# DNSSEC

## Domain Name System Security Extensions

Jelte Jansen

Radboud University, June 2nd, 2015

SIDN labs
Internet Research & Innovation

SIDN

# DNSSEC

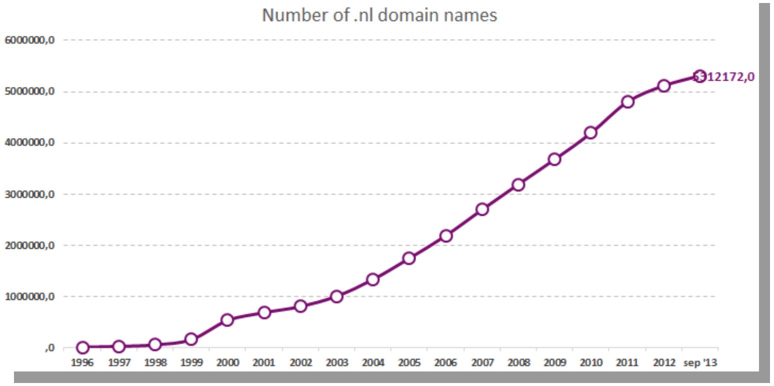
## Rebuilding the airplane in-flight since 1995

Jelte Jansen

Radboud University, June 2nd, 2015

# SIDN

- Domain name registry for .nl ccTLD
- > 5.5 million domain names
- 2.4 million domain names secured with DNSSEC



# SIDN Labs

- R&D team SIDN
- Improve services of SIDN
- Center of expertise
- Increase security of the Internet (mostly in .nl)
- Facilitate external research

De DNSSEC-validatiemonitor

Marco Davids | 25 maart 2014 - 14:01 uur

In Nederland lopen we wereldwijd voorop als het aankomt op het aantal met DNSSEC beveiligde .nl-domeinnamen. Ruim 30% van alle bijna 5,5 miljoen domeinnamen is gesigneerd, alles bij elkaar bijna 1,7 miljoen domeinnamen. Wanneer er een bepaald percentage aan fouten in DNSSEC-beveiliging sluipt, gaat het bij dit soort aantallen al gauw om enkele honderden domeinnamen. Teveel, vonden we bij SIDN Labs.

meer

TAGS dnssec validatie errors valideren

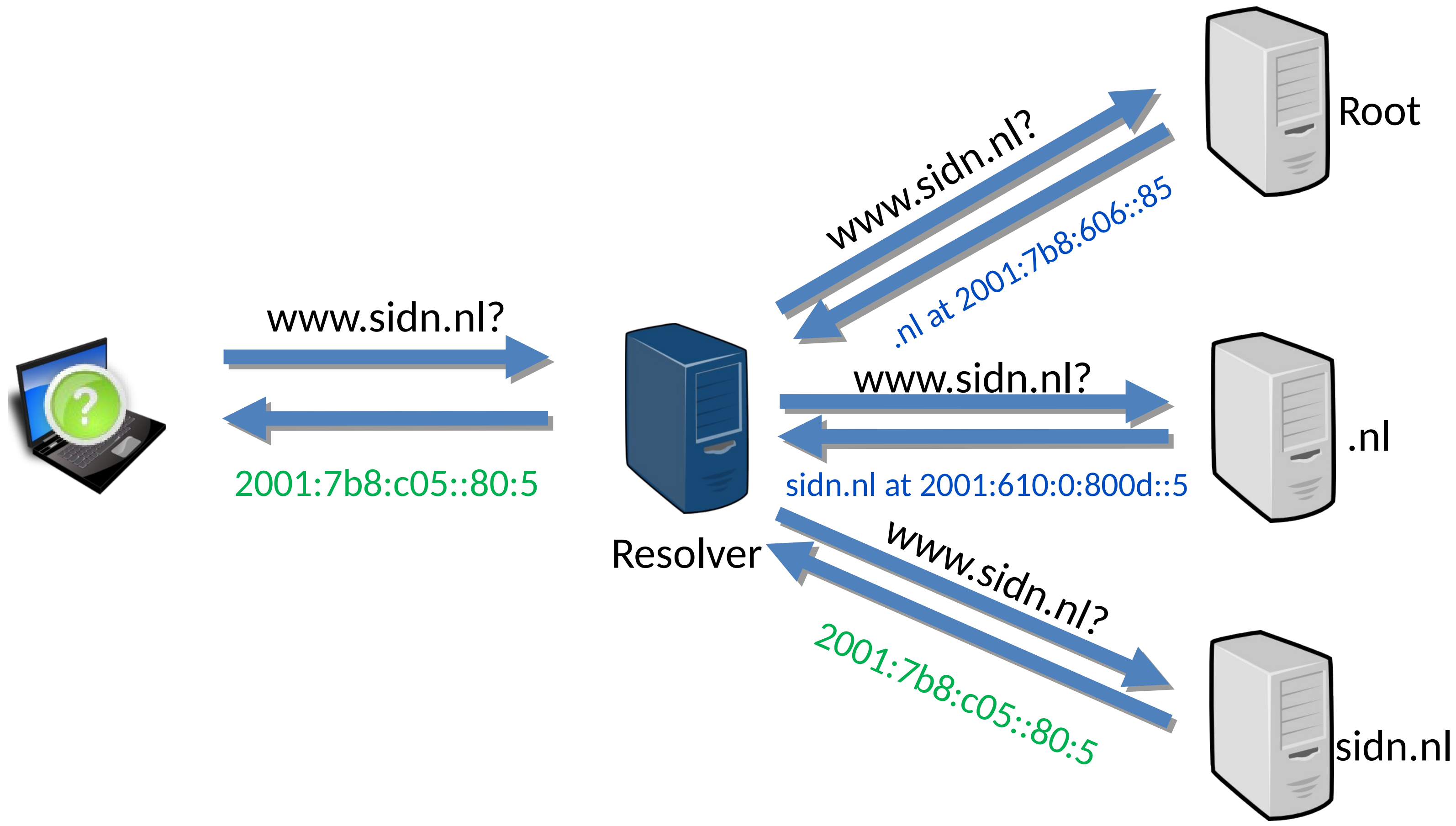
Twitter

SIDN host een RIPE Atlas Anchor

Marco Davids | 11 februari 2014 - 11:55 uur

Je hebt ze misschien wel eens gezien, langs de weg of op een vliegveld. Sensoren die bijvoorbeeld de luchtkwaliteit meten, of de hoeveelheid decibels. Misschien sta je er niet bij stil, maar ergens is het heel logisch dat er organisaties zijn, wiens verantwoording het is om dat soort gegevens permanent te meten.

# DNSSEC Recap in 27.4 seconds



# Addresses are just one type of DNS data

- A (IPv4 address)
- AAAA (IPv6 address)
- MX (mail server name)
- NS (name server name)
- TXT (arbitrary text data)
- CNAME (canonical name pointer)
- GPOS (geographical position)

and many, many more

# DNS is used in…

- web
- mail
- firewalls
- network configuration
- voip
- games
- ip-television
- p2p
- streaming
- etc.
- etc.
- etc.

With very few exceptions, nearly *everything* uses DNS.

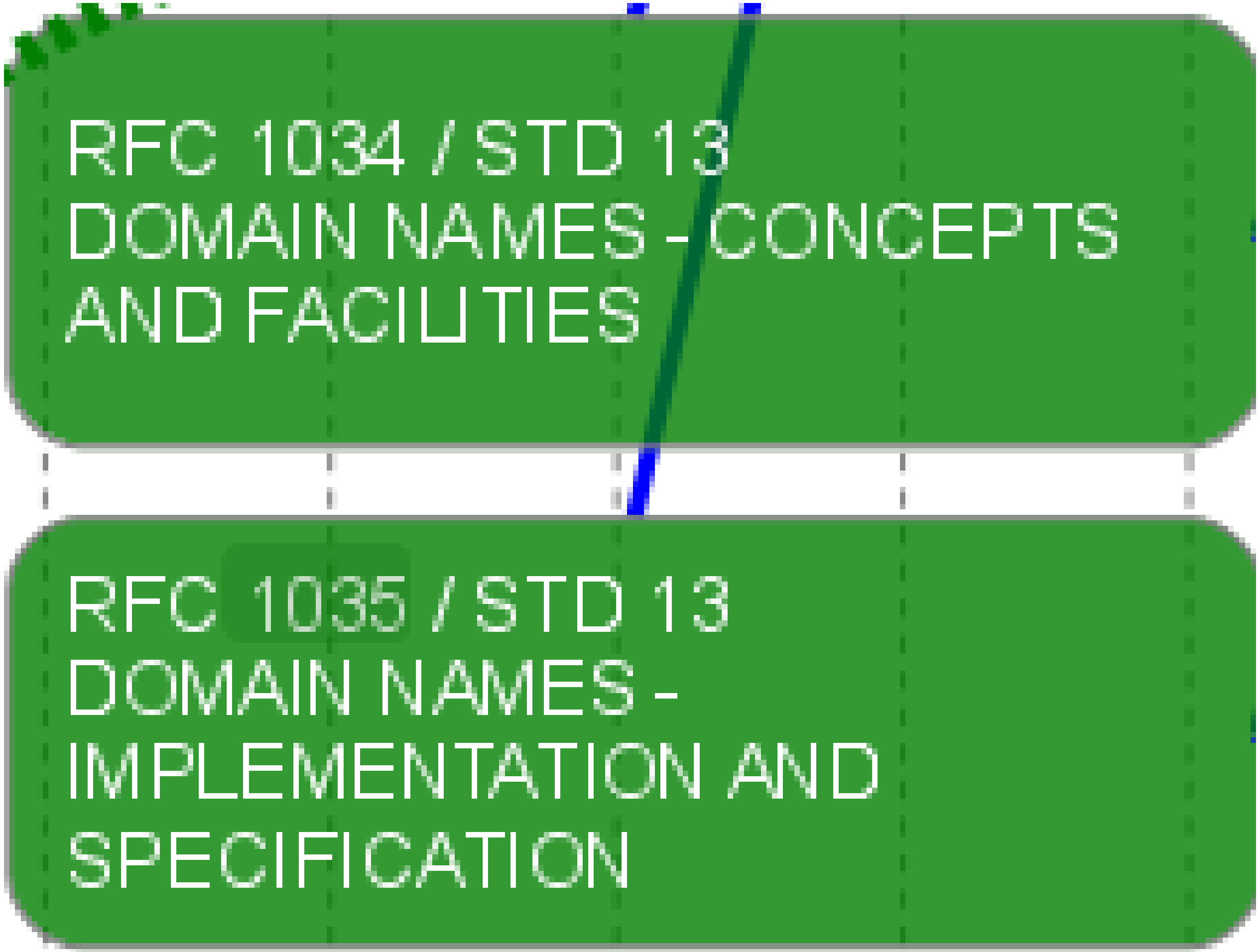
# DNS is used in…

- web
- mail
- firewal
- networ
- voip
- games
- ip-tele
- p2p
- stream
- etc.
- etc.
- etc.

With ver

# DNS is old

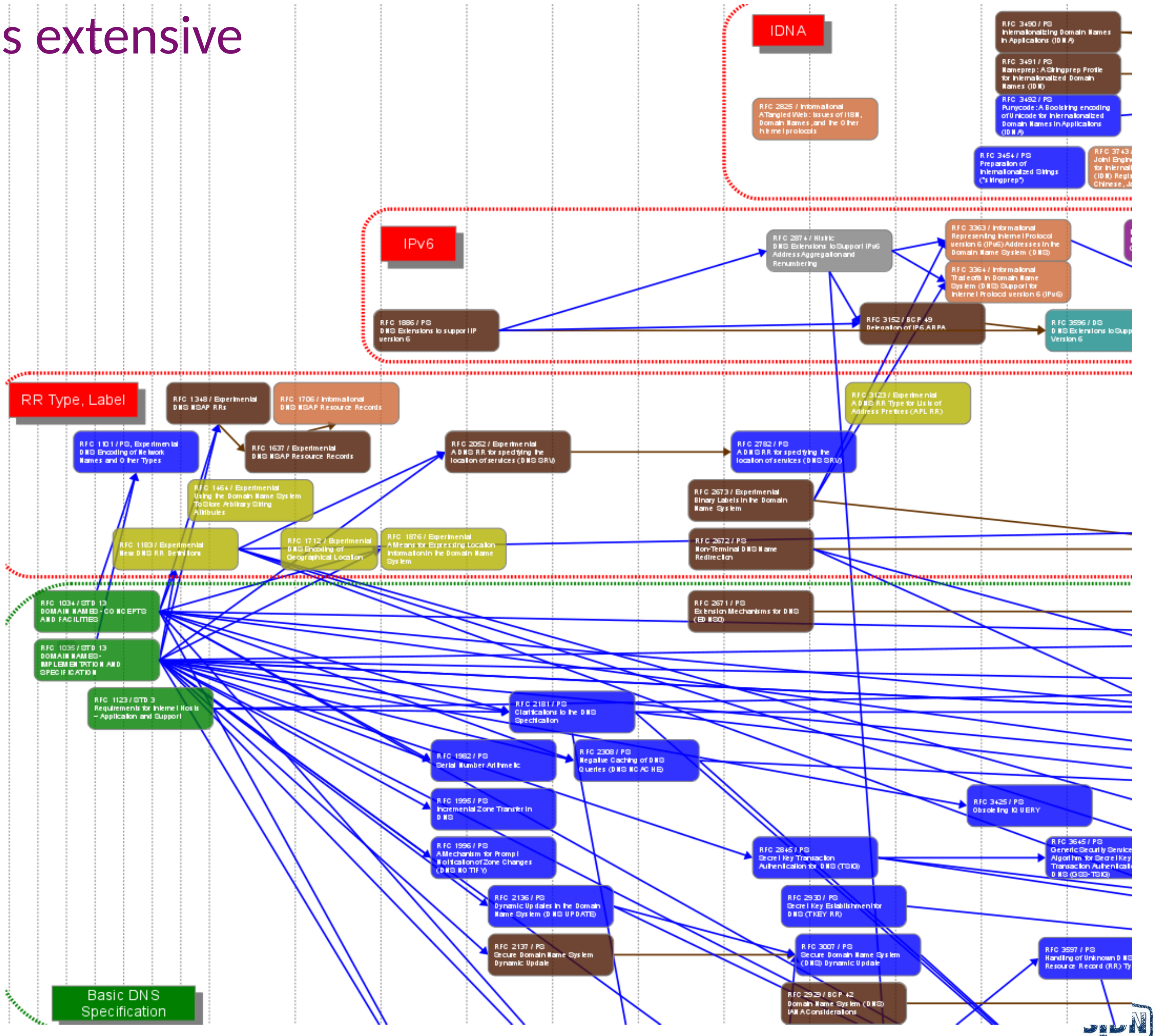
Defined in RFC 1034 / RFC 1035 (1986)



RFC 1034 / STD 13
DOMAIN NAMES - CONCEPTS AND FACILITIES

RFC 1035 / STD 13
DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION

# DNS is extensive



IDNA

RFC 3490 / PS
Internationalizing Domain Names in Applications (IDNA)

RFC 3491 / PS
Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)

RFC 3492 / PS
Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)

RFC 2825 / Informational
A Tangled Web: Issues of I18N, Domain Names, and the Other Internet protocols

RFC 3454 / PS
Preparation of Internationalized Strings ("stringprep")

RFC 3743 /
Joint Engin
for Internat
(IDN) Regis
Chinese, J

IPv6

RFC 1886 / PS
DNS Extensions to support IP version 6

RFC 2874 / Historic
DNS Extensions to Support IPv6 Address Aggregation and Renumbering

RFC 3363 / Informational
Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)

RFC 3364 / Informational
Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6)

RFC 3152 / BCP 49
Delegation of IP6.ARPA

RFC 3596 / DS
DNS Extensions to Supp
Version 6

RR Type, Label

RFC 1348 / Experimental
DNS NSAP RRs

RFC 1706 / Informational
DNS NSAP Resource Records

RFC 1101 / PS, Experimental
DNS Encoding of Network Names and Other Types

RFC 1637 / Experimental
DNS NSAP Resource Records

RFC 2052 / Experimental
A DNS RR for specifying the location of services (DNS SRV)

RFC 2782 / PS
A DNS RR for specifying the location of services (DNS SRV)

RFC 3123 / Experimental
A DNS RR Type for Lists of Address Prefixes (APL RR)

RFC 1464 / Experimental
Using the Domain Name System To Store Arbitrary String Attributes

RFC 1183 / Experimental
New DNS RR Definitions

RFC 1712 / Experimental
DNS Encoding of Geographical Location

RFC 1876 / Experimental
A Means for Expressing Location Information in the Domain Name System

RFC 2673 / Experimental
Binary Labels in the Domain Name System

RFC 2672 / PS
Non-Terminal DNS Name Redirection

RFC 2671 / PS
Extension Mechanisms for DNS (EDNS0)

RFC 1034 / STD 13
DOMAIN NAMES - CONCEPTS AND FACILITIES

RFC 1035 / STD 13
DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION

RFC 1123 / STD 3
Requirements for Internet Hosts – Application and Support

RFC 2181 / PS
Clarifications to the DNS Specification

RFC 1982 / PS
Serial Number Arithmetic

RFC 2308 / PS
Negative Caching of DNS Queries (DNS NCACHE)

RFC 1995 / PS
Incremental Zone Transfer in DNS

RFC 1996 / PS
A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)

RFC 2136 / PS
Dynamic Updates in the Domain Name System (DNS UPDATE)

RFC 2137 / PS
Secure Domain Name System Dynamic Update

RFC 3425 / PS
Obsoleting IQUERY

RFC 2845 / PS
Secret Key Transaction Authentication for DNS (TSIG)

RFC 3645 / PS
Generic Security Service
Algorithm for Secret Key
Transaction Authenticati
DNS (GSS-TSIG)

RFC 2930 / PS
Secret Key Establishment for DNS (TKEY RR)

RFC 3007 / PS
Secure Domain Name System (DNS) Dynamic Update

RFC 3597 / PS
Handling of Unknown DNS
Resource Record (RR) Ty

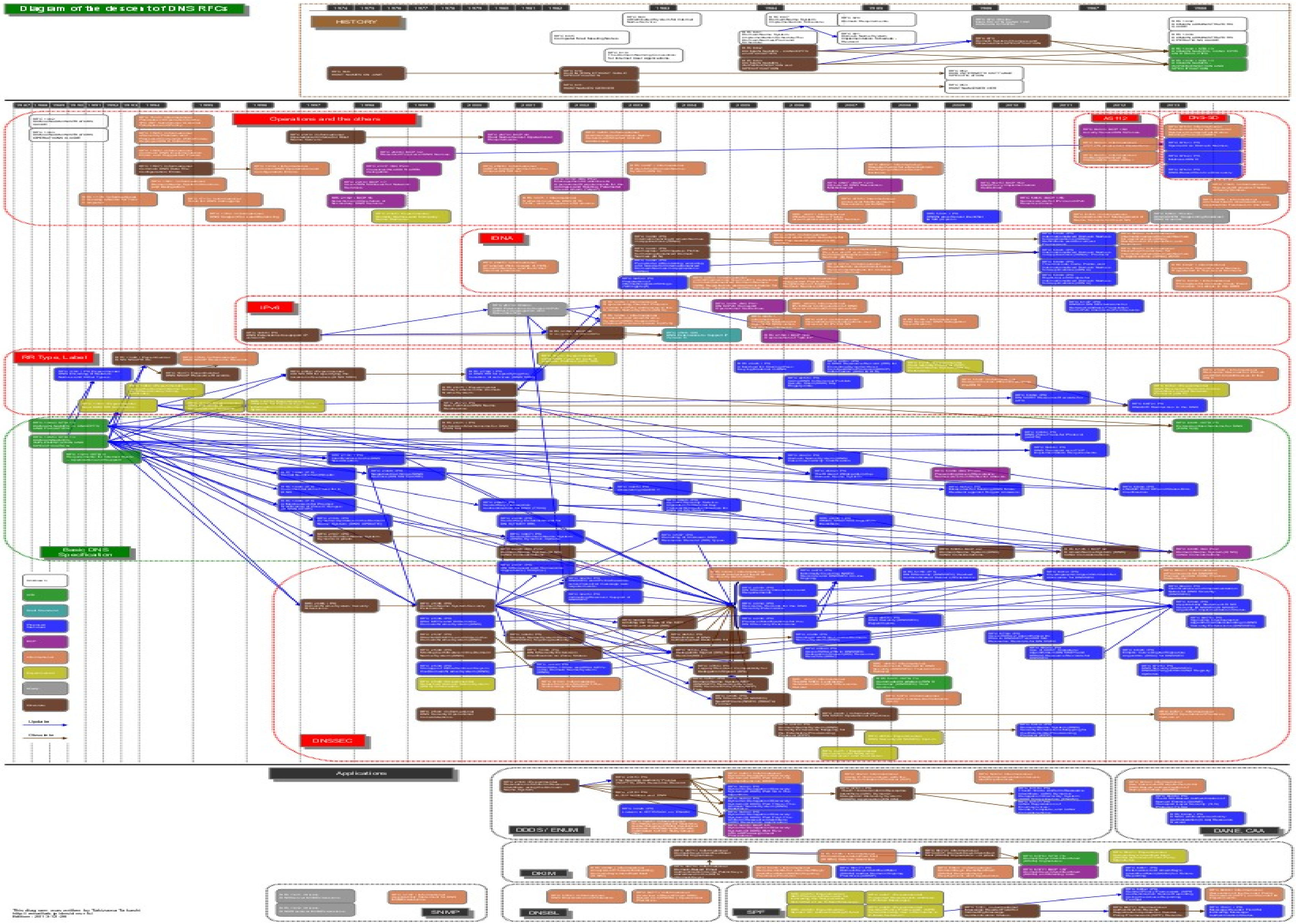
RFC 2929 / BCP 42
Domain Name System (DNS) IANA Considerations

Basic DNS Specification

labs
Internet Research & Innovation

SIDN

# DNS is extensive



# DNS is scalable

- Over 250 million second-level domain names
- Distributed at delegation points
- You can run your own dns for your own zone
- Answers can be easily cached

# DNS is insecure as %?)!

- Mostly UDP so no handshake/source verification
- Originally, only 16 bits of data to match answer to question (qid section)
- Easy to spoof answers

# DNS is insecure as %?)!

SEEING TROUBLE
Security researcher Dan Kaminsky first spotted a basic vulnerability in the Internet last winter.

## The Flaw at the Heart of the Internet

DAN KAMINSKY DISCOVERED A FUNDAMENTAL SECURITY PROBLEM IN THE INTERNET AND GOT PEOPLE TO CARE IN TIME TO FIX IT. IT'S A DRAMATIC STORY WITH A HAPPY ENDING ... BUT WE WERE LUCKY THIS TIME.

*By* ERICA NAONE

Dan Kaminsky, uncharacteristically, was not looking for bugs earlier this year when he happened upon a flaw at the core of the Internet. The security researcher was using his knowledge of Internet infrastructure to come up with a better way to stream videos to users. Kaminsky's expertise is in the Internet's domain name system (DNS), the protocol responsible for matching websites' URLs with the numeric addresses of the servers that host them. The same content can be hosted by multiple servers with several addresses, and Kaminsky thought he had a great trick for directing users to the servers best able to handle their requests at any given moment.

Normally, DNS is reliable but not nimble. When a computer–say, a server that helps direct traffic across Comcast's network–requests the numerical address associated with a given URL, it stores the answer for a period of time known as "time to live," which can be anywhere from seconds to days. This helps to reduce the number of requests the server makes. Kaminsky's idea was to bypass the time to live, allowing the server to get a fresh answer every time it wanted to know a site's address. Consequently, traffic on Comcast's network would be sent to the optimal address at every moment, rather than to whatever address had already been stored. Kaminsky was sure that the strategy could significantly speed up content distribution.

It was only later, after talking casually about the idea with a friend, that Kaminsky realized his "trick" could completely break the security of the domain name system and, therefore, of the Internet itself. The time to live, it turns out, was at the core of DNS security; being able to bypass it allowed for a wide variety of attacks. Kaminsky wrote a little code to make sure the situation was as bad as he thought it was. "Once I saw it work, my stomach dropped," he says. "I thought, 'What the heck am I going to do about this? This affects everything.'"

Kaminsky's technique could be used to direct Web surfers to any Web page an attacker chose. The most obvious use is to send people to phishing sites (websites designed to trick people into entering banking passwords and other personal information, allowing an attacker to steal their identities) or other fake versions of Web pages. But the danger is even worse: protocols such as those used to deliver e-mail or for secure communications over the Internet ultimately rely on DNS. A creative attacker could use Kaminsky's technique to intercept sensitive e-mail, or to create forged versions of the certificates that ensure secure transactions between users and banking websites. "Every day I find another domino," Kaminsky says. "Another thing falls over if DNS is bad. ... I mean, literally, you look around and see anything that's using a network–*anything* that's using a network–and it's probably using DNS."

Kaminsky called Paul Vixie, president of the Internet Systems Consortium, a nonprofit corporation that supports several aspects of Internet infrastructure, including the software most commonly used in the domain name system. "Usually, if somebody wants to report a problem, you expect that it's going to take a fair amount of time for them to explain it–maybe a whiteboard, maybe a Word document or two," Vixie says. "In this case, it took 20 seconds for him to explain the problem, and another 20 seconds for him to answer my objections. After that, I said, 'Dan, I am speaking to you over an unsecure cell phone. Please do not ever say to anyone what you just said to me over an unsecure cell phone again.'"

Perhaps most frightening was that because the vulnerability was not located in any particular hardware or software but in the design of the DNS protocol itself, it wasn't clear how to fix it. In secret, Kaminsky and Vixie gathered together some of the top DNS experts in the world: people from the U.S. government and

*Photograph by* JOHN KEATLEY

# Choices of protection:

- Protect the channel (DNSCurve, DNSCrypt)
- Protect the data (DNSSEC)

# A short history of DNSSEC

- Flaws identified around 1990 (and probably earlier)

- Flaws publicized in 1995

- Attempt 1 for DNSSEC in 1997 (RFC 2065)
  - Too hard to implement

- Attempt 2 for DNSSEC in 1999 (RFC 2535)
  - Did not scale

- Attempt 3 for DNSSEC ('dnssec-bis'...) in 2005 (RFC 4033-4035)

- Additional features later added to 3rd version (e.g.  NSEC3 in 2008)

- Note: Kaminsky's exploit was published in 2008

# Attacks are real

SEEING TROUBLE
Security researcher Dan Kaminsky first spotted a basic vulnerability in the Internet last winter.

## The Flaw at the Heart of the Internet

DAN KAMINSKY DISCOVERED A FUNDAMENTAL SECURITY PROBLEM IN THE INTERNET AND GOT PEOPLE TO CARE IN TIME TO FIX IT. IT'S A DRAMATIC STORY WITH A HAPPY ENDING ... BUT WE WERE LUCKY THIS TIME.

*By* ERICA NAONE

Dan Kaminsky, uncharacteristically, was not looking for bugs earlier this year when he happened upon a flaw at the core of the Internet. The security researcher was using his knowledge of Internet infrastructure to come up with a better way to stream videos to users. Kaminsky's expertise is in the Internet's domain name system (DNS), the protocol responsible for matching websites' URLs with the numeric addresses of the servers that host them. The same content can be hosted by multiple servers with several addresses, and Kaminsky thought he had a great trick for directing users to the servers best able to handle their requests at any given moment.

Normally, DNS is reliable but not nimble. When a computer–say, a server that helps direct traffic across Comcast's network–requests the numerical address associated with a given URL, it stores the answer for a period of time known as "time to live," which can be anywhere from seconds to days. This helps to reduce the number of requests the server makes. Kaminsky's idea was to bypass the time to live, allowing the server to get a fresh answer every time it wanted to know a site's address. Consequently, traffic on Comcast's network would be sent to the optimal address at every moment, rather than to whatever address had already been stored. Kaminsky was sure that the strategy could significantly speed up content distribution.

It was only later, after talking casually about the idea with a friend, that Kaminsky realized his "trick" could completely break the security of the domain name system and, therefore, of the Internet itself. The time to live, it turns out, was at the core of DNS security; being able to bypass it allowed for a wide variety of attacks. Kaminsky wrote a little code to make sure the situation was as bad as he thought it was. "Once I saw it work, my stomach dropped," he says. "I thought, 'What the heck am I going to do about this? This affects everything.'"

Kaminsky's technique could be used to direct Web surfers to any Web page an attacker chose. The most obvious use is to send people to phishing sites (websites designed to trick people into entering banking passwords and other personal information, allowing an attacker to steal their identities) or other fake versions of Web pages. But the danger is even worse: protocols such as those used to deliver e-mail or for secure communications over the Internet ultimately rely on DNS. A creative attacker could use Kaminsky's technique to intercept sensitive e-mail, or to create forged versions of the certificates that ensure secure transactions between users and banking websites. "Every day I find another domino," Kaminsky says. "Another thing falls over if DNS is bad. ... I mean, literally, you look around and see anything that's using a network–*anything* that's using a network–and it's probably using DNS."

Kaminsky called Paul Vixie, president of the Internet Systems Consortium, a nonprofit corporation that supports several aspects of Internet infrastructure, including the software most commonly used in the domain name system. "Usually, if somebody wants to report a problem, you expect that it's going to take a fair amount of time for them to explain it–maybe a whiteboard, maybe a Word document or two," Vixie says. "In this case, it took 20 seconds for him to explain the problem, and another 20 seconds for him to answer my objections. After that, I said, 'Dan, I am speaking to you over an unsecure cell phone. Please do not ever say to anyone what you just said to me over an unsecure cell phone again.'"

Perhaps most frightening was that because the vulnerability was not located in any particular hardware or software but in the design of the DNS protocol itself, it wasn't clear how to fix it. In secret, Kaminsky and Vixie gathered together some of the top DNS experts in the world: people from the U.S. government and

*Photograph by* JOHN KEATLEY

# Attacks are real

Security / Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

## Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

By Larry Seltzer | Posted 2009-04-22 Email Print

**OPINION: Attack shows the potential for serious spoofing attacks that could leave end users helpless. The only real solution is DNSSEC, which will take years to implement under the best of circumstances.**

art
ernet

A FUNDAMEN-
HE INTERNET
N TIME TO FIX IT.
A HAPPY END-
THIS TIME.

SEEING TROUBLE Security researcher Dan Kaminsky first spotted a basic vulnerability in the Internet last winter.

Dan Kaminsky, uncharacteristically, was not looking for bugs earlier this year when he happened upon a flaw at the core of the Internet. The security researcher was using his knowledge of Internet infrastructure to come up with a better way to stream videos to users. Kaminsky's expertise is in the Internet's domain name system (DNS), the protocol responsible for matching websites' URLs with the numeric addresses of the servers that host them. The same content can be hosted by multiple servers with several addresses, and Kaminsky thought he had a great trick for directing users to the servers best able to handle their requests at any given moment.

Normally, DNS is reliable but not nimble. When a computer–say, a server that helps direct traffic across Comcast's network–requests the numerical address associated with a given URL, it stores the answer for a period of time known as "time to live," which can be anywhere from seconds to days. This helps to reduce the number of requests the server makes. Kaminsky's idea was to bypass the time to live, allowing the server to get a fresh answer every time it wanted to know a site's address. Consequently, traffic on Comcast's network would be sent to the optimal address at every moment, rather than to whatever address had already been stored. Kaminsky was sure that the strategy could significantly speed up content distribution.

It was only later, after talking casually about the idea with a friend, that Kaminsky realized his "trick" could completely break the security of the domain name system and, therefore, of the Internet itself. The time to live, it turns out, was at the core of DNS security; being able to bypass it allowed for a wide variety of attacks. Kaminsky wrote a little code to make sure the situation was as bad as he thought it was. "Once I saw it work, my stomach dropped," he says. "I thought, 'What the heck am I going to do about this? This affects everything.'"

Kaminsky's technique could be used to direct Web surfers to any Web page an attacker chose. The most obvious use is to send people to phishing sites (websites designed to trick people into entering banking passwords and other personal information, allowing an attacker to steal their identities) or other fake versions of Web pages. But the danger is even worse: protocols such as those used to deliver e-mail or for secure communications over the Internet ultimately rely on DNS. A creative attacker could use Kaminsky's technique to intercept sensitive e-mail, or to create forged versions of the certificates that ensure secure transactions between users and banking websites. "Every day I find another domino," Kaminsky says. "Another thing falls over if DNS is bad. ... I mean, literally, you look around and see anything that's using a network–*anything* that's using a network–and it's probably using DNS."

Kaminsky called Paul Vixie, president of the Internet Systems Consortium, a nonprofit corporation that supports several aspects of Internet infrastructure, including the software most commonly used in the domain name system. "Usually, if somebody wants to report a problem, you expect that it's going to take a fair amount of time for them to explain it–maybe a whiteboard, maybe a Word document or two," Vixie says. "In this case, it took 20 seconds for him to explain the problem, and another 20 seconds for him to answer my objections. After that, I said, 'Dan, I am speaking to you over an unsecure cell phone. Please do not ever say to anyone what you just said to me over an unsecure cell phone again.'"

Perhaps most frightening was that because the vulnerability was not located in any particular hardware or software but in the design of the DNS protocol itself, it wasn't clear how to fix it. In secret, Kaminsky and Vixie gathered together some of the top DNS experts in the world: people from the U.S. government and

*Photograph by* JOHN KEATLEY

FEATURE STORY 63

# Attacks are real

Security / Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

## Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

By Larry Seltzer | Posted 2009-04-22 Email Print

**OPINION: Attack shows the potential for serious spoofing attacks that could leave end u only real solution is DNSSEC, which will take years to implement under the best of circu**

*Topic: Security*

## DNS cache poisoning attac exploited in the wild

***Summary:** UPDATE: Arbor Networks have provided more details in thei Attack Activity" analysis, SANS confirmed HD Moore's statement on DNS c DNS servers. Numerous independent sources are starting to see evidence o attempts on their local networks, in what appears to be an attempt to take "recent" DNS cache poisoning vulnerability :" client 143.*

By Dancho Danchev for Zero Day | July 29, 2008 -- 03:24 GMT (04:

Get the ZDNet Security newsletter now

**UPDATE:** Arbor Networks have provided more details in their "30 Days of DNS analysis, SANS confirmed HD Moore's statement on DNS cache poisoned AT&T independent sources are starting to see evidence of DNS cache poisoning atten networks, in what appears to be an attempt to take advantage of the "recent"

The DNS server at is NOT overtly vulnerable, however, it may be subtly vulnerable if POOR or FAIR.

| Test | Rating | Notes | Scatterploi |
|---|---|---|---|

# Attacks are real

## Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

By Larry Seltzer | Posted 2009-04-22

**OPINION: Attack shows the potential for serious spoofing attacks that could leave end u**
**only real solution is DNSSEC, which will take years to implement under the best of circu**

*Topic: Security*

## DNS cache poisoning attac exploited in the wild

***Summary:*** *UPDATE: Arbor Networks have provided more details in thei*
*HD Moore's statement on DNS c*
*ces are starting to see evidence o*
*ppears to be an attempt to take*
*y :" client 143.*

y | July 29, 2008 -- 03:24 GMT (04:

re details in their "30 Days of DNS
ent on DNS cache poisoned AT&T
ence of DNS cache poisoning atten
to take advantage of the "recent"
ble, however, it may be subtly vulnerable

## GOOGLE'S MALAYSIAN DOMAINS HIT WITH DNS CACHE POISONING ATTACK

PREVIOUS CONTRIBUTORS

OCT 11, 2013

Google's Malaysian domains google.com.my and google.my were hijacked, redirecting users to a webpage that announced the attack was perpetrated by a Pakistani group called Madleets. MYNIC, the sole administrator for web addresses in Malaysia confirmed the attack in a statement.

"We can confirm there was unauthorised redirection of www.google.com.my and www.google.my to

# Attacks are real

Security / Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

## Report Claims DNS Cache Poisoning Attack Against Brazilian Bank and ISP

By Larry Seltzer | Posted 2009-04-22 Email Print

**OPINION: Attack shows the potential for serious spoofing attacks that could leave end u**
**only real solution is DNSSEC, which will take years to implement under the best of circu**

*Topic: Security*

## DNS cache poisoning attac exploited in the wild

***Summary:*** *UPDATE: Arbor Networks have provided more details in thei*
*HD Moore's statement on DNS c*
*ces are starting to see evidence o*
*ppears to be an attempt to take c*
*y :" client 143.*

y | July 29, 2008 -- 03:24 GMT (04:

HOME « NEWS « TOP SECURITY STORIES « GOOGLE'S MALAYSIAN DOMAINS HIT WITH DNS CACHE POISONING...

## GOOGLE'S MALAYSIAN DOMAINS HIT WITH DNS CACHE POISONING ATTACK

# DNS poisoning slams web traffic from millions in China into the wrong hole

## ISP blames unspecified attack for morning outage

By John Leyden, 21 Jan 2014

# DNSSEC: Requirements

- Spoof protection (duh)
- Replay protection (outside of signature expiry)
- Backwards compatibility with non-DNSSEC-aware resolvers
- Support for offline signing (private keys not on nameservers)
- No global revocation list of public keys
- Crypto algorithm agility

More requirements defined in RFC 4033

# DNSSEC: OK let's just sign some stuff!

But what do you sign?

- Every packet?

Problem: answers not in fixed form, additional data, and depend on query.

- Every Resource Record (RR)?

Problem: a signature for *every* RR is too much data

# DNSSEC: OK let's just sign some stuff!

But what do you sign?

A lookup is a combination of **`(name, class, type)`**

So that becomes the minimum data set: the RRset

*A Resource Record Set (RRset), is the set of records of a given type for a given domain.*

| name | TTL | CLASS | TYPE | VALUE |
|---|---|---|---|---|
| `tjeb.nl.` | `3600` | `IN` | `NS` | `ns.tjeb.nl.` |
| `tjeb.nl.` | `3600` | `IN` | `NS` | `ext.ns.whyscream.net.` |
| `tjeb.nl.` | `3600` | `IN` | `NS` | `ns-ext.nlnetlabs.nl.` |
| `tjeb.nl.` | `3600` | `IN` | `TXT` | `"This is my zone"` |
| `tjeb.nl.` | `3600` | `IN` | `TXT` | `"There are many like it"` |
| `tjeb.nl.` | `3600` | `IN` | `TXT` | `"But this one is mine"` |
| `tjeb.nl.` | `3600` | `IN` | `A` | `178.18.82.80` |
| `www.tjeb.nl.` | `600` | `IN` | `A` | `178.18.82.80` |

(Q: How many RRsets in above zone snippet?)

# DNSSEC: Normal query

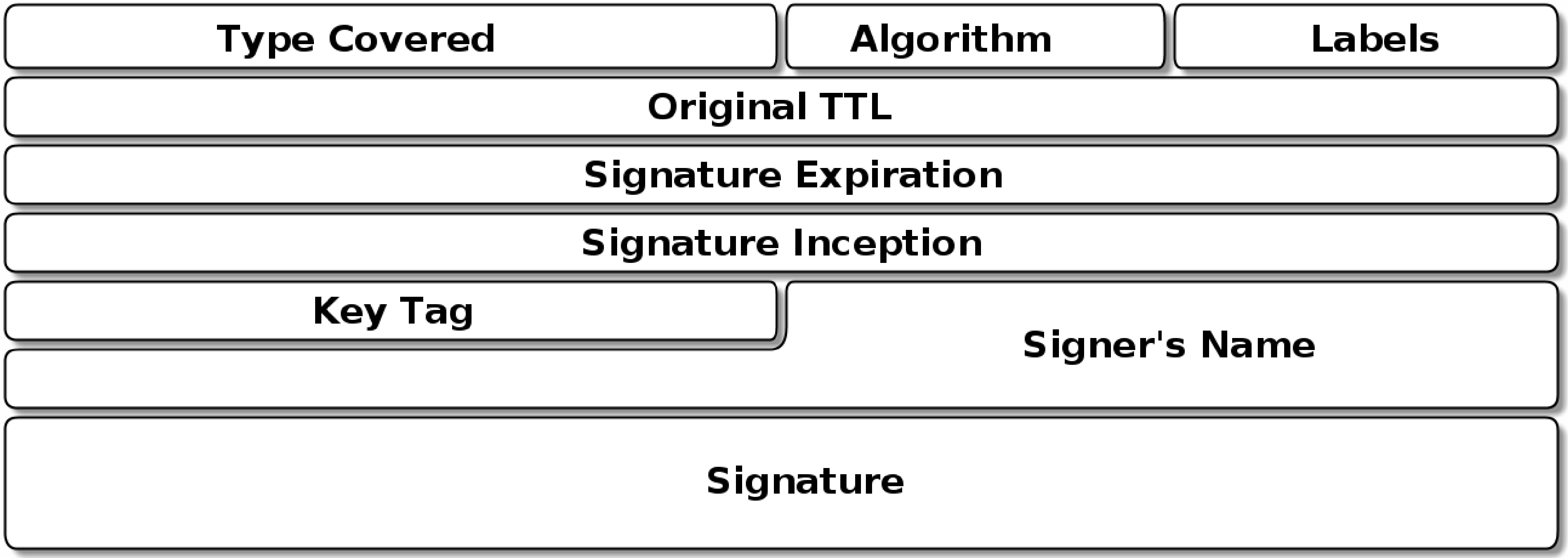
```
> dig NS tjeb.nl @ns.tjeb.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.    3600  IN NS ns-ext.nlnetlabs.nl.
tjeb.nl.    3600  IN NS ext.ns.whyscream.net.
tjeb.nl.    3600  IN NS ns.tjeb.nl.
<snip>
```

# DNSSEC: DNSSEC query

```
> dig +dnssec NS tjeb.nl @ns.tjeb.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.    3600  IN NS ns.tjeb.nl.
tjeb.nl.    3600  IN NS ns-ext.nlnetlabs.nl.
tjeb.nl.    3600  IN NS ext.ns.whyscream.net.
tjeb.nl.    3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
11S4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
<snip>
```

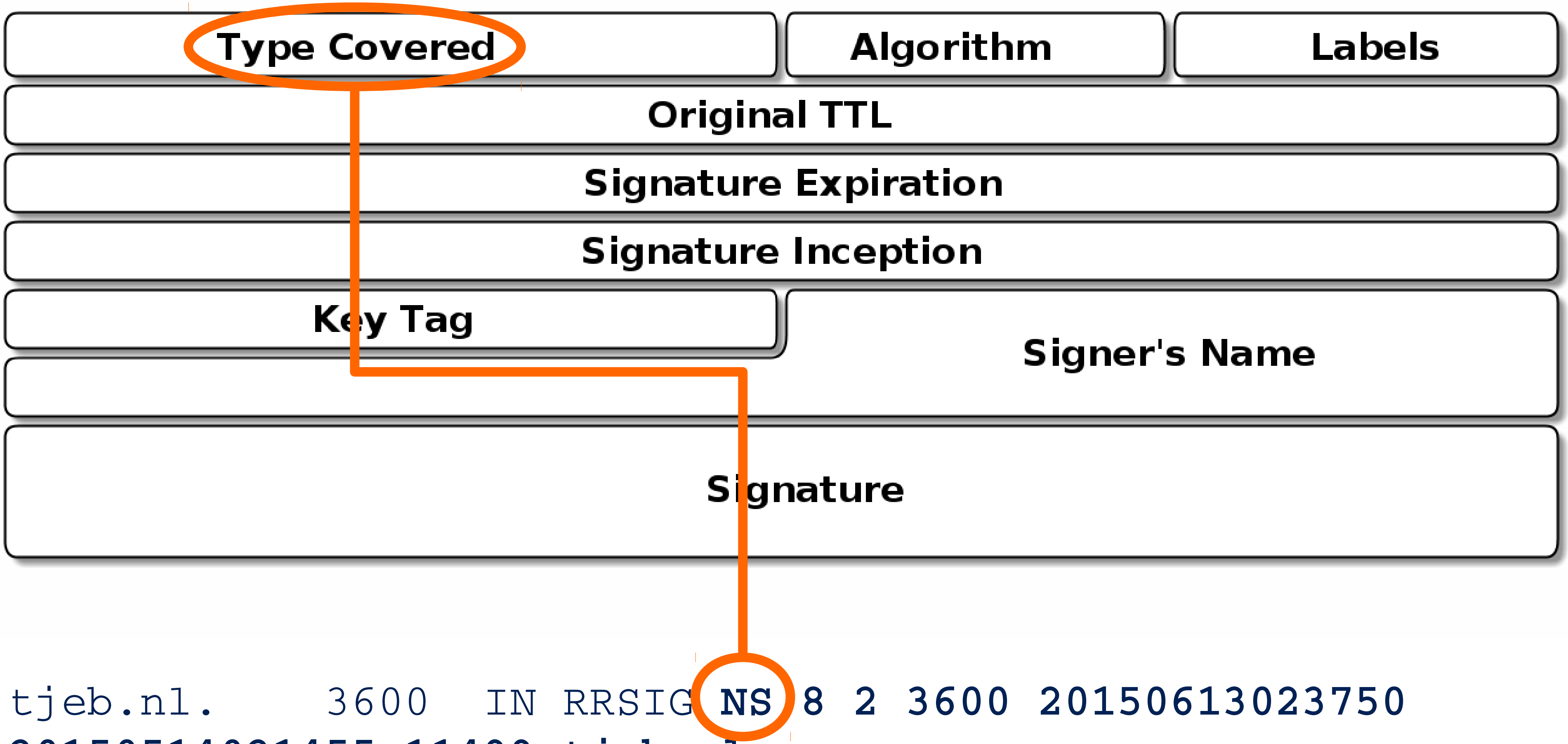
**RRSIG = Signature over an RRset**

# DNSSEC: RRSIG



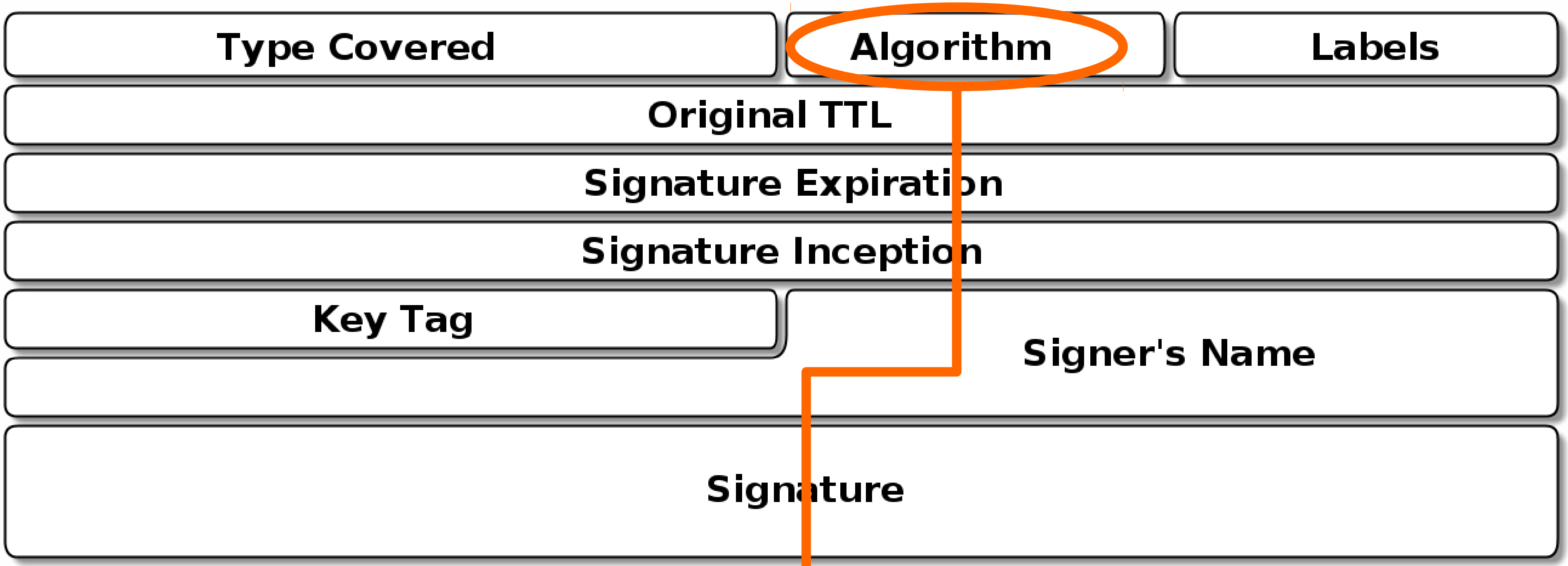
```
tjeb.nl.     3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
11S4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/51ZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



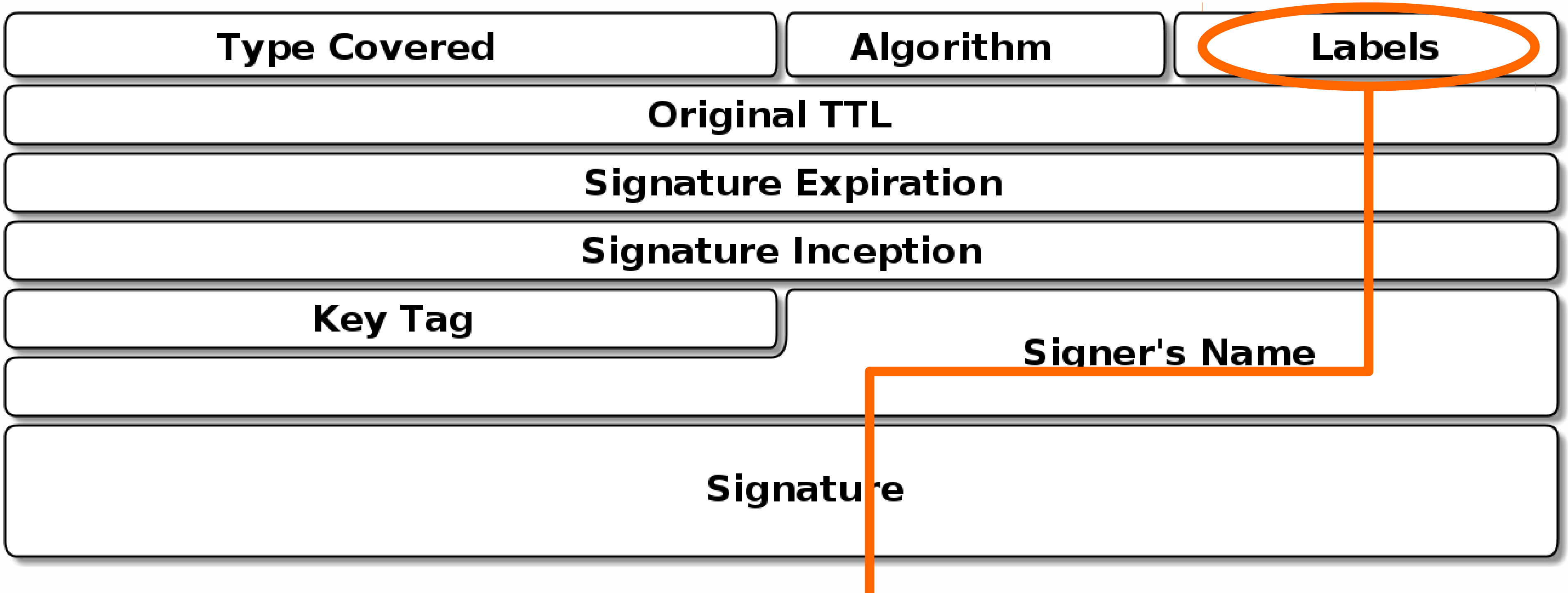
```
tjeb.nl.    3600   IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



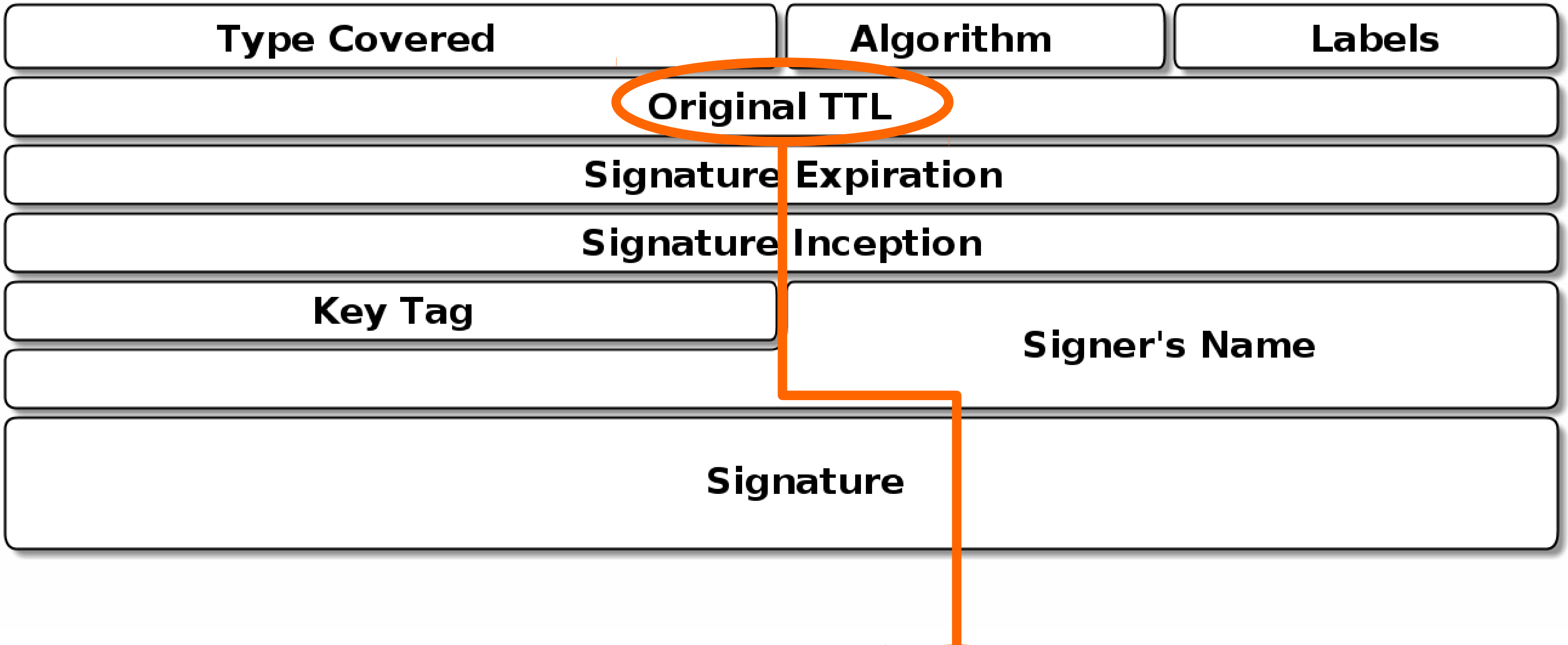
```
tjeb.nl.     3600   IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



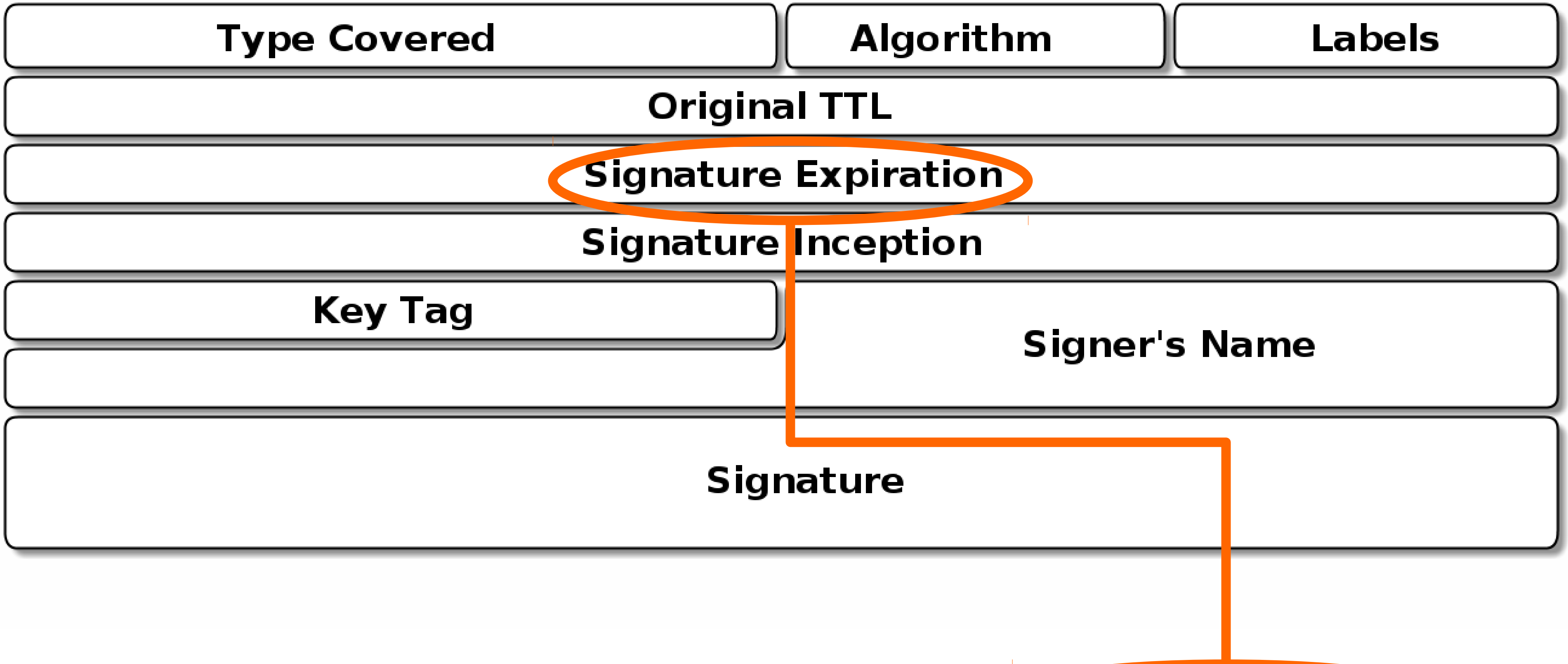
```
tjeb.nl.     3600   IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



```
tjeb.nl.     3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

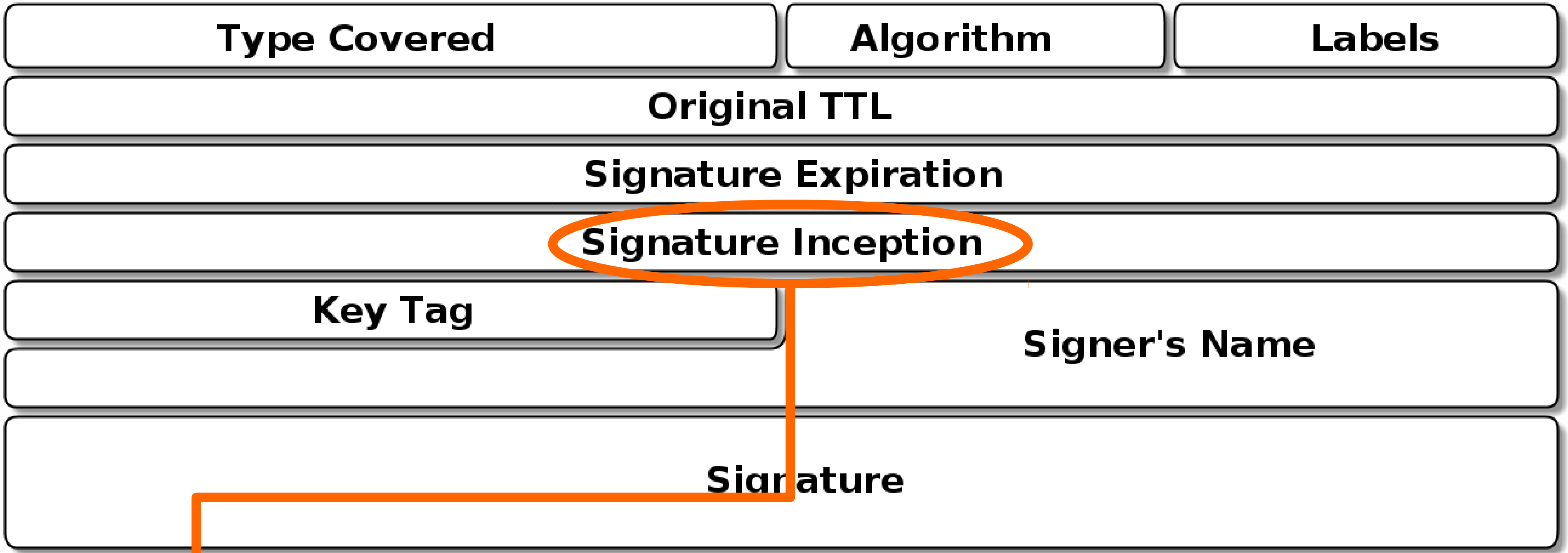
# DNSSEC: RRSIG



| | | |
|---|---|---|
| Type Covered | Algorithm | Labels |
| Original TTL | | |
| Signature Expiration | | |
| Signature Inception | | |
| Key Tag | Signer's Name | |
| Signature | | |

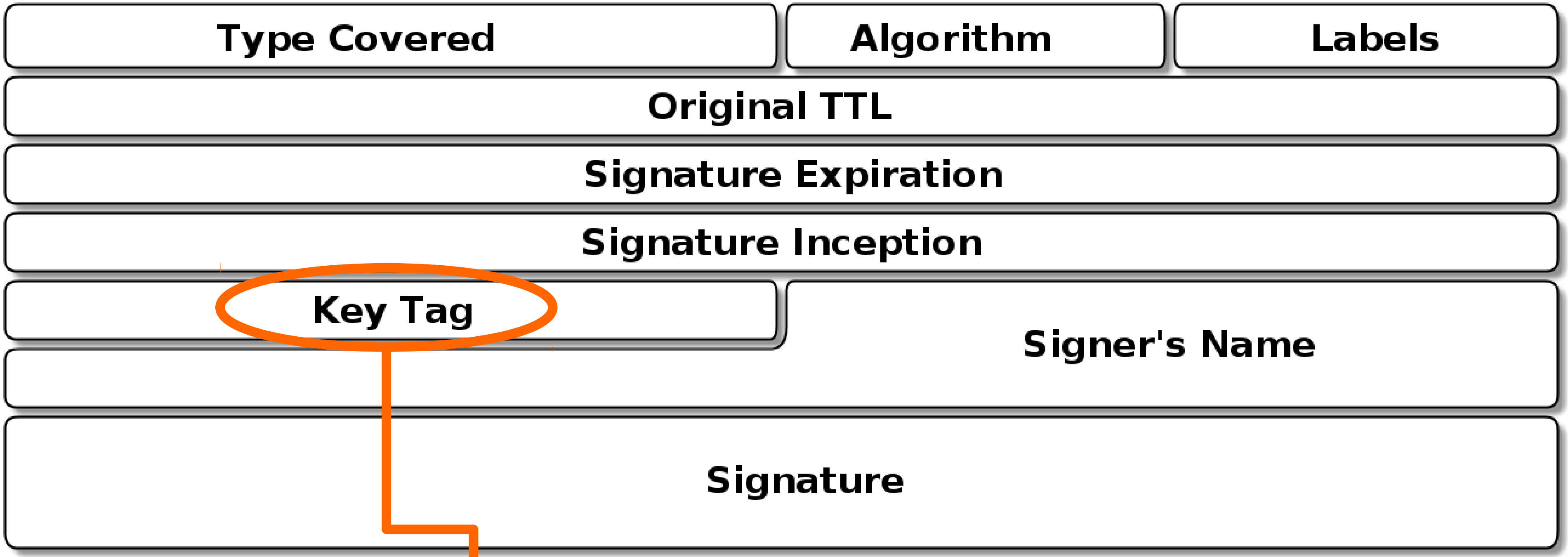
```
tjeb.nl.    3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



```
tjeb.nl.     3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

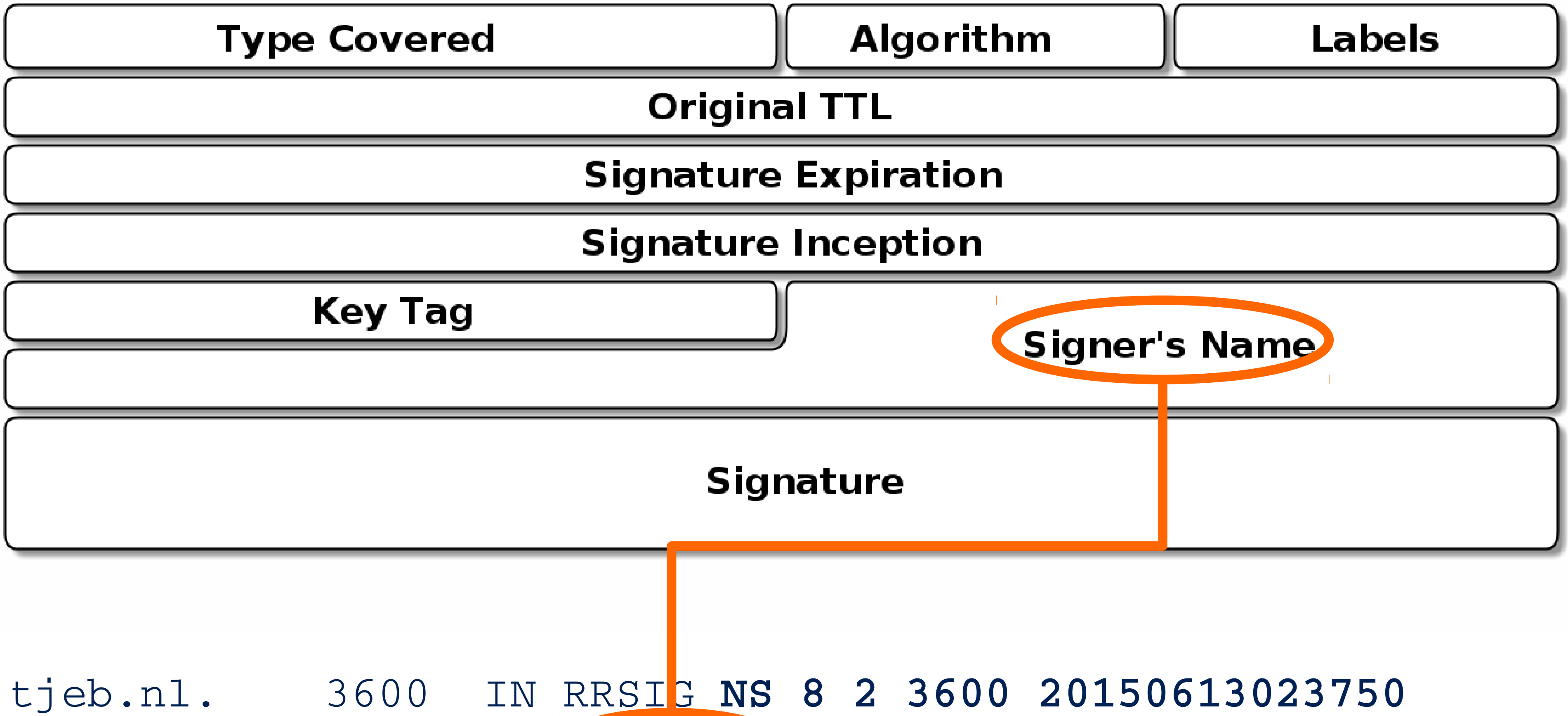
# DNSSEC: RRSIG



| | | |
|---|---|---|
| Type Covered | Algorithm | Labels |
| Original TTL | | |
| Signature Expiration | | |
| Signature Inception | | |
| Key Tag | Signer's Name | |
| Signature | | |

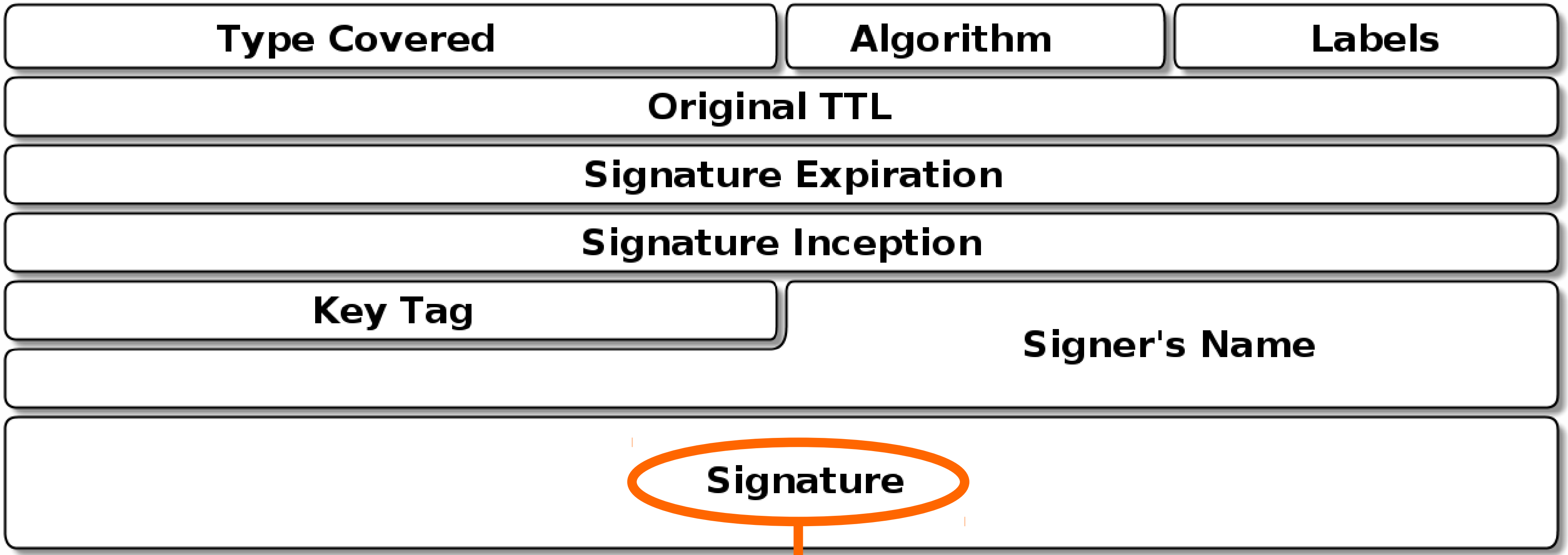
```
tjeb.nl.     3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



```
tjeb.nl.     3600   IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: RRSIG



```
tjeb.nl.     3600   IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/51ZwOqfy15o3b7PLDY
WwE=
```

# DNSSEC: Signature content

```
Signature = sign(RRSIG_RDATA | RR(1) | RR(2)... )
```

RRSIG_RDATA is the wire format of the RRSIG RDATA fields
with the Signer's Name field in canonical form and
the Signature field excluded

```
RR(i) = owner | type | class | TTL | RDATA length | RDATA
```

# DNSSEC: DNSSEC query (again)

```
> dig +dnssec NS tjeb.nl @ns.tjeb.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.    3600  IN NS ns.tjeb.nl.
tjeb.nl.    3600  IN NS ns-ext.nlnetlabs.nl.
tjeb.nl.    3600  IN NS ext.ns.whyscream.net.
tjeb.nl.    3600  IN RRSIG NS 8 2 3600 20150613023750
20150514021455 11499 tjeb.nl.
1lS4C3055edHcTXag7F+R1Kb61FTHNfQc2M6WEMGFD+yG+YGaKPCIZnD /
NTzmhOa+j8APVOWYSeyffxruyTt+bc6hLmnD3hGV9ScGxe3yg4mMecz
m2tTVh35RSLg1KaeJsFpGNwSkt9V8oSAnTvm2/5lZwOqfy15o3b7PLDY
WwE=
<snip>
```

# DNSSEC: Get public key for signature

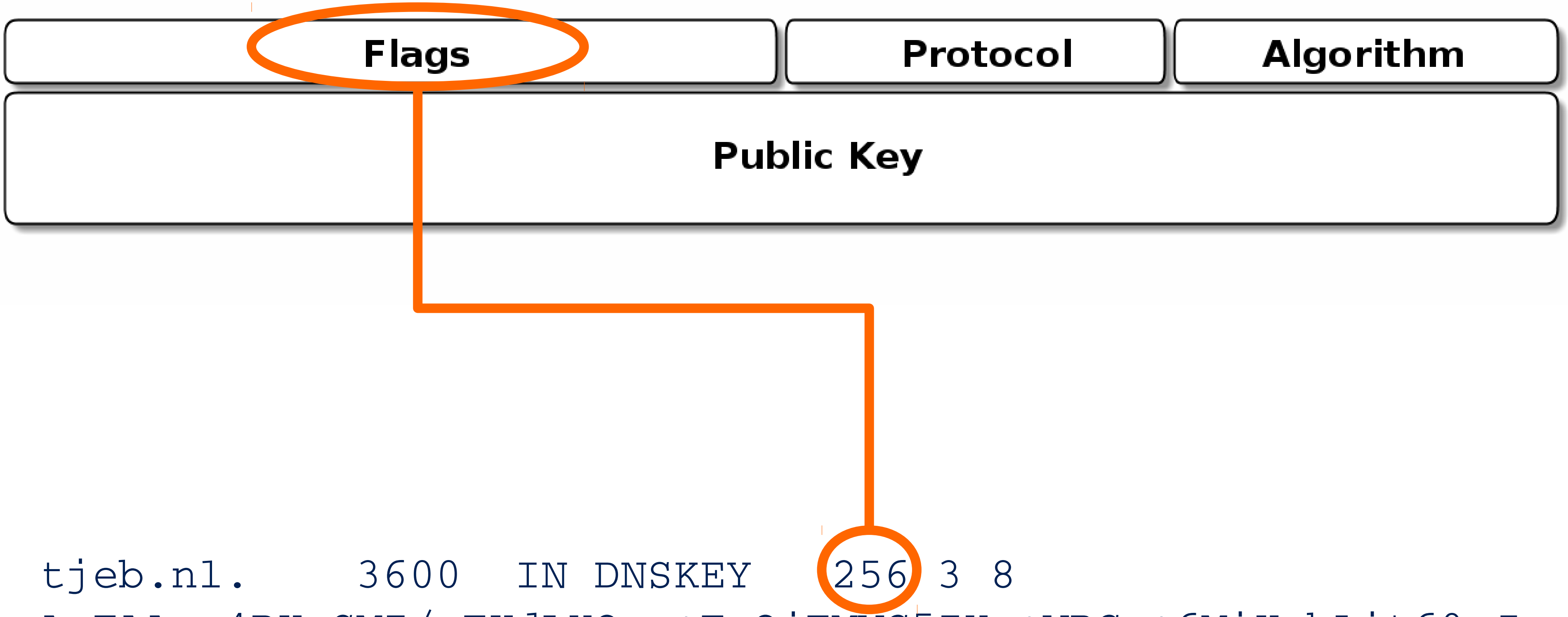
```
> dig DNSKEY tjeb.nl @ns.tjeb.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      3600   IN  DNSKEY 256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP761AFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
tjeb.nl.      3600   IN  DNSKEY 257 3 8
AwEAAcHR47QfC0dlPEQkAsKRh3VYFvUKlIerSdlT7HBS3/NOQ6ghVs9u
Yskdbs2pLSRbu4CSu6X0MgKZO0lxoJhi6FqBa33Oc0Mmp/dd6AW4pNdZ
a4icP6fKT+HcPbLU9dUsrjDo13iXgUy3gls5BLG9KnTaLzWs9KmxTInB
UHFLjZa70Fl+ILNfJ/e1D6eX3C104nmGSWpO6OB+nQDz46ra23eGJ7Ee
NAu1/uhPcqeXg3HWKjqHTzQW5XxVyMhdXx/ILC3SZhsqNqlkKZjmmHbg
7V1+iograUg1XEaxaOE25W9jrzvQnMxlZT8I9LTyyi1YArvxMCTcGkNW
Ri4Ca4/HEDs=
<snip>
```

***DNSKEY = Public key of keyset***

# DNSSEC: Get public key for signature

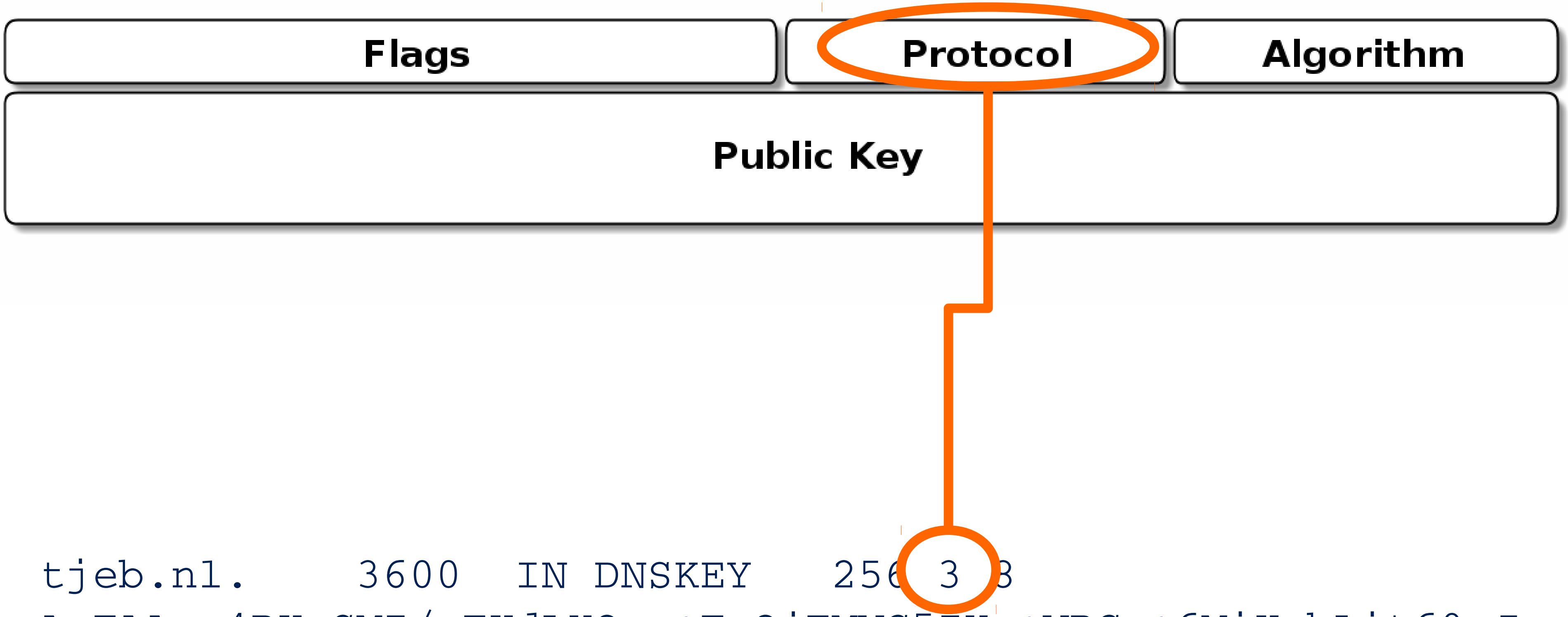
```
tjeb.nl.     3600   IN DNSKEY    256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP76lAFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
```

# DNSSEC: Get public key for signature



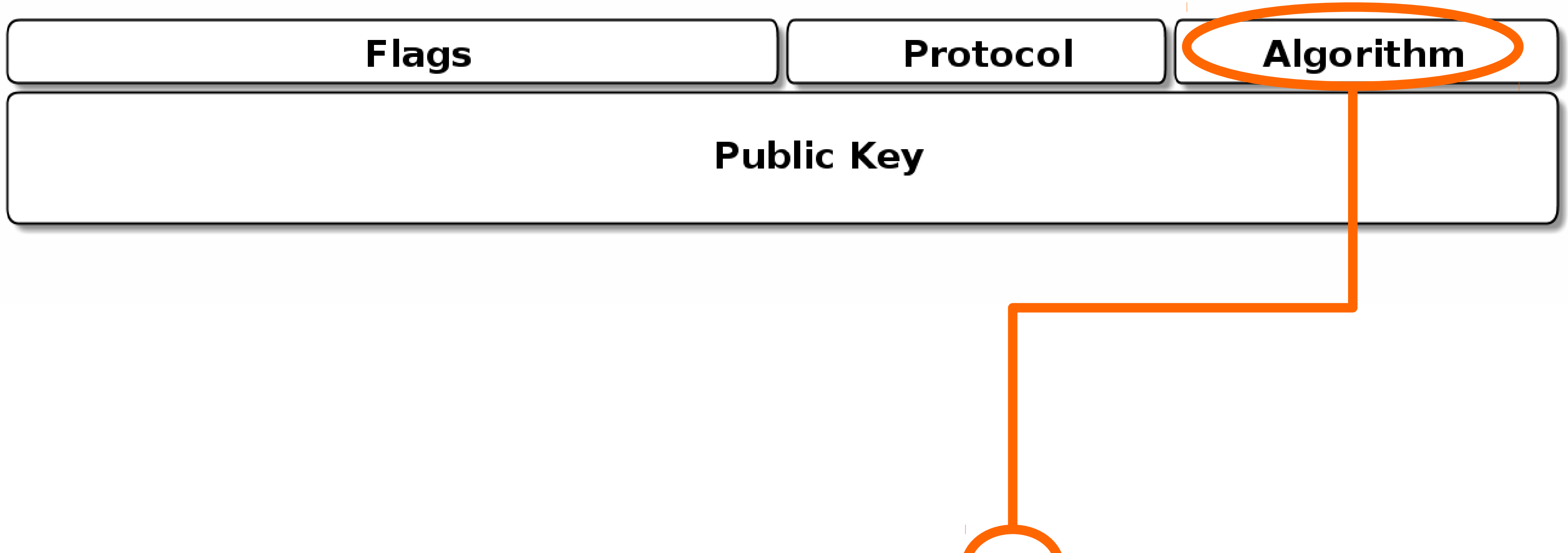
```
tjeb.nl.     3600   IN DNSKEY    256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP761AFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
```

# DNSSEC: Get public key for signature



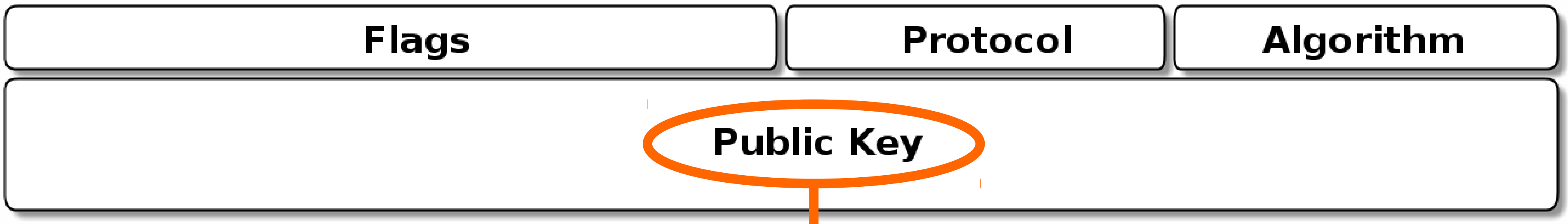
```
tjeb.nl.    3600  IN DNSKEY   256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP76lAFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
```

# DNSSEC: Get public key for signature



```
tjeb.nl.     3600   IN DNSKEY    256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP76lAFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
```

# DNSSEC: Get public key for signature



```
tjeb.nl.     3600  IN DNSKEY    256 3 8
AwEAAee4BKqSMI/wEKdLXQyn+TzOjEMWG5IXy+WRGw+6MiKrbLit60eJ
xNXszf/zR55UUtMqP761AFkFwZgpmUs6ac3pYOTUYRVFjjG1/hnUF1/t
hd9uZLe1E3gwa5m6dcOHaspG5xYsJ2wEBmYj1z1xTh70892PwxVR9R9G
MKh4YyNt
```

# DNSSEC: Have RRSIG and DNSKEY, now what

- Validated signature with key(s)
- Validated keyset with (key-signing) key(s)
- Need to validate that key-signing key too…
- Ask parent

# DNSSEC: Get digest of key at parent

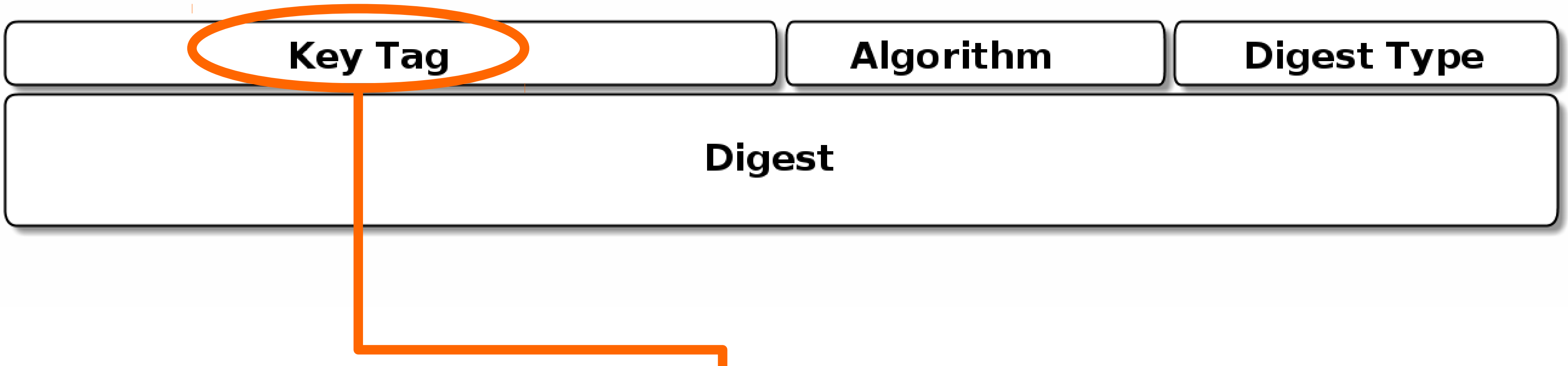
```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      7200   IN DS 17992 8 2
764501411DE58E8618945054A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

***DS = Hash of a child's public key***

# DNSSEC: Get digest of key at parent

```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      7200   IN DS  17992 8 2
764501411DE58E8618945054A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

# DNSSEC: Get digest of key at parent

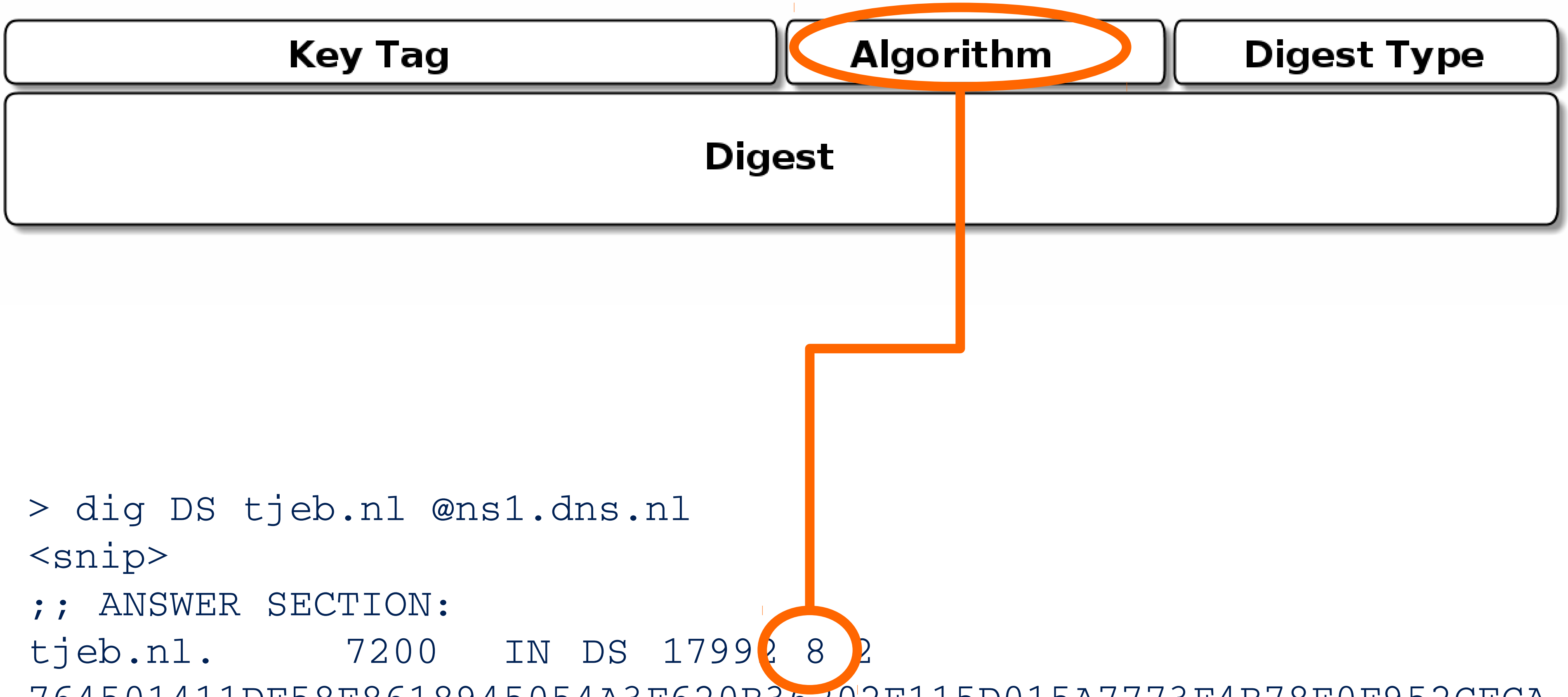


**Key Tag** | **Algorithm** | **Digest Type**

**Digest**

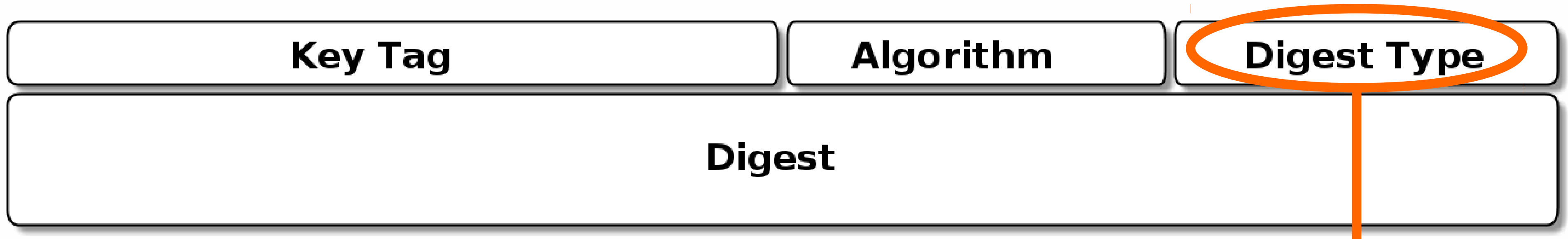
```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      7200   IN  DS 17992 8 2
764501411DE58E86189450 54A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

# DNSSEC: Get digest of key at parent



```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      7200   IN  DS  17992 8 2
764501411DE58E8618945054A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

# DNSSEC: Get digest of key at parent

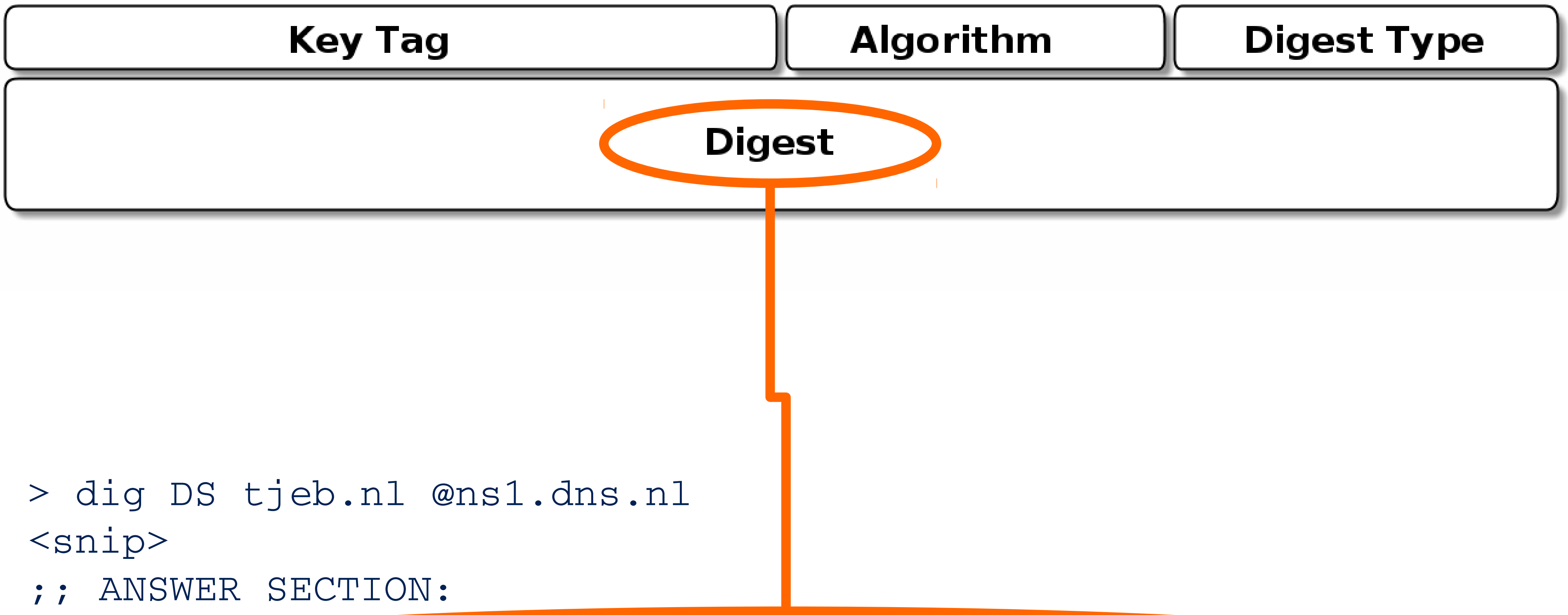


**Key Tag** | **Algorithm** | **Digest Type**

**Digest**

```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.      7200   IN DS  17992 3 2
764501411DE58E8618945054A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

# DNSSEC: Get digest of key at parent



Key Tag | Algorithm | Digest Type

Digest

```
> dig DS tjeb.nl @ns1.dns.nl
<snip>
;; ANSWER SECTION:
tjeb.nl.        7200   IN  DS  17992 8 2
764501411DE58E86189450 54A3F620B36202E115D015A7773F4B78E0F952CECA
<snip>
```

# Turtles all the way up

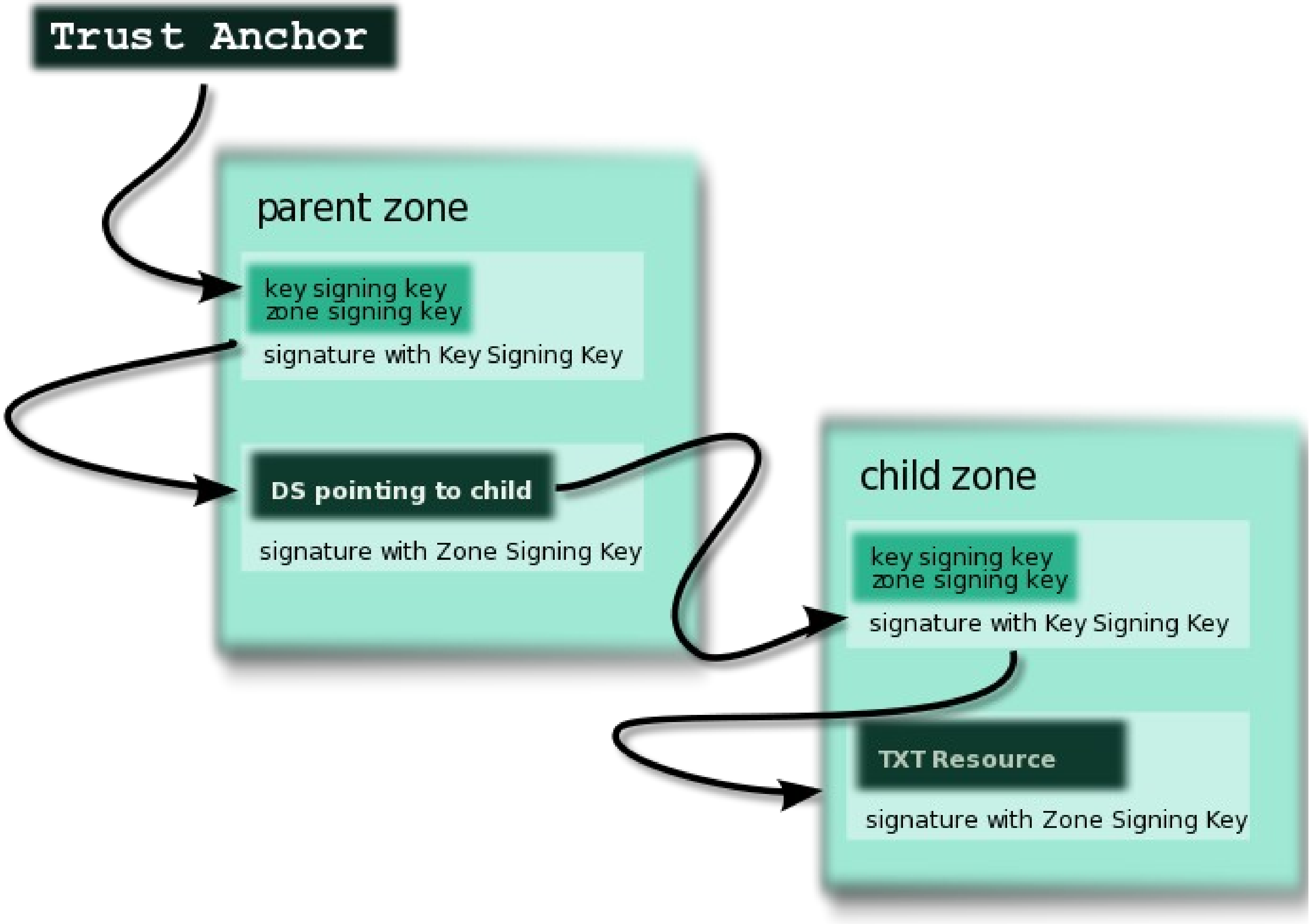
To check RR: check RRSIG
To check RRSIG: check DNSKEY
To check DNSKEY: check DS at parent
To check DS: check RRSIG of DS
To check RRSIG of DS, check DNSKEY
Etc.

That is, up to a 'trust anchor':
a preconfigured trusted public key

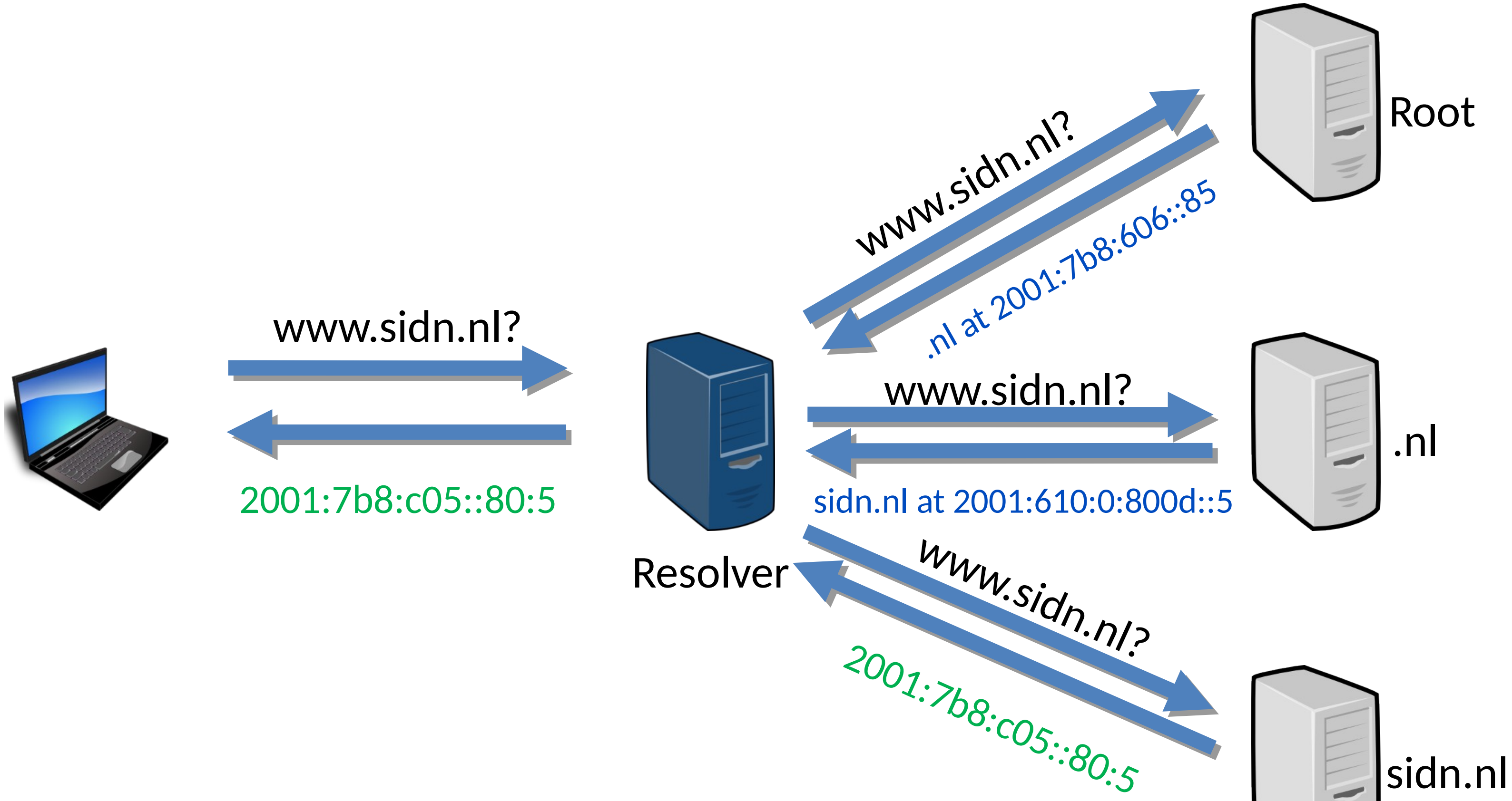
***Chain of trust: the set of signed elements from an anchor to the data***

# DNSSEC validation

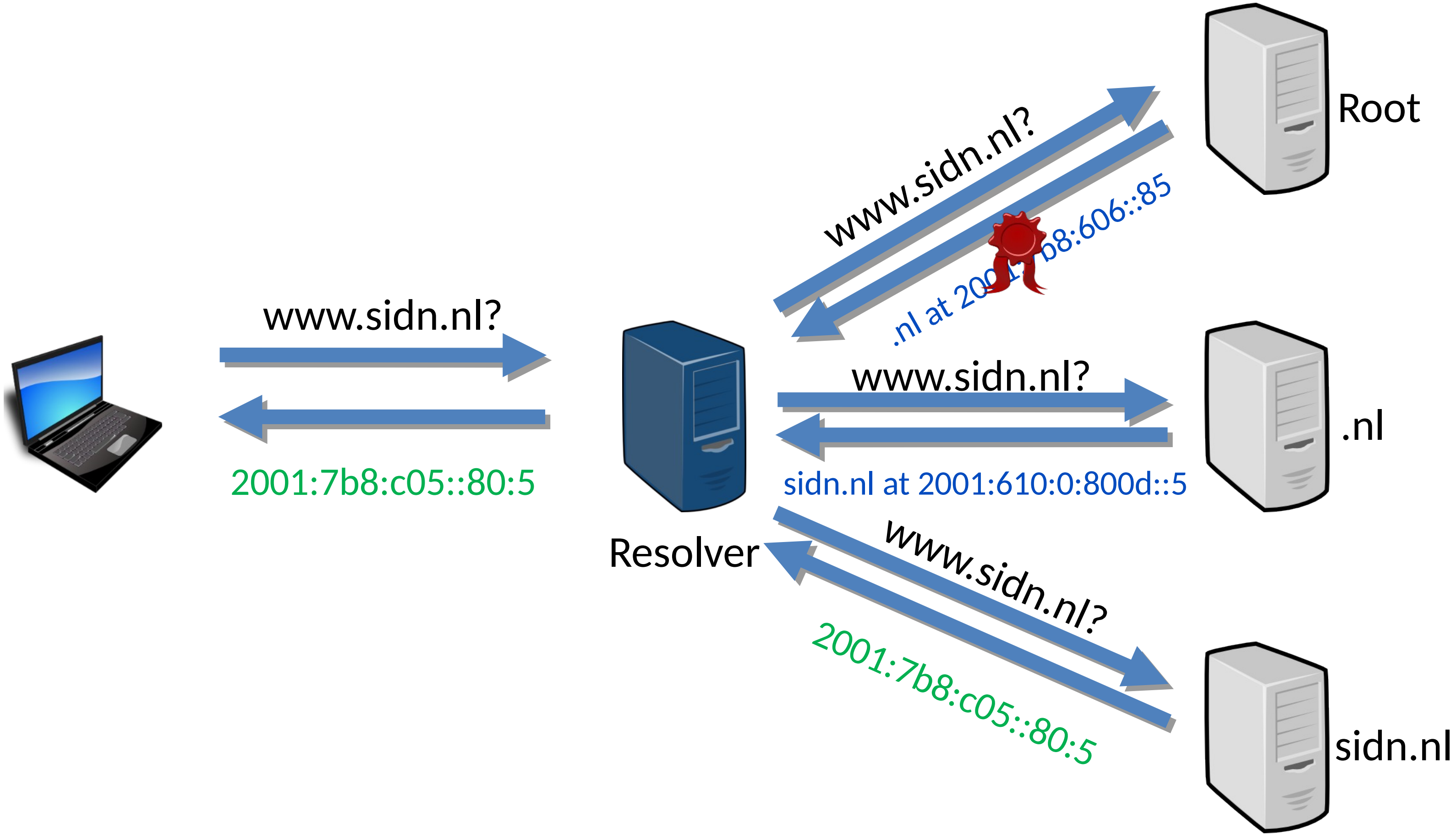
In practice a validator goes top-down:



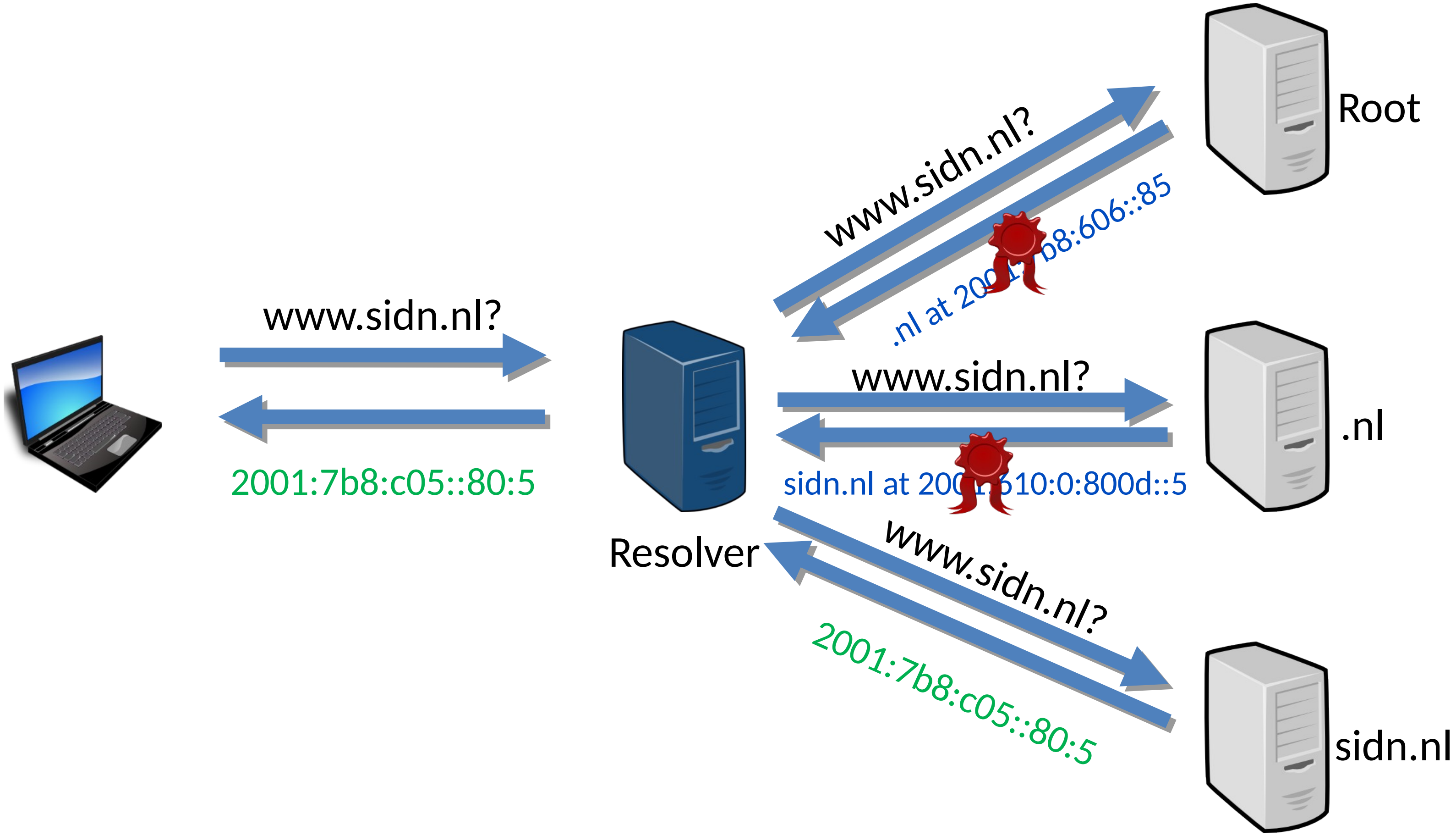
# DNSSEC on DNS recap



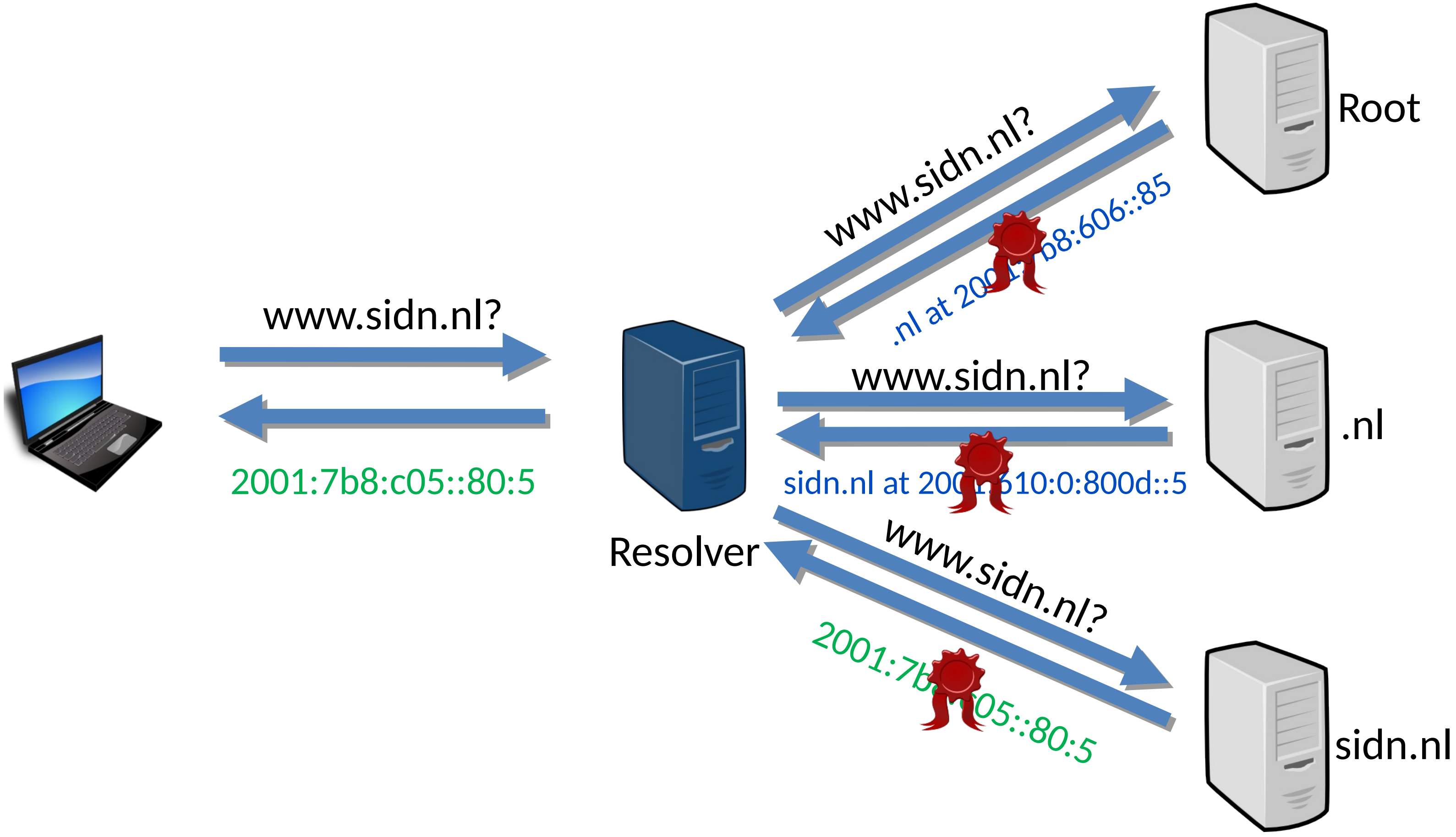
# DNSSEC on DNS recap



# DNSSEC on DNS recap



# DNSSEC on DNS recap



www.sidn.nl?

2001:7b8:c05::80:5

Resolver

www.sidn.nl?

.nl at 2001:7b8:606::85

Root

www.sidn.nl?

sidn.nl at 2001:610:0:800d::5

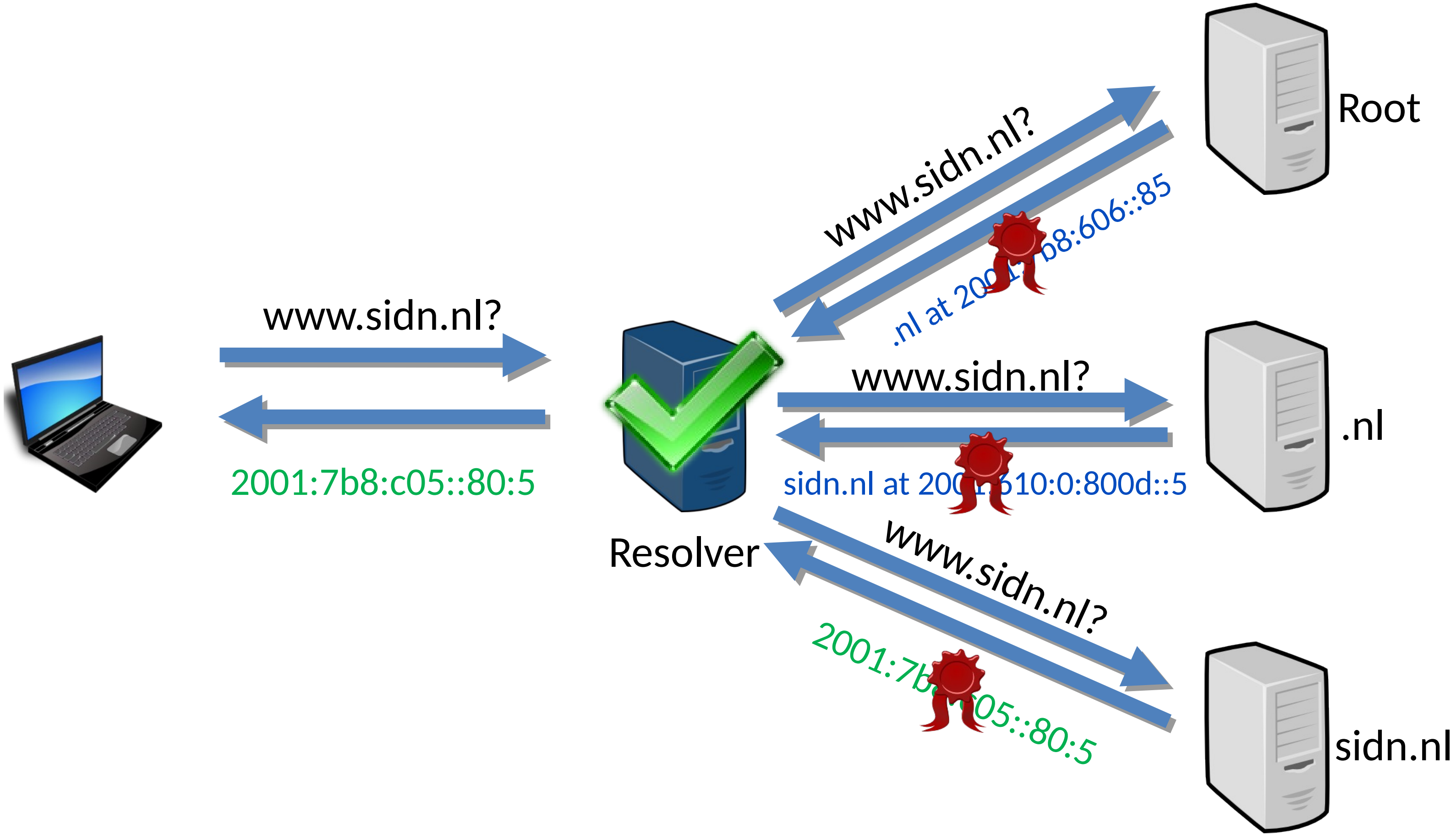
.nl

www.sidn.nl?

2001:7b8:c05::80:5

sidn.nl

# DNSSEC on DNS recap



# DNSSEC: Denial of existence

Problem: How to sign data that does not exist?

Normally you'd sign the RRsets of data, but if the name doesn't exist (NXDOMAIN) or there is no data of the queried type (NOERROR/NODATA), what can you sign?

Already decided that signing the packets was not an option.

Need to sign the 'holes' in the data.

# DNSSEC: NSEC

NSEC records 'cover' the spots between existing names in a zone,
by claiming 'there is no data between these names'.

NSEC records also have a list of types that do exist at their **exact** name

Say you have a zone with the names

`a.zone.` (with an address RR)
`b.zone.` (with a TXT RR)
`c.zone.` (with a TXT RR)

When signing, add nsec records

```
a.zone. NSEC b.zone. A
b.zone. NSEC c.zone. TXT
c.zone. NSEC a.zone. TXT
```

# DNSSEC: Get digest of key at parent

```
> dig +dnssec nosuchdomain.tjeb.nl. @ns.tjeb.nl
<snip>
;; ANSWER SECTION:
newsubdomain.tjeb.nl.  3600   IN NSEC   ns.tjeb.nl. NS RRSIG NSEC
<snip>
```

***NSEC: Shows what data exists and what does not***

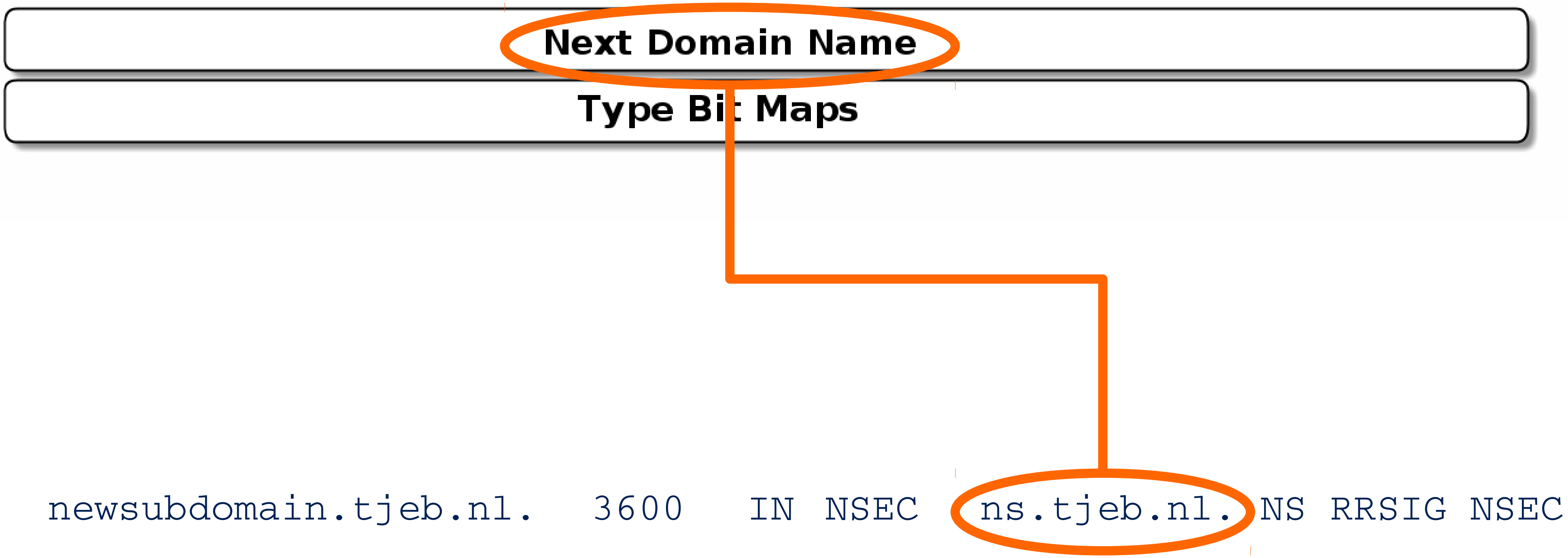
# DNSSEC: NSEC example

**Next Domain Name**

**Type Bit Maps**

```
newsubdomain.tjeb.nl.   3600   IN  NSEC   ns.tjeb.nl. NS RRSIG NSEC
```

# DNSSEC: NSEC example

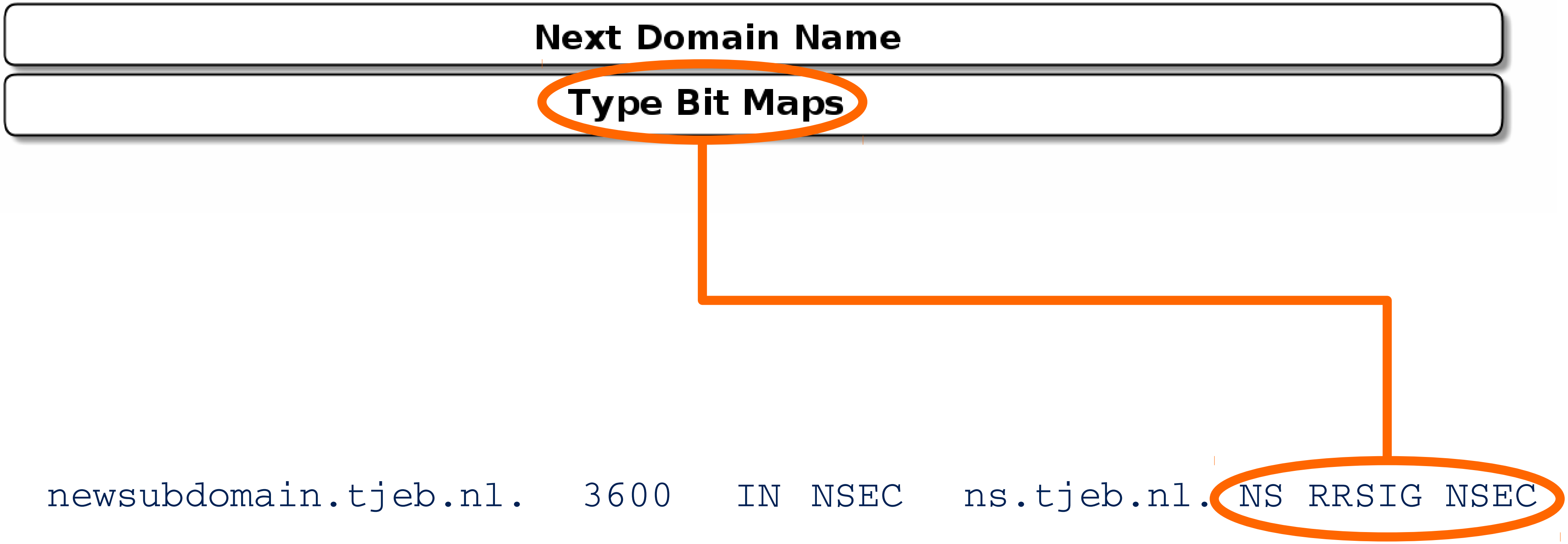


**Next Domain Name**

**Type Bit Maps**

```
newsubdomain.tjeb.nl.   3600   IN  NSEC  ns.tjeb.nl. NS RRSIG NSEC
```

# DNSSEC: NSEC example



**Next Domain Name**

**Type Bit Maps**

```
newsubdomain.tjeb.nl.   3600   IN  NSEC   ns.tjeb.nl. NS RRSIG NSEC
```

# DNSSEC: NSEC gives new problem

If you have a record that shows you next name, you can 'walk' the zone, and get all its contents:

```
> ldns-walk example.nl.
example.nl.   example.nl. A NS SOA TXT AAAA RRSIG NSEC DNSKEY TYPE65534
*.example.nl. MX TXT RRSIG NSEC
*._domainkey.example.nl. TXT RRSIG NSEC
alpha.example.nl. TXT RRSIG NSEC
bravo.example.nl. TXT RRSIG NSEC
delta.example.nl. TXT RRSIG NSEC
echo.example.nl. TXT RRSIG NSEC
www.example.nl. A AAAA RRSIG NSEC
```

# DNSSEC: NSEC3

NSEC3 (RFC 5155) was defined in 2008 to add two new things:

- Protection from zone walking
- Optional signing of delegation

NSEC3 records hash their names and their next names, resolvers hash the queried name and check if the hash falls between the hashed names.

# DNSSEC: NSEC3

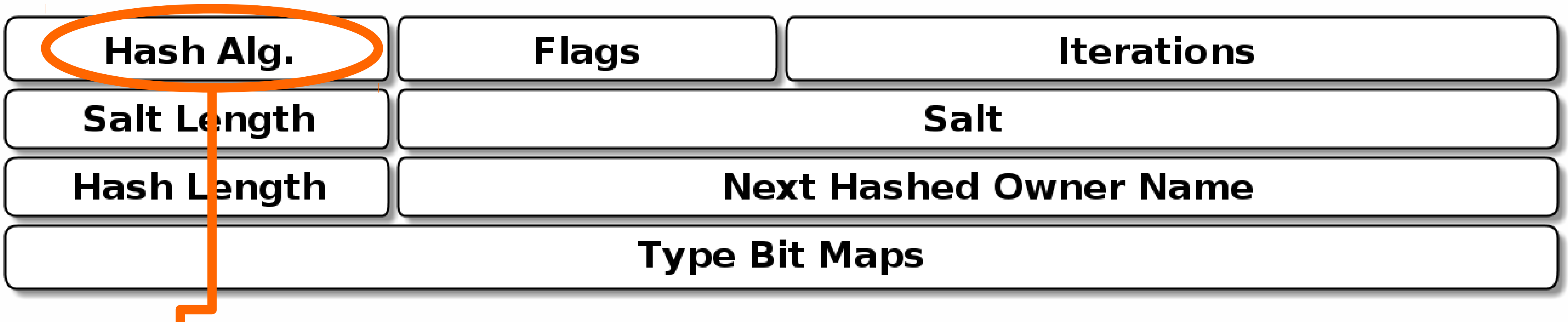
```
> dig +dnssec nosuchdomain.nsec3.tjeb.nl @ns.tjeb.nl
<snip>
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
<snip>
```

# DNSSEC: NSEC3

<table>
<tr><td>Hash Alg.</td><td>Flags</td><td>Iterations</td></tr>
<tr><td>Salt Length</td><td colspan="2">Salt</td></tr>
<tr><td>Hash Length</td><td colspan="2">Next Hashed Owner Name</td></tr>
<tr><td colspan="3">Type Bit Maps</td></tr>
</table>

```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

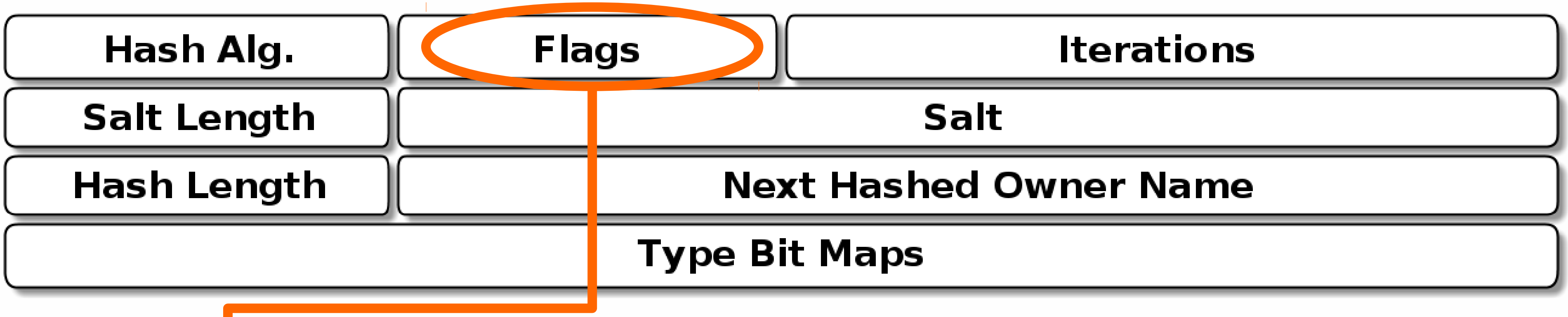
# DNSSEC: NSEC3



| | | |
|---|---|---|
| Hash Alg. | Flags | Iterations |
| Salt Length | Salt | |
| Hash Length | Next Hashed Owner Name | |
| Type Bit Maps | | |

```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3



**Hash Alg.** | **Flags** | **Iterations**

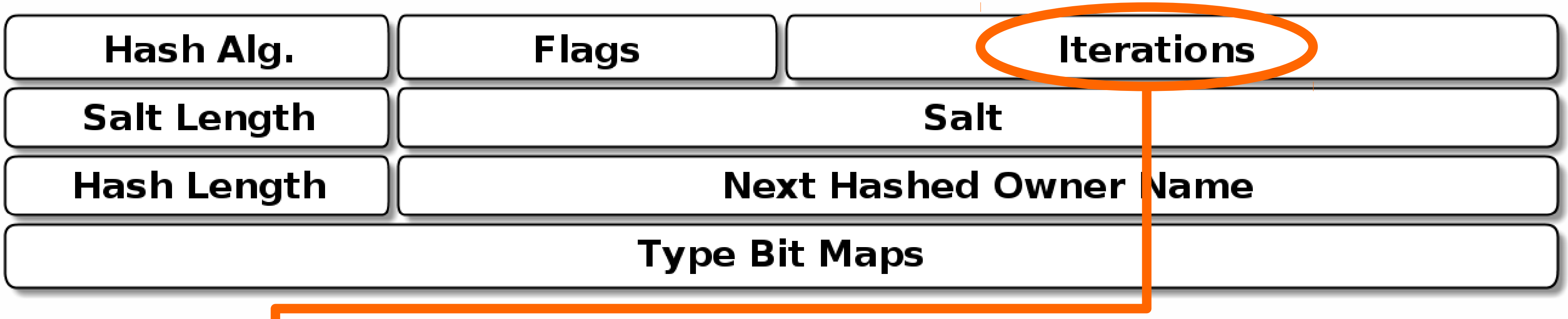
**Salt Length** | **Salt**

**Hash Length** | **Next Hashed Owner Name**

**Type Bit Maps**

```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3



**Hash Alg.** | **Flags** | **Iterations**

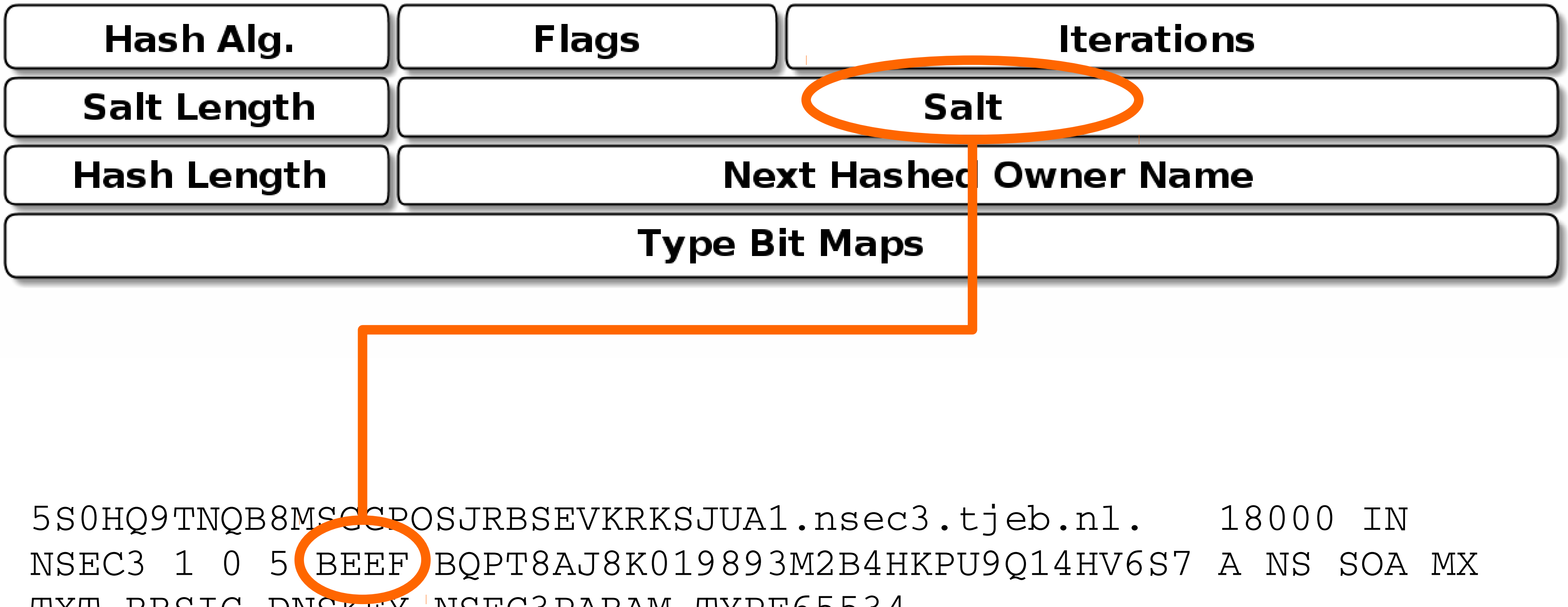
**Salt Length** | **Salt**

**Hash Length** | **Next Hashed Owner Name**

**Type Bit Maps**

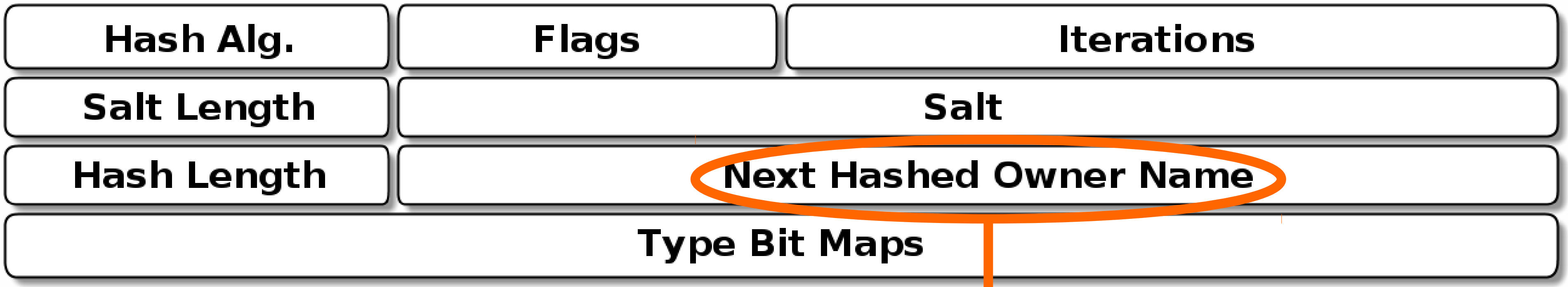
```
5S0HQ9TNQB3MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3



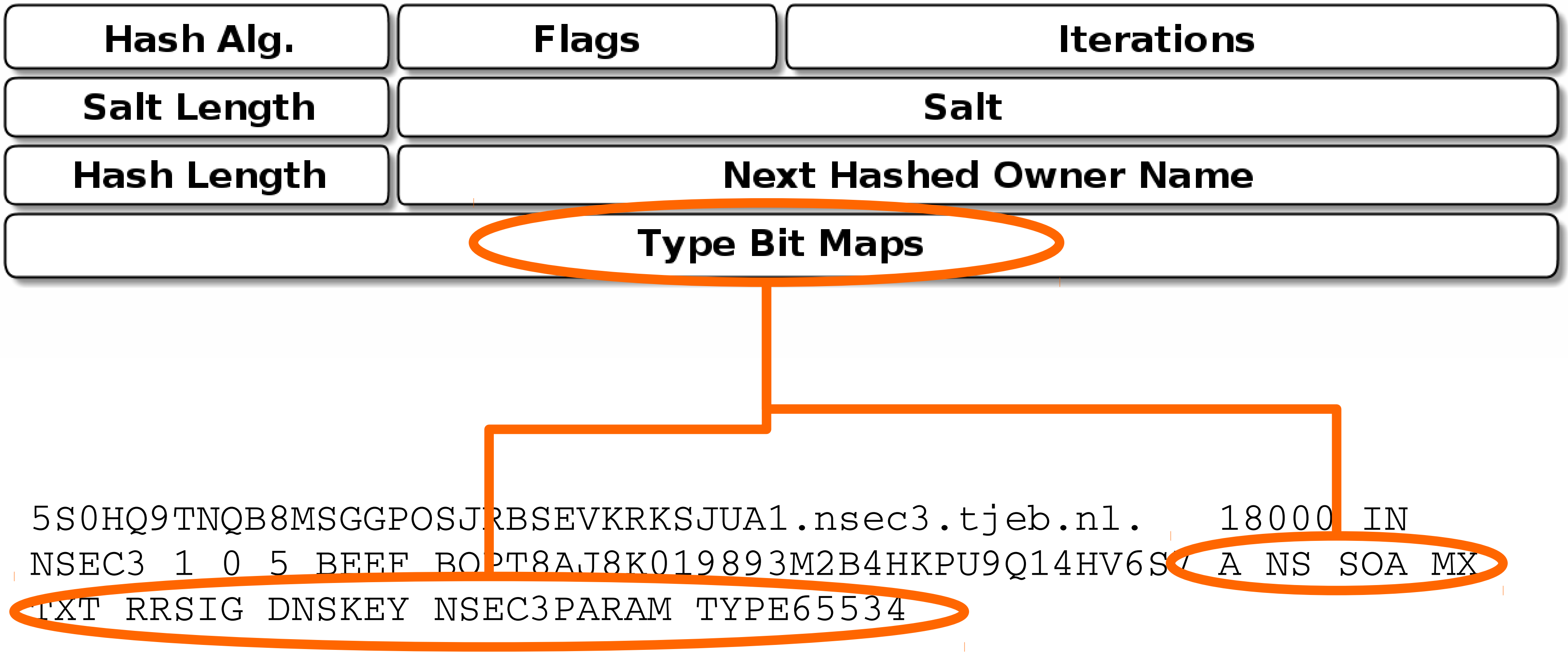
```
5S0HQ9TNQB8MSCCPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.    18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3



```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.    18000 IN
NSEC3 1 0 5 BEEF BOPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3



```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.   18000 IN
NSEC3 1 0 5 BEEF BOPT8AJ8K019893M2B4HKPU9Q14HV6SV A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

# DNSSEC: NSEC3

Define H(x) to be the hash of x using the Hash Algorithm selected by the NSEC3 RR, k to be the number of Iterations, and || to indicate concatenation. Then define:

```
IH(salt, x, 0) = H(x || salt)
```

and

```
IH(salt, x, k) = H(IH(salt, x, k-1) || salt), if k > 0
```

Then the calculated hash of an owner name is

```
IH(salt, owner name, iterations)
```

# DNSSEC: NSEC3

Apply that to example:

nosuchdomain.tjeb.nl with 5 iterations of SHA-1 with salt BEEF gives

**`aql5q4nkstddqs1e3fb5us22lu8bnfc9`**

We got:

```
5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1.nsec3.tjeb.nl.    18000 IN
NSEC3 1 0 5 BEEF BQPT8AJ8K019893M2B4HKPU9Q14HV6S7 A NS SOA MX
TXT RRSIG DNSKEY NSEC3PARAM TYPE65534
```

So we need to check whether

**`5S0HQ9TNQB8MSGGPOSJRBSEVKRKSJUA1 <`**
**`aql5q4nkstddqs1e3fb5us22lu8bnfc9 <`**
**`BQPT8AJ8K019893M2B4HKPU9Q14HV6S7`**

(case-insensitive)

**Conclusion: Existence denied.**

# And that is DNSSEC!

- Pre-defined trust anchor(s)
- Signatures in RRSIGs
- Public keys in DNSKEY sets
- Key(s) of child in signed DS sets
- Authenticated denial of existence done through NSEC or NSEC3

# Bonus content

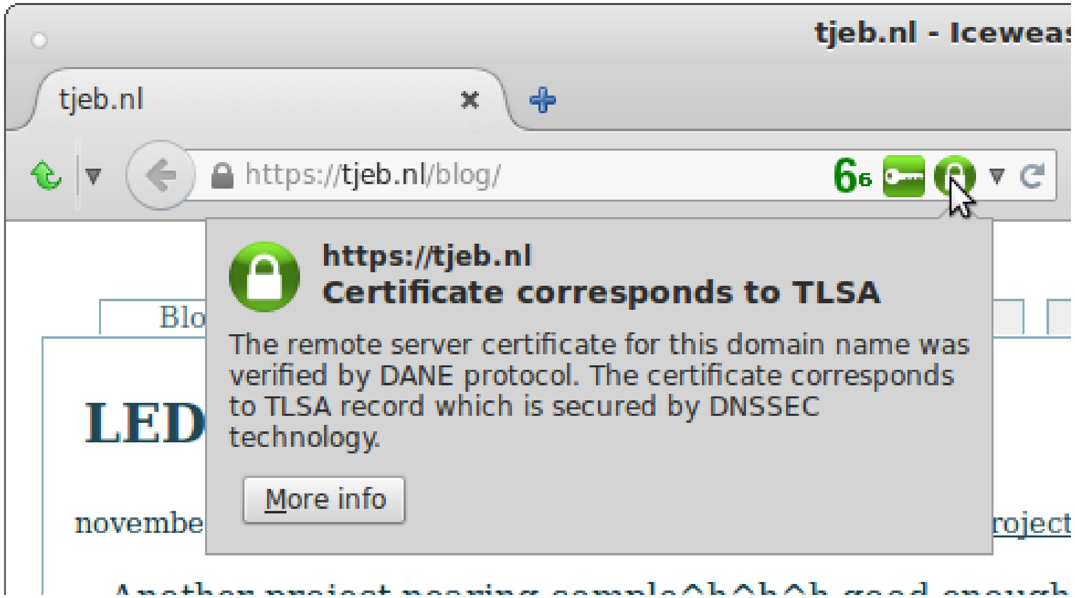
(if there is time)

# DNSSEC as base for other security protocols

- DANE (RFC 6698): connects X.509 (known mostly from https) to DNS(SEC)
  - adds to CA trust
  - allows for self-signed certificates

- In browser (with plugin; no native support)

- Mail Transfer Agents
  - native support in Postfix (2.11)
  - Experimental support in Exim (4.85)



**https://tjeb.nl**
**Certificate corresponds to TLSA**

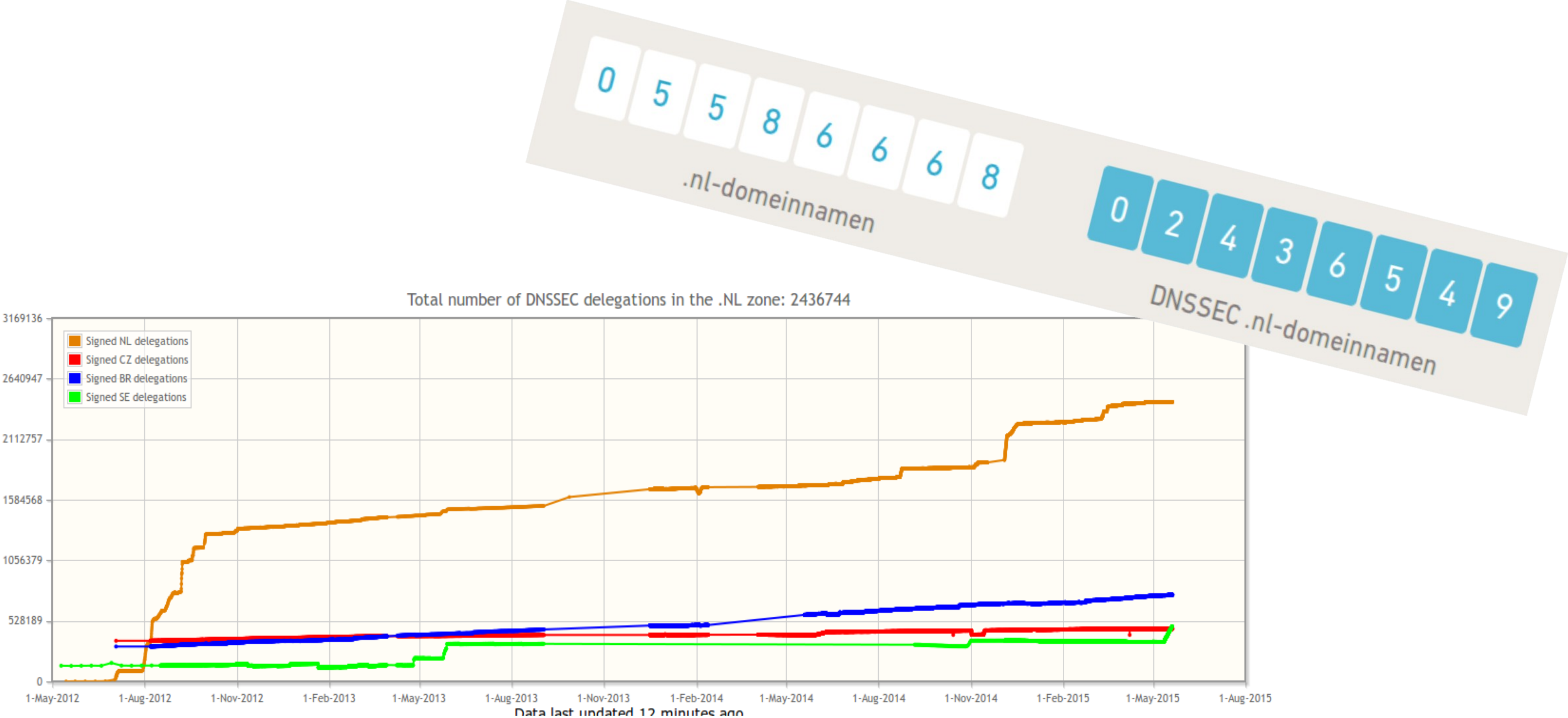
The remote server certificate for this domain name was verified by DANE protocol. The certificate corresponds to TLSA record which is secured by DNSSEC technology.

# DNSSEC as base for other security protocols

- SSHFP (RFC 4255): can be used as host key check in SSH

- Native support in OpenSSH
  - but disabled by default

# DNSSEC in .nl: some numbers



0 5 5 8 6 6 6 8
.nl-domeinnamen

0 2 4 3 6 5 4 9
DNSSEC .nl-domeinnamen

# DNSSEC Information (dutch)

- http://www.dnssec.nl
- http://www.dnsseccursus.nl

SIDN DNSSEC cursus
Klik hier voor de Nederlandse versie

SIDN DNSSEC course
Click here for the English version

# Questions?

Jelte Jansen

Research Engineer

Jelte.jansen@sidn.nl

🐦@twitjeb

www.sidnlabs.nl

SIDN Labs, het research- en developmentteam van SIDN

SIDN Labs, is het R&D-team van SIDN. Wij zijn verantwoordelijk voor het ontwikkelen van nieuwe internettechnieken en –systemen en zijn daarbij actief op drie gebieden: DNS-schaalbaarheid en -veiligheid, 'DNS big data' en privacy- en identitybeheer. Op deze website bloggen wij over onze projecten. Meer over SIDN Labs.

WEBLOG

Firefox and the mysterious rise of ANY-queries

Moritz Muller | 17 maart 2015 - 15:29 uur

Because of its use for malicious purposes such as DNS amplification attacks, DNS ANY queries have been debated for a long time.

When a client sends a DNS request with this query type to a name server, it expects that the server replies with every records which it has available for the queried domain. For example, in case we send an ANY query for the domain example.nl we receive among others the A, AAAA and NS records, DNSSEC related information and TXT records. In this case, the total response is 2624 bytes long, which is 26 times longer than a DNS query for a single A record (100 bytes long).

TAGS dns-amplification dns record any queries meer ⇕

DNSSEC-statistieken op sidnlabs.nl

Maarten Wullink | 17 februari 2015 - 12:04 uur

Zojuist hebben we een nieuwe pagina toegevoegd aan deze website. Een pagina met DNSSEC-statistieken. Deze toepassing maakt gebruik van data uit het ENTRADA (ENhanced Top-level domain Resilience through Advanced Data Analysis) platform en DRS (Domeinnaamregistratiesysteem).

SIDN LABS-TEAM

Maarten Wullink
Research engineer

Marco Davids
Technisch adviseur

Jelte Jansen
Research engineer

Cristian Hesselman
Manager SIDN Labs

Moritz Müller
Student

LAATSTE TWEETS

SIDN — 25 maart 2015, 11:57
@Hostnet Hartelijk gefeliciteerd met het behalen van deze mijlpaal! - hostnet.nl/nieuws/hostnet… pic.twitter.com/49kSvY5Hwz

SIDN — 25 maart 2015, 10:13
In The.nlyst: Gedwongen websiteblokkades door isp's.