSEC** records
  - **RRSIG** records (signature over each RRset)
  - **DS** records (optional)
- Generate 'key-set' and 'ds-set' files
- Remember: Test! (use recursive resolver)

# Securing the Zone



- **Publish signed zone**
- **Signed zone is regular zone file format**
  - **With extra resource records**
- **Make sure all your DNS servers are DNSSEC capable!**

# Step 3 : Setting up DNSSEC



- **Distribute your public key (DNSKEY)**
  - To parent zone (key-set or ds-set can be used)
  - To everyone who wants/needs you as SEP
- **Make sure to inform everyone of key rollovers!**

```
$ dnssec-dsfromkey kexample.net.+005+20704
```

# Verifying with the Recursive Resolver



- To verify the content of a zone:
  - Get the public (key signing) key and check that this key belongs to the zone owner
- Configure the keys you trust as secure entry points in `named.conf`

```
trusted-keys {  
    "example.net." 256 3 1 "AQ...QQ==";  
};
```



# ***Configure DNSSEC for the Domain***

*Exercise D*



# Flags and Scenarios

Section 8

# Flags Intro



- **Flags modify or fine-tune DNS queries.**
- **They have effect on the communications between the “recursive resolver” and the “authoritative server”**
- **They are used in both the query and the response**

# Types of flags



- **Command line flags**
- **Internal flags** in question or answer section of dig response
  - **DNS flags vs DNSSEC flags**



# Flags Intro 2



- **DNSSEC happens between “recursive resolver“ and “authoritative server”**
- **DNS queried by**
  - client application or
  - diagnostics tool (dig) running on client
- **Diagnostic tool tells recursive resolver which flags to set**
- **Diagnostic tool shows flags received by “recursive resolver” in response from “authoritative server”**

# Where Do You See These Flags?



- dig (query) response:

```
$ dig ns .
```

question. what you typed

```
; <<>> DiG 9.9.4 <<>> ns .
```

```
:: global options: +cmd
```

```
:: Got answer:
```

```
:: ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53999
```

status NOERROR/SERVFAIL

```
:: flags: qr rd ra ad; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1
```

```
:: OPT P flags CTION:
```

```
; EDNS: version: 0, flags:; udp: 4096
```

```
:: QUESTION SECTION:
```

```
;                IN      NS
```

question. translated to its own "language"

```
:: ANSWER SECTION:
```

```
.      183478    IN      NS      f.root-servers.net.  
.      183478    IN      NS      m.root-servers.net.  
.      183478    IN      NS      c.root-servers.net.
```

answer

```
:: Query time: 0 msec
```

```
:: SERVER: 193.0.19.101#53(193.0.19.101)
```

```
:: WHEN: Mon Oct 31 11:32:56 CET 2016
```

```
:: MSG SIZE rcvd: 239
```

# dig



- **DNS diagnostic**
- **Can simulate DNS queries and more**
- **flags:**
  - +dnssec
  - +cdflag
  - +multiline

# dig Command Line Flags



## +dnssec flag

- Requests DNSSEC records be sent by setting the DNSSEC OK bit (DO flag) the query.

# dig Command Line Flags








## +cdflag

- **Set the CD (checking disabled) bit/flag in the query. This requests the recursive server to not perform DNSSEC validation of responses.**






# Internal Flags



## DNS:

- qr query response 
- rd recursion desired  
- ra recursion available 
- aa authoritative server 

## DNSSEC:

- ad authenticated data 
- cd checking disabled  
- do show me DNSSEC data  

# DNS Flags Explained: qr



- **Query response: This is a response to a query**
  - only used in responses

# DNS Flags Explained: rd



- **Recursion desired: “If you, the recursive resolver” don’t know the answer, then go look it up, if necessary in several steps, from the authoritative servers**
  - In query: an instruction
  - In response; info. I was asked to do recursion in the query



# DNS Flags Explained: ra



- **Recursion available: response to “rd” flag**
  - Only in response: Info. “Recursion as instructed in the query”

# DNS Flags Explained: aa



- **“Authoritative answer” flag**
  - The “recursive resolver” didn’t have to do recursively query other authoritative servers, because by chance it was itself authoritative for what was being queried.
  - Only in the response

# DNSSEC Flags Explained: ad



- **“Authenticated Data” flag**
  - **“ad” flag tells us that the answer received has passed the validation process We can have confidence in the authenticity and integrity of the answer**
- **only in response**

# DNSSEC Flags Explained: cd



- **“Checking Disable” flag**
  - disables DNSSEC validation in dig
  - appears in both query and response

# DNSSEC Flags Explained: do



- “DNSSEC OK” flag
- visualise the RRSIG records with the query
  - appears in both the query and the answer

# DNSSEC Statuses



- **NOERROR**
- **SERVFAIL**

# DNS and DNSSEC Statuses



DNS	DNSSEC
NOERROR	NOERROR
NXDOMAIN	NSEC/NSEC3
SERVFAIL	SERVFAIL

# dig



- `dig www.isc.org`
  - `dig www.isc.org A`
  - `dig @192.168.1.7 www.isc.org A`
- ← same answer



which recursive  
server to use

- **DNSSEC validated answers?**
  - depends whether Server and Recursive Resolver configured for DNSSEC



# dig Example 1



If DNSSEC is disabled on resolver:

No DNSSEC validation  
on recursive resolver

DNSSEC enabled on  
server

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
```



```
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline  
; (1 server found)  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416  
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```



recursion was  
desired

recursion  
was available

# dig Example 1



If DNSSEC is disabled on resolver: (whole answer)

No DNSSEC validation  
on recursive resolver

DNSSEC enabled on  
server

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.      IN  A

;; ANSWER SECTION:
www.isc.org.      60  IN  A 149.20.64.69
```

# dig Example 2



## If DNSSEC enabled on resolver:

DNSSEC validation on recursive resolver

DNSSEC enabled on server

show DNSSEC data (RRSIG)

sets the DO flag

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32472
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
```

Authenticated  
(DNSSEC validated)  
data

“I am showing  
you the DNSSEC  
records (RRSIG)”

How would the flags and  
answers be different without  
the +dnsec flag?

# dig Example 2



If DNSSEC enabled on resolver: (whole answer)

```
$ dig @192.168.1.7 www.isc.org. A +dnssec +multiline
; <<>> DiG 9.10.0-P2 <<>> @192.168.1.7 www.isc.org. A +dnssec +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NCERROR, id: 32472
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.isc.org.      IN A

;; ANSWER SECTION:
www.isc.org.      4 IN A 149.20.64.69
www.isc.org.      4 IN RRSIG A 5 3 60 (
20141029233238 20140929233238 4521 isc.org.
DX5BaGVd4KzU2AIH911Kar/UmdmkARyPhJVLr0oyPZaq
5zoobGqFI4efvzL0mcpncuUg3BSU5Q48WdBu92xinMdb
E75zl+adgEBOsFgFQR/zqM3myt/8SngWm4+TQ3XFh9eN
iqExHZZuZ268Ntlxqgf9OmKRRv8X8YigaPShuyU= )

;; Query time: 3 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 03 16:40:04 CST 2014
;; MSG SIZE  rcvd: 223
```

shows RRSIG  
record

# dig Examples 3 + 4



- Let's use dig to examine a domain with “broken”  
DNSSEC
- 3: Validation NOT enabled on recursive server
- 4: Validation ENABLED on recursive server

# dig Example 3



No DNSSEC validation  
on recursive server

DNSSEC broken  
on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28878
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.    IN  A

;; ANSWER SECTION:
www.dnssec-failed.org.    7200  IN  A  68.87.109.242
www.dnssec-failed.org.    7200  IN  A  69.252.193.191

;; Query time: 955 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:42:50 CST 2014
;; MSG SIZE rcvd: 82
```



# dig Example 4



DNSSEC validation on recursive server

DNSSEC broken on server

```
$ dig @192.168.1.7 www.dnssec-failed.org. A

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.dnssec-failed.org. A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 46592
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.dnssec-failed.org.      IN  A

;; Query time: 2435 msec
;; SERVER: 192.168.1.7#53(192.168.1.7)
;; WHEN: Fri Oct 17 07:44:56 CST 2014
;; MSG SIZE rcvd: 50
```

# dig Example 5 (Diagnostics)



- All DNSSEC validation failures -> “SERVFAIL”
  - how do I know failure because of validation?
  - +cd flag!
  - “checking disabled”



# dig Example 5 (Diagnostics)



DNSSEC validation  
on recursive server

DNSSEC broken  
on server

CHECKING  
DISABLED

```
$ dig @192.168.1.7 www.isc.org. A +cd

; <<>> DiG 9.10.1 <<>> @192.168.1.7 www.isc.org. A +cd
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR id: 33590
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isc.org.      IN  A

;; ANSWER SECTION:
www.isc.org.      30  IN  A 149.20.64.69
```

# +dnssec



dig +dnssec

sets DO flag  
shows DNSSEC records

**DNSSEC  
Fails?**

Y

N

**+cd set?**

**+cd set?**

Y ↓

↓ N

Y ↓

↓ N

**no  
DNSSec  
validation  
+RRSIG  
shown**

**'SERVFAIL'**

**no  
DNSSec  
validation  
+RRSIG  
shown**

**DNSSec  
validation  
+RRSIG  
shown**

like a  
normal  
DNS  
query



# Key Rollovers

## Section 9

# Keys need to be changed



- **Keys become old quickly**
  - New exploits are discovered every day
  - Brute force becomes less and less expensive
- **Your keys could be stolen or compromised**
- **You need to have a plan**

# Key rollover methods



- **Pre-publish**
- **Double signature**
  
- **Both for ZSK and KSK**
  - Rolling a KSK means changing parent DS records
  
- **Rollover times depend on TTL and method**

# Pre-publish method



*new* RRSIG  
*new* DNSKEY  
  
*old* RRSIG  
*old* DNSKEY



# Pre-publishing Method



- **A new DNSKEY record is introduced with new key**
  - Not used for signing, yet
- **After TTL expires, new RRSIGs are created with new DNSKEY**
  - Old DNSKEY remains published
- **After TTL expires again, old DNSKEY is removed**

# Double Signature Method



*new* RRSIG  
*new* DNSKEY  
  
*old* RRSIG  
*old* DNSKEY





# Double signature Method



- **A new DNSKEY is introduced, and immediately used to sign the records**
- **We have two RRSIGs for every record, with signatures from both DNSKEYs**
- **After TTL expires, old DNSKEY is removed, and records are again signed only once**

# So do I Have to Remember to Rollover?

- **No, you can automate it**
  - in the configuration
  - including the schedule
  
- **Just provide ahead of time enough DNSSEC keys for the next few rollovers**

# Keys in practice



- **A key has 5 important dates:**
  - Publication
  - Activation
  - Inactivation
  - Revocation
  - Deletion
- **BIND with *auto-dnssec* will automatically manage them for you**

# Recommendations



- **Use pre-publishing for ZSK**
  - Especially for large zones
  
- **Use double signature for KSK**
  - KSK double-signs the DNSKEY, not the zone
  
- **For KSK rollovers, update DS records**

# CDS/CDNSKEY



- **Same exact records as DS and DNSKEY**
- **Published at child to influence parent**
- **Not used for initial setup**
- **Require chain of trust to be already established**

# Recommendations



- **Change your keys regularly**
- **Set up automatic rotation every 6 months**
- **You can already prepare for 2-4 years worth of keys**



**DANE**

**Distributing secure data**

Section 10

# Authenticating



- **With DNSSEC, we can assume the content of the DNS response is authentic**
  
- **We can use it to distribute authenticated data**
  - Certificate hashes
  - SSH key Fingerprints



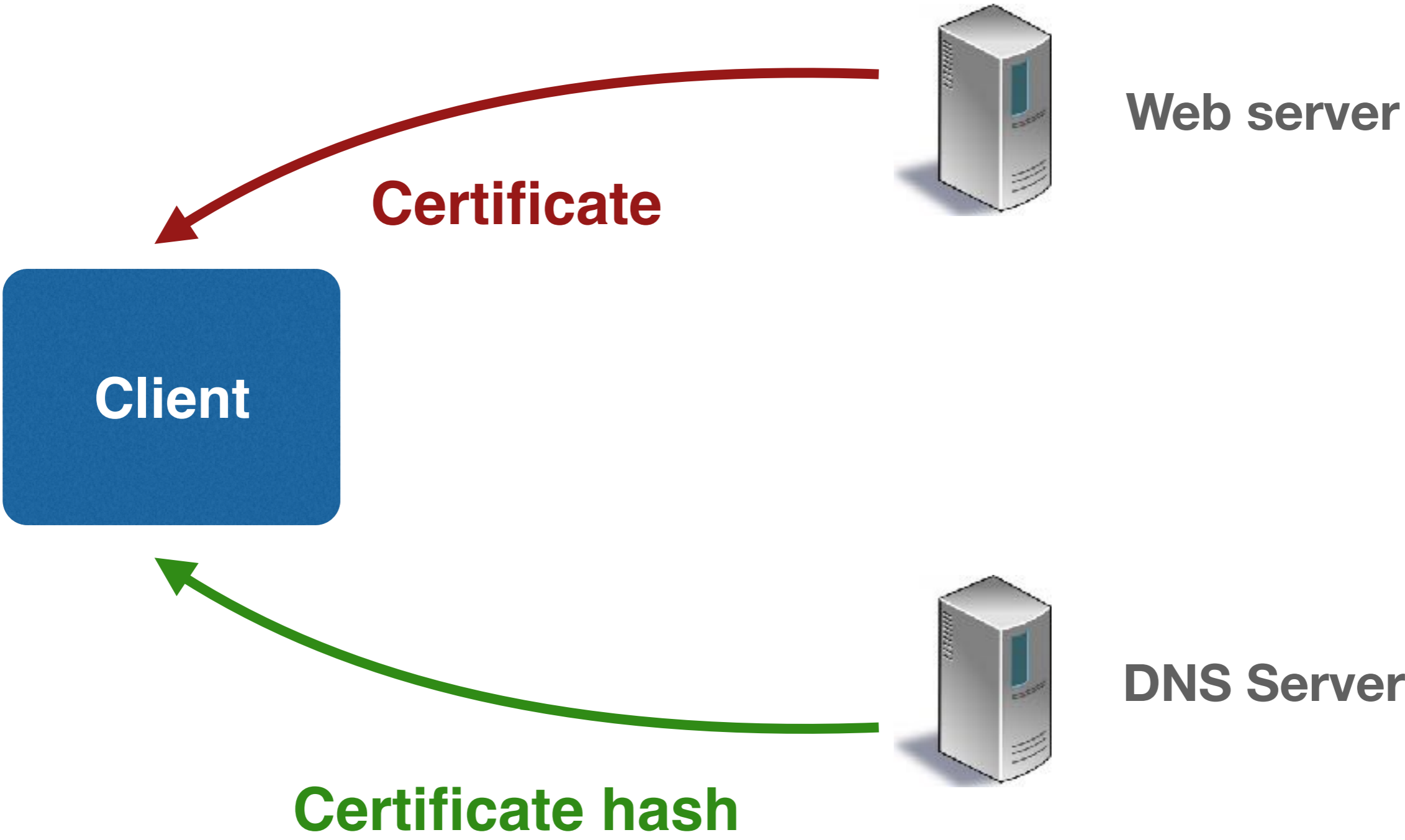
# SSH Certificate Clicking Hell



The image displays a collage of overlapping security warning windows from various browsers and operating systems, illustrating the "SSH Certificate Clicking Hell" concept.

- Security Alert (Windows):** A grey window with a red 'X' icon. Text includes: "change with this site cannot be viewed or However, there is a problem with the site's", "ertificate was issued by a company you have trust View the certificate to determine whether".
- Certificate Error (Windows):** A blue window with a red 'X' icon. Text includes: "The server has presented an invalid certificate: (unable to get local issuer certificate)", "Do you wish to accept it and connect anyway?", "Add This Certificate to Trusted List", "Accept", "Reject".
- SSL Certificate Error (Firefox):** A grey window with a blue circuit icon. Text includes: "SSL Certificate Error", "The certificate for this server is invalid. You might be connecting to a server that is pretending to be 'lancelot' which could put your confidential information at risk.".
- This Connection is Untrusted (Firefox):** A yellow window with a warning icon. Text includes: "This Connection is Untrusted", "You have asked Firefox to connect securely to [accountingonline.gov.uk] but we can't confirm that your connection is secure.", "Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, if a site's identity can't be verified.", "What Should I Do?", "If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.", "Get me out of here!", "Technical Details", "I Understand the Risks".
- Browser Address Bar (Chrome/Firefox):** A yellow bar showing a URL: "https://www.accountingonline.gov.uk/".
- Technical Details (Firefox):** A white window with a red 'X' icon. Text includes: "There is a problem with this website's security certificate.", "The security certificate presented by this website was issued for a different website's address.", "Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.", "We recommend that you close this webpage and do not continue to this website.", "Click here to close this webpage.", "Continue to this website (not recommended).", "More information".

# DANE Mechanism: Alternate certificate validation

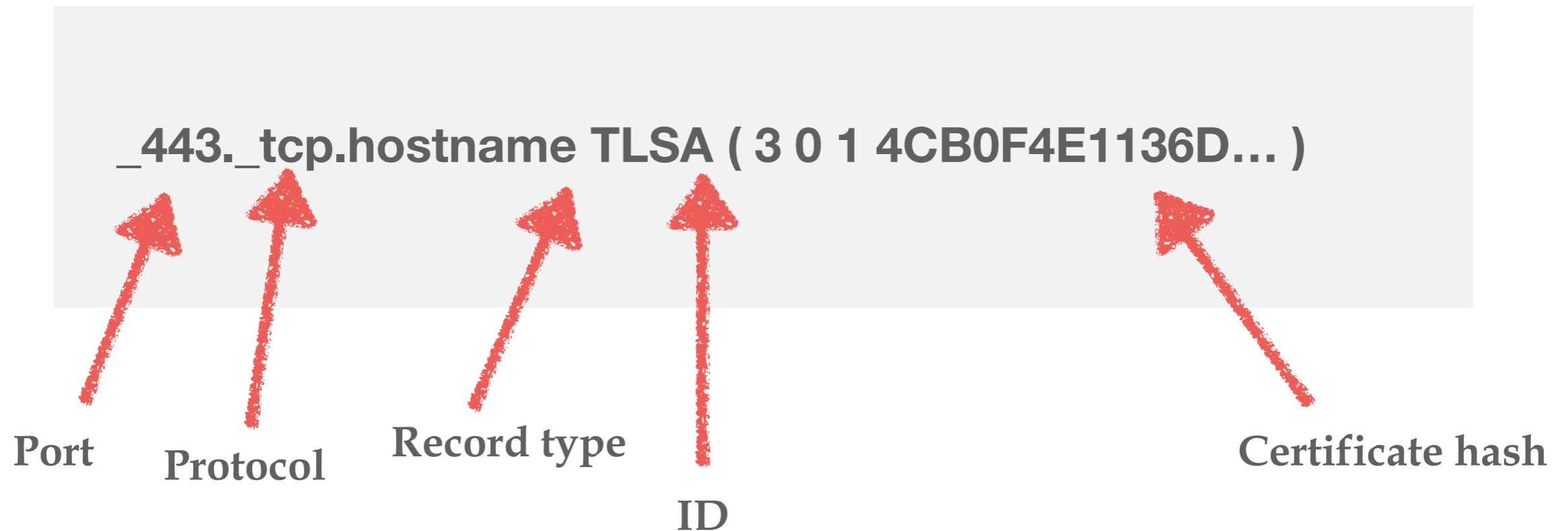


# Certificates



- **TLSA Record**

- Stands for Transport Layer Security A record
- You can put a certificate, to be validated by the client
- Also a self-signed certificate



# SSH host fingerprints



- Fingerprints can be stored in DNS records
- SSHFP records
- Validates the SSH keys using DNS

```
hostname IN SSHFP 1 1 372h1173312eqrqr  
hostname IN SSHFP 1 2 383h23r73rwdqwe
```



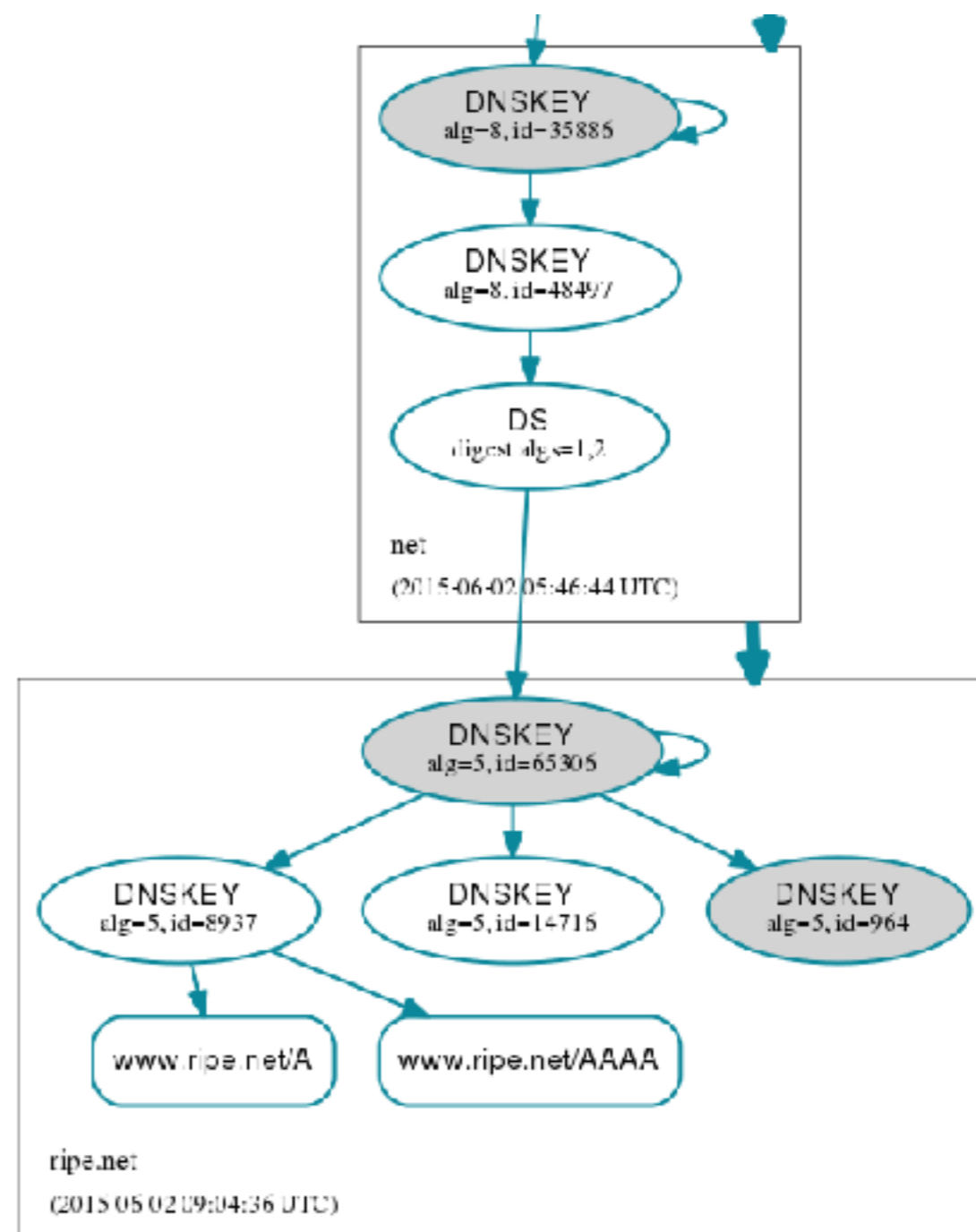
# Troubleshooting, Tips and Tricks

Section 11

# Troubleshooting basics



- If query returns *servfail*, DNSSEC did not validate
  - A non-existent record will generate NSEC/NSEC3
- Use dig, drill, or dnsviz.net to investigate
  - Also check port 53 TCP



# Changing registrar



- **Changing registrar is a quick procedure nowadays**
  - It also involves moving DS records
  - and checking if they're right
- **A bogus DS record breaks DNSSEC**
- **Solution is to remove DS records prior to transfer**
  - Add them back after the transfer

# DNSSEC TLSA Validator



- A plugin for browsers to check DNSSEC/TLSA
- Works on every browser (IE, Chrome, Safari, Firefox)



- <https://www.dnssec-validator.cz>



# DNSSEC-Trigger

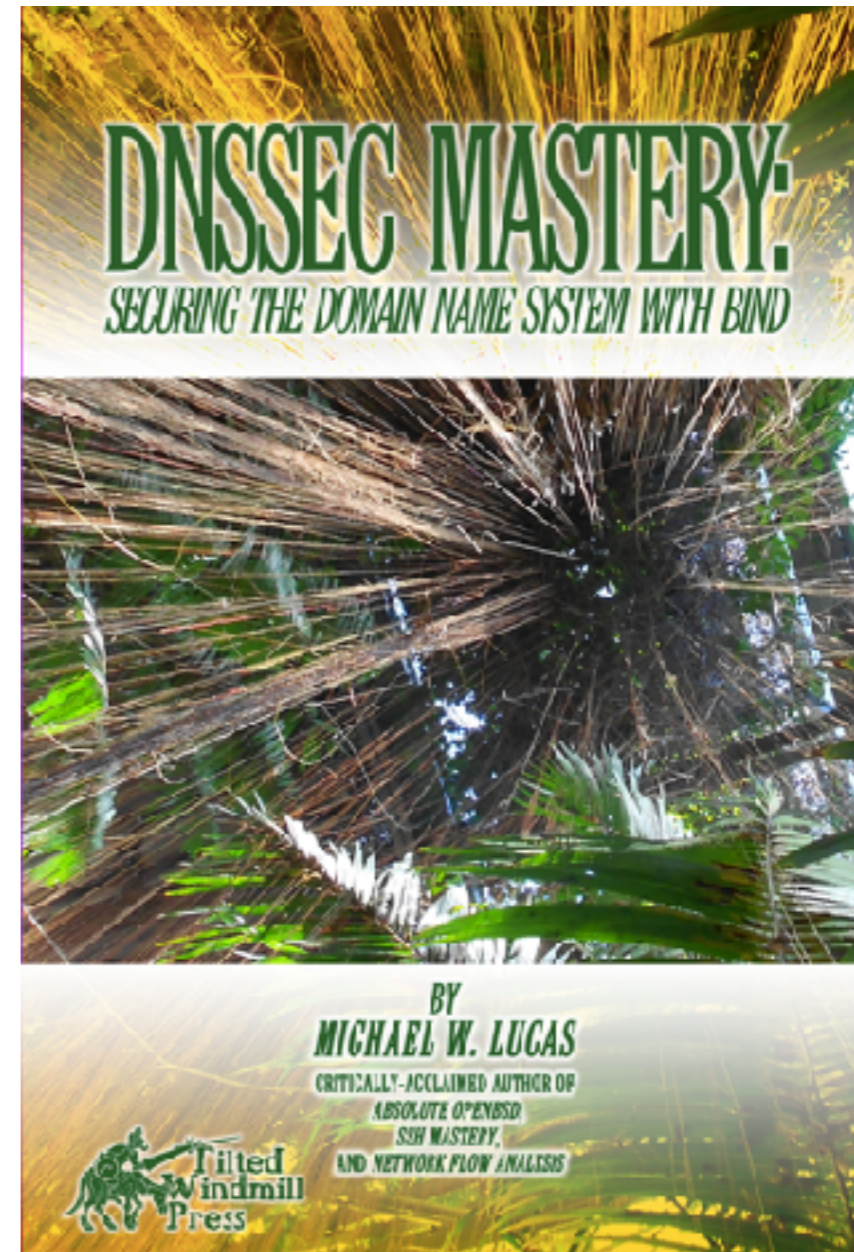


- **Clients rely on resolvers**
  - An attacker can still get into the way between the resolver and the user
- **Use DNSSEC-trigger to run a local resolver with DNSSEC capabilities**
- **<http://www.nlnetlabs.nl/projects/dnssec-trigger/>**

# DNSSEC Mastery



- Published in December 2012
- Walkthrough in configuring DNSSEC on BIND
- Available from <https://www.tiltedwindmillpress.com/>



# BIND ARM



- **Bind Administrator Reference Manual**
- **One-stop resource for every aspect of BIND**
- **<ftp://ftp.isc.org/isc/bind9>**



# ***Check and Troubleshoot***

*Exercise E*



# RIPE NCC

Academy

**Graduate to the next level!**

<https://academy.ripe.net>



<https://www.ripe.net/training/dnssec/survey>



twitter

@TrainingRIPENCC

**The End!**

**Край**

**Y Diwedd**

**النهاية**

**Соңы**

**ჟღერჟ**

**Fí**

**Finis**

**Ende**

**Finvezh**

**Liðugt**

**Кінець**

**Konec**

**Kraj**

**Ěnn**

**Fund**

**پایان**

**Lõpp**

**Beigas**

**Vége**

**Son**

**Край**

**An Críoch**

**הסוף**

**Fine**

**Endir**

**Sfârșit**

**Fin**

**Τέλος**

**Einde**

**Канец**

**Конец**

**Slut**

**Slutt**

**დასასრული**

**Pabaiga**

**Fim**

**Амаіа**

**Lopru**

**Tmíem**

**Koniec**