The DNS security mess

D. J. Bernstein

University of Illinois at Chicago;
Technische Universiteit Eindhoven

The Domain Name System

`tue.nl` wants to see `http://www.ru.nl`.

Browser at `tue.nl`

↑ "The web server `www.ru.nl` has IP address 131.174.78.60."

Administrator at `ru.nl`

Now `tue.nl` retrieves web page from IP address 131.174.78.60.

S security mess

ernstein

ty of Illinois at Chicago;
che Universiteit Eindhoven

The Domain Name System

`tue.nl` wants to see
`http://www.ru.nl`.

Browser at `tue.nl`

"The web server
`www.ru.nl`
has IP address
131.174.78.60."

Administrator at `ru.nl`

Now `tue.nl`
retrieves web page from
IP address 131.174.78.60.

Same fo

`tue.nl`
`someone`

Mail c

Adminis

Now `tue`
delivers
IP addre

mess

is at Chicago;
siteit Eindhoven

The Domain Name System

`tue.nl` wants to see
`http://www.ru.nl`.

Browser at `tue.nl`

“The web server
`www.ru.nl`
has IP address
131.174.78.60.”

Administrator at `ru.nl`

Now `tue.nl`
retrieves web page from
IP address 131.174.78.60.

Same for Internet

`tue.nl` has mail t
`someone@ru.nl`.

Mail client at

“The mail s
`ru.n`
has IP ad
192.87.10

Administrator a

Now `tue.nl`
delivers mail to
IP address 192.87.

ago;
hoven

The Domain Name System

`tue.nl` wants to see `http://www.ru.nl`.

Browser at `tue.nl`

“The web server `www.ru.nl` has IP address 131.174.78.60.”

Administrator at `ru.nl`

Now `tue.nl` retrieves web page from IP address 131.174.78.60.

Same for Internet mail.

`tue.nl` has mail to deliver t

`someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for `ru.nl` has IP address 192.87.102.77.”

Administrator at `ru.nl`

Now `tue.nl` delivers mail to IP address 192.87.102.77.

The Domain Name System

`tue.nl` wants to see
`http://www.ru.nl`.

Browser at `tue.nl`

“The web server
`www.ru.nl`
has IP address
131.174.78.60.”

Administrator at `ru.nl`

Now `tue.nl`
retrieves web page from
IP address 131.174.78.60.

Same for Internet mail.

`tue.nl` has mail to deliver to
`someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for
`ru.nl`
has IP address
192.87.102.77.”

Administrator at `ru.nl`

Now `tue.nl`
delivers mail to
IP address 192.87.102.77.

main Name System

wants to see

/`www.ru.nl`.

wser at `tue.nl`

“The web server
`www.ru.nl`
has IP address
131.174.78.60.”

strator at `ru.nl`

e.nl
web page from
ess 131.174.78.60.

Same for Internet mail.

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for
`ru.nl`
has IP address
192.87.102.77.”

Administrator at `ru.nl`

Now `tue.nl`
delivers mail to
IP address 192.87.102.77.

Forging

`tue.nl`

someone

Mail cli

“T

Attack

Now tue
delivers
IP addre
actually

e System

see

nl.

ue.nl

erver

nl

ress

60.”

t ru.nl

from

4.78.60.

Same for Internet mail.

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for `ru.nl` has IP address 192.87.102.77.”

Administrator at `ru.nl`

Now `tue.nl` delivers mail to IP address 192.87.102.77.

Forging DNS pack

`tue.nl` has mail t

`someone@ru.nl`.

Mail client at tu

“The mail serv `ru.nl` has IP addr 204.13.202.7

Attacker anywh

Now `tue.nl` delivers mail to IP address 204.13.

actually the attack

Same for Internet mail.

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for
`ru.nl`
has IP address
192.87.102.77.”

Administrator at `ru.nl`

Now `tue.nl`
delivers mail to
IP address 192.87.102.77.

Forging DNS packets

`tue.nl` has mail to deliver t

`someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for
`ru.nl`
has IP address
204.13.202.78.”

Attacker anywhere on ne

Now `tue.nl`
delivers mail to
IP address 204.13.202.78,
actually the attacker’s mach

Same for Internet mail.

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

"The mail server for `ru.nl` has IP address 192.87.102.77."

Administrator at `ru.nl`

Now `tue.nl` delivers mail to IP address 192.87.102.77.

Forging DNS packets

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

"The mail server for `ru.nl` has IP address 204.13.202.78."

Attacker anywhere on network

Now `tue.nl` delivers mail to IP address 204.13.202.78, actually the attacker's machine.

r Internet mail.

has mail to deliver to

e@ru.nl.

lient at tue.nl

"The mail server for

ru.nl

has IP address

192.87.102.77."

strator at ru.nl

e.nl

mail to

ss 192.87.102.77.

Forging DNS packets

`tue.nl` has mail to deliver to

`someone@ru.nl`.

Mail client at `tue.nl`

"The mail server for

`ru.nl`

has IP address

204.13.202.78."

Attacker anywhere on network

Now `tue.nl`

delivers mail to

IP address 204.13.202.78,

actually the attacker's machine.

How forg

Client se

Attacker

some pa

Attacker

- the na
- the qu
- ≈ the

  so clie

  before
- the qu
- the qu

mail.

o deliver to

tue.nl

erver for
l
ldress
2.77."

t ru.nl

.102.77.

Forging DNS packets

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

"The mail server for
`ru.nl`
has IP address
204.13.202.78."

Attacker anywhere on network

Now `tue.nl`
delivers mail to
IP address 204.13.202.78,
actually the attacker's machine.

How forgery really

Client sends query
Attacker has to re
some parts of the

Attacker must ma

- the name: `ru.n`
- the query type:
- ≈ the query tim
  so client sees for
  before legitimate
- the query UDP
- the query ID.

to

Forging DNS packets

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

“The mail server for
`ru.nl`
has IP address
204.13.202.78.”

Attacker anywhere on network

Now `tue.nl`
delivers mail to
IP address 204.13.202.78,
actually the attacker’s machine.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match
- the name: `ru.nl`.
- the query type: mail. (“M
- $\approx$ the query time,
  so client sees forgery
  before legitimate answer.
- the query UDP port.
- the query ID.

Forging DNS packets

`tue.nl` has mail to deliver to `someone@ru.nl`.

Mail client at `tue.nl`

"The mail server for
`ru.nl`
has IP address
204.13.202.78."

Attacker anywhere on network

Now `tue.nl`
delivers mail to
IP address 204.13.202.78,
actually the attacker's machine.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time, so client sees forgery before legitimate answer.
- the query UDP port.
- the query ID.

DNS packets

has mail to deliver to

e@ru.nl.

ent at tue.nl

The mail server for

ru.nl

has IP address

204.13.202.78."

er anywhere on network

e.nl

mail to

ess 204.13.202.78,

the attacker's machine.

How forgery really works

Client sends query.

Attacker has to repeat

some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time,
  so client sees forgery
  before legitimate answer.
- the query UDP port.
- the query ID.

The hard

for attac

Control

by trigge

Many wa

kets

to deliver to

ue.nl

ver for

ress

78."

here on network

.202.78,

ker's machine.

How forgery really works

Client sends query.

Attacker has to repeat

some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time,
  so client sees forgery
  before legitimate answer.
- the query UDP port.
- the query ID.

The hard way

for attackers to do

Control name, typ

by triggering client

Many ways to do

to

twork

ine.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. (“MX”.)
- $\approx$ the query time,
  so client sees forgery
  before legitimate answer.
- the query UDP port.
- the query ID.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time, so client sees forgery before legitimate answer.
- the query UDP port.
- the query ID.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time,
  so client sees forgery
  before legitimate answer.
- the query UDP port.
- the query ID.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

How forgery really works

Client sends query.
Attacker has to repeat
some parts of the query.

Attacker must match

- the name: `ru.nl`.
- the query type: mail. ("MX".)
- $\approx$ the query time, so client sees forgery before legitimate answer.
- the query UDP port.
- the query ID.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

gery really works

ends query.
r has to repeat
rts of the query.

r must match
me: `ru.nl`.
uery type: mail. ("MX".)
query time,
nt sees forgery
legitimate answer.
uery UDP port.
uery ID.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

The eas
for attac

1. Brea
on the s

2. Usin
sniff net
the clien
Immedia

works

.

peat

query.

tch

l.

mail. ("MX".)

e,

rgery

e answer.

port.

The hard way

for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

The easy way

for attackers to do

1. Break into a co
on the same netwo

2. Using that com
sniff network to se
the client's query.
Immediately forge

MX".)

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

The hard way
for attackers to do this:

Control name, type, time
by triggering client.
Many ways to do this.

Guess port and ID
(or predict them if
they're poorly randomized).
16-bit port, 16-bit ID.

If guess fails, try again.
After analysis, optimization:
this is about as much traffic
as downloading a movie.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

d way
ckers to do this:

name, type, time
ering client.
ays to do this.

ort and ID
ict them if
poorly randomized).
ort, 16-bit ID.

fails, try again.
alysis, optimization:
bout as much traffic
loading a movie.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

Security

Many D
(e.g. que
stop the
but are t
by the e

o this:

e, time
t.
this.

f
domized).
ID.

gain.
imization:
uch traffic
movie.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

Security theater

Many DNS "defen
(e.g. query repetit
stop the hard atta
but are trivially br
by the easy attack

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

<u>Security theater</u>

Many DNS "defenses"
(e.g. query repetition)
stop the hard attack
but are trivially broken
by the easy attack.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

Security theater

Many DNS "defenses"
(e.g. query repetition)
stop the hard attack
but are trivially broken
by the easy attack.

The easy way
for attackers to do this:

1. Break into a computer on the same network.

2. Using that computer, sniff network to see the client's query. Immediately forge answer.

Sometimes skip step 1: the network *is* the attacker. e.g. DNS forgery by hotels, Iranian government, et al.

Security theater

Many DNS "defenses" (e.g. query repetition) stop the hard attack but are trivially broken by the easy attack.

Why don't people realize this? Answer: The hard attack receives much more publicity than the easy attack.

The easy way
for attackers to do this:

1. Break into a computer
on the same network.

2. Using that computer,
sniff network to see
the client's query.
Immediately forge answer.

Sometimes skip step 1:
the network *is* the attacker.
e.g. DNS forgery by hotels,
Iranian government, et al.

Security theater

Many DNS "defenses"
(e.g. query repetition)
stop the hard attack
but are trivially broken
by the easy attack.

Why don't people realize this?
Answer: The hard attack
receives much more publicity
than the easy attack.

Security researchers
can't publish easy attacks.

y way
ckers to do this:

k into a computer
ame network.

g that computer,
work to see
nt's query.
ntely forge answer.

hes skip step 1:
work *is* the attacker.
S forgery by hotels,
government, et al.

Security theater

Many DNS “defenses”
(e.g. query repetition)
stop the hard attack
but are trivially broken
by the easy attack.

Why don't people realize this?
Answer: The hard attack
receives much more publicity
than the easy attack.

Security researchers
can't publish easy attacks.

June 200

“.ORG b
TLD to
DNSSEC
a signific
effort to
for the .
the first
Domain
zone wit
Extensio
the .OR
domain
this need

o this:

omputer

ork.

nputer,

ee

answer.

ep 1:

attacker.

oy hotels,

nt, et al.

Security theater

Many DNS “defenses”
(e.g. query repetition)
stop the hard attack
but are trivially broken
by the easy attack.

Why don’t people realize this?
Answer: The hard attack
receives much more publicity
than the easy attack.

Security researchers
can’t publish easy attacks.

June 2009: excitin

“.ORG becomes th
TLD to sign their
DNSSEC . . . Toda
a significant milest
effort to bolster or
for the .ORG com
the first open gene
Domain to success
zone with Domain
Extensions (DNSS
the .ORG zone is
domain registry to
this needed securit

Security theater

Many DNS “defenses” (e.g. query repetition) stop the hard attack but are trivially broken by the easy attack.

Why don’t people realize this? Answer: The hard attack receives much more publicity than the easy attack.

Security researchers can’t publish easy attacks.

June 2009: exciting news!

“.ORG becomes the first op
TLD to sign their zone with
DNSSEC . . . Today we reac
a significant milestone in ou
effort to bolster online secur
for the .ORG community. W
the first open generic Top-L
Domain to successfully sign
zone with Domain Name Se
Extensions (DNSSEC). To d
the .ORG zone is the largest
domain registry to implemen
this needed security measure

Security theater

Many DNS "defenses" (e.g. query repetition) stop the hard attack but are trivially broken by the easy attack.

Why don't people realize this? Answer: The hard attack receives much more publicity than the easy attack.

Security researchers can't publish easy attacks.

June 2009: exciting news!

".ORG becomes the first open TLD to sign their zone with DNSSEC . . . Today we reached a significant milestone in our effort to bolster online security for the .ORG community. We are the first open generic Top-Level Domain to successfully sign our zone with Domain Name Security Extensions (DNSSEC). To date, the .ORG zone is the largest domain registry to implement this needed security measure."

theater

NS "defenses"
ery repetition)
hard attack
trivially broken
asy attack.

n't people realize this?
The hard attack
much more publicity
easy attack.

researchers
blish easy attacks.

June 2009: exciting news!

".ORG becomes the first open TLD to sign their zone with DNSSEC . . . Today we reached a significant milestone in our effort to bolster online security for the .ORG community. We are the first open generic Top-Level Domain to successfully sign our zone with Domain Name Security Extensions (DNSSEC). To date, the .ORG zone is the largest domain registry to implement this needed security measure."

"What d
.ORG Zo
Signing
of our D
We are r
signing t
within th
This pro
the zone
of the or
integrity

ses”
ion)
ck
oken
.

realize this?
attack
re publicity
ck.

rs
attacks.

June 2009: exciting news!

“.ORG becomes the first open TLD to sign their zone with DNSSEC . . . Today we reached a significant milestone in our effort to bolster online security for the .ORG community. We are the first open generic Top-Level Domain to successfully sign our zone with Domain Name Security Extensions (DNSSEC). To date, the .ORG zone is the largest domain registry to implement this needed security measure.”

“What does it me
.ORG Zone is ‘sig
Signing our zone i
of our DNSSEC te
We are now crypt
signing the author
within the .ORG z
This process adds
the zone, which al
of the origin authe
integrity of data.”

is?

y

June 2009: exciting news!

".ORG becomes the first open TLD to sign their zone with DNSSEC . . . Today we reached a significant milestone in our effort to bolster online security for the .ORG community. We are the first open generic Top-Level Domain to successfully sign our zone with Domain Name Security Extensions (DNSSEC). To date, the .ORG zone is the largest domain registry to implement this needed security measure."

"What does it mean that th
.ORG Zone is 'signed'?
Signing our zone is the first
of our DNSSEC test phase.
We are now cryptographical
signing the authoritative dat
within the .ORG zone file.
This process adds new recor
the zone, which allows verifi
of the origin authenticity an
integrity of data."

June 2009: exciting news!

".ORG becomes the first open TLD to sign their zone with DNSSEC . . . Today we reached a significant milestone in our effort to bolster online security for the .ORG community. We are the first open generic Top-Level Domain to successfully sign our zone with Domain Name Security Extensions (DNSSEC). To date, the .ORG zone is the largest domain registry to implement this needed security measure."

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

09: exciting news!

becomes the first open
sign their zone with
C . . . Today we reached
cant milestone in our
bolster online security
ORG community. We are
open generic Top-Level
to successfully sign our
th Domain Name Security
ons (DNSSEC). To date,
G zone is the largest
registry to implement
ded security measure."

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

Cryptog
Verificat
Integrity

ng news!

he first open
zone with
ay we reached
tone in our
nline security
munity. We are
eric Top-Level
sfully sign our
Name Security
EC). To date,
the largest
implement
ty measure."

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

Cryptography! Au
Verification! Auth
Integrity! Sounds

en

hed
r
ity
/e are
evel
our
curity
late,
t
nt
e."

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

“What does it mean that the .ORG Zone is ‘signed’? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data.”

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

Now I simply configure the new `.org` public key into my DNS software. Because the `.org` servers are signing with DNSSEC, it is no longer possible for attackers to forge data from those servers!

"What does it mean that the .ORG Zone is 'signed'? Signing our zone is the first part of our DNSSEC test phase. We are now cryptographically signing the authoritative data within the .ORG zone file. This process adds new records to the zone, which allows verification of the origin authenticity and integrity of data."

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

Now I simply configure the new `.org` public key into my DNS software. Because the `.org` servers are signing with DNSSEC, it is no longer possible for attackers to forge data from those servers!

... or is it?

does it mean that the
one is 'signed'?

our zone is the first part
NSSEC test phase.
now cryptographically
the authoritative data
ne .ORG zone file.
cess adds new records to
e, which allows verification
rigin authenticity and
of data."

Cryptography! Authority!
Verification! Authenticity!
Integrity! Sounds great!

Now I simply configure
the new `.org` public key
into my DNS software.
Because the `.org` servers
are signing with DNSSEC,
it is no longer possible
for attackers to forge
data from those servers!

... or is it?

Decemb

Let's fin

```
$ dig
d0.org
a0.org
c0.org
b2.org
a2.org
b0.org

$ dig
  b0.
199.19
```

an that the
ned'?
s the first part
est phase.
ographically
itative data
zone file.
new records to
lows verification
enticity and

Cryptography! Authority!
Verification! Authenticity!
Integrity! Sounds great!

Now I simply configure the new `.org` public key into my DNS software. Because the `.org` servers are signing with DNSSEC, it is no longer possible for attackers to forge data from those servers!

... or is it?

December 2015: n

Let's find a `.org` s

```
$ dig +short n
d0.org.afilias
a0.org.afilias
c0.org.afilias
b2.org.afilias
a2.org.afilias
b0.org.afilias

$ dig +short \
  b0.org.afili
199.19.54.1
```

e

part

ly

a

ds to

cation

d

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

Now I simply configure the new `.org` public key into my DNS software. Because the `.org` servers are signing with DNSSEC, it is no longer possible for attackers to forge data from those servers!

... or is it?

December 2015: reality

Let's find a `.org` server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info
c0.org.afilias-nst.info
b2.org.afilias-nst.org.
a2.org.afilias-nst.info
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.or
199.19.54.1
```

Cryptography! Authority! Verification! Authenticity! Integrity! Sounds great!

Now I simply configure the new `.org` public key into my DNS software. Because the `.org` servers are signing with DNSSEC, it is no longer possible for attackers to forge data from those servers!

… or is it?

December 2015: reality

Let's find a `.org` server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info.
c0.org.afilias-nst.info.
b2.org.afilias-nst.org.
a2.org.afilias-nst.info.
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.org
199.19.54.1
```

raphy! Authority!
tion! Authenticity!
r! Sounds great!

mply configure
.org public key
DNS software.
the .org servers
ng with DNSSEC,
onger possible
ckers to forge
m those servers!

it?

December 2015: reality

Let's find a .org server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info.
c0.org.afilias-nst.info.
b2.org.afilias-nst.org.
a2.org.afilias-nst.info.
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.org
199.19.54.1
```

Look up

```
$ dig
  www
  @199
```

Everythi

```
;; AUT
greenp
  864
  ns-
;; ADD
ns-em
  864
  37.
```

thority!
enticity!
great!

igure
lic key
vare.
servers
DNSSEC,
sible
rge
ervers!

December 2015: reality

Let's find a `.org` server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info.
c0.org.afilias-nst.info.
b2.org.afilias-nst.org.
a2.org.afilias-nst.info.
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.org
199.19.54.1
```

Look up greenpea

```
$ dig \
  www.greenpea
  @199.19.54.1
```

Everything looks r

```
;; AUTHORITY S
greenpeace.org
  86400 IN NS
  ns-emea.gree
;; ADDITIONAL
ns-emea.greenp
  86400 IN A
  37.48.104.54
```

December 2015: reality

Let's find a `.org` server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info.
c0.org.afilias-nst.info.
b2.org.afilias-nst.org.
a2.org.afilias-nst.info.
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.org
199.19.54.1
```

Look up `greenpeace.org`:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.or
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

December 2015: reality

Let's find a `.org` server:

```
$ dig +short ns org
d0.org.afilias-nst.org.
a0.org.afilias-nst.info.
c0.org.afilias-nst.info.
b2.org.afilias-nst.org.
a2.org.afilias-nst.info.
b0.org.afilias-nst.org.
```

```
$ dig +short \
  b0.org.afilias-nst.org
199.19.54.1
```

Look up `greenpeace.org`:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.org.
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

er 2015: reality

d a .org server:

```
 +short ns org
g.afilias-nst.org.
g.afilias-nst.info.
g.afilias-nst.info.
g.afilias-nst.org.
g.afilias-nst.info.
g.afilias-nst.org.

 +short \
org.afilias-nst.org
9.54.1
```

Look up greenpeace.org:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.org.
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

Where's

Have to

```
$ dig
  www
  @19
```

Old ans

```
h9p7u
np90u
C3 1
69T6U
 NS S
3PARA

h9p7u
```

reality

server:

```
s org
-nst.org.
-nst.info.
-nst.info.
-nst.org.
-nst.info.
-nst.org.

as-nst.org
```

Look up greenpeace.org:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.org.
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

Where's the crypto

Have to ask for sig

```
$ dig +dnssec
  www.greenpea
  @199.19.54.1
```

Old answer + four

```
h9p7u7tr2u91d0
np90u3h.org. 8
C3 1 1 1 D399E
69T6U801GSG9E1
 NS SOA RRSIG
3PARAM

h9p7u7tr2u91d0
```

.
.
.
g

Look up greenpeace.org:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.org.
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines

```
h9p7u7tr2u91d0v0ljs9l1g
np90u3h.org. 86400 IN N
C3 1 1 1 D399EAAB H9PAR
69T6U801GSG9E1LMITK4DEM
 NS SOA RRSIG DNSKEY NS
3PARAM


h9p7u7tr2u91d0v0ljs9l1g
```

Look up `greenpeace.org`:

```
$ dig \
  www.greenpeace.org \
  @199.19.54.1
```

Everything looks normal:

```
;; AUTHORITY SECTION:
greenpeace.org.
  86400 IN NS
  ns-emea.greenpeace.org.
;; ADDITIONAL SECTION:
ns-emea.greenpeace.org.
  86400 IN A
  37.48.104.54
```

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines:

```
h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN NSE
C3 1 1 1 D399EAAB H9PARR6
69T6U8O1GSG9E1LMITK4DEMOT
 NS SOA RRSIG DNSKEY NSEC
3PARAM


h9p7u7tr2u91d0v0ljs9l1gid
```

greenpeace.org:

```
\
.greenpeace.org \
9.19.54.1
```

ng looks normal:

```
THORITY SECTION:
peace.org.
00 IN NS
emea.greenpeace.org.
DITIONAL SECTION:
ea.greenpeace.org.
00 IN A
48.104.54
```

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines:

```
h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN NSE
C3 1 1 1 D399EAAB H9PARR6
69T6U801GSG9E1LMITK4DEMOT
 NS SOA RRSIG DNSKEY NSEC
3PARAM

h9p7u7tr2u91d0v0ljs9l1gid
```

```
np90u
IG NS
06023
448 o
EZ1/m
UJRUA
Tcziy
ALtRD
80Jdf
YBNTu
Cota

bgca0
qng3p
C3 1
```

ace.org:

ce.org \

normal:

ECTION:

.

npeace.org.

SECTION:

eace.org.

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines:

```
h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN NSE
C3 1 1 1 D399EAAB H9PARR6
69T6U801GSG9E1LMITK4DEMOT
 NS SOA RRSIG DNSKEY NSEC
3PARAM

h9p7u7tr2u91d0v0ljs9l1gid
```

```
np90u3h.org. 8
IG NSEC3 7 2 8
06023715 20151
448 org. OGLya
EZ1/mnvAG8NJ2z
UJRUAfKVCzaWJj
TcziyRmM8iYvBN
ALtRDom1rdpsVD
80Jdf2sbfXmZd1
YBNTujz2NPadBA
Cota iBk=

bgca0g0ug0p6o7
qng3p2f.org. 8
C3 1 1 1 D399E
```

rg.

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines:

```
h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN NSE
C3 1 1 1 D399EAAB H9PARR6
69T6U801GSG9E1LMITK4DEM0T
 NS SOA RRSIG DNSKEY NSEC
3PARAM

h9p7u7tr2u91d0v0ljs9l1gid
```

```
np90u3h.org. 86400 IN R
IG NSEC3 7 2 86400 2016
06023715 20151216013715
448 org. OGLyaFRtHdR6UB
EZ1/mnvAG8NJ2z5nBi5ALpY
UJRUAfKVCzaWJjZ rpgB6Wg
TcziyRmM8iYvBNwzmUxoPzg
ALtRDom1rdpsVDxGveMJu6
80Jdf2sbfXmZd1viiz+RXRv
YBNTujz2NPadBATPlUNr0sb
Cota iBk=

bgca0g0ug0p6o7425emkt9u
qng3p2f.org. 86400 IN N
C3 1 1 1 D399EAAB BGDM7
```

Where's the crypto?

Have to ask for signatures:

```
$ dig +dnssec \
  www.greenpeace.org \
  @199.19.54.1
```

Old answer + four new lines:

```
h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN NSE
C3 1 1 1 D399EAAB H9PARR6
69T6U8O1GSG9E1LMITK4DEM0T
 NS SOA RRSIG DNSKEY NSEC
3PARAM

h9p7u7tr2u91d0v0ljs9l1gid
np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201601
06023715 20151216013715 1
448 org. OGLyaFRtHdR6UBeq
EZ1/mnvAG8NJ2z5nBi5ALpYtE
UJRUAfKVCzaWJjZ rpgB6WgcF
TcziyRmM8iYvBNwzmUxoPzgkv
ALtRDom1rdpsVDxGveMJu6 pE
80Jdf2sbfXmZd1viiz+RXRvNI
YBNTujz2NPadBATPlUNr0sbQj
Cota iBk=

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN NSE
C3 1 1 1 D399EAAB BGDM7MS
```

the crypto?

ask for signatures:

```
+dnssec \
.greenpeace.org \
9.19.54.1
```

wer + four new lines:

```
7tr2u91d0v0ljs9l1gid
3h.org. 86400 IN NSE
1 1 D399EAAB H9PARR6
8O1GSG9E1LMITK4DEMOT
OA RRSIG DNSKEY NSEC
M

7tr2u91d0v0ljs9l1gid
```

```
np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201601
06023715 20151216013715 1
448 org. OGLyaFRtHdR6UBeq
EZ1/mnvAG8NJ2z5nBi5ALpYtE
UJRUAfKVCzaWJjZ rpgB6WgcF
TcziyRmM8iYvBNwzmUxoPzgkv
ALtRDom1rdpsVDxGveMJu6 pE
80Jdf2sbfXmZd1viiz+RXRvNI
YBNTujz2NPadBATPlUNr0sbQj
Cota iBk=

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN NSE
C3 1 1 1 D399EAAB BGDM7MS
```

```
F9V1T
 A RR

bgca0
qng3p
IG NS
30150
448 o
P7sk0
Vn/Q4
0rk7b
fevV8
StsWz
sK+PU
ggs9
```

o?

gnatures:

\
ce.org \

r new lines:

```
v0ljs9l1gid
6400 IN NSE
AAB H9PARR6
LMITK4DEMOT
DNSKEY NSEC


v0ljs9l1gid
```

```
np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201601
06023715 20151216013715 1
448 org. OGLyaFRtHdR6UBeq
EZ1/mnvAG8NJ2z5nBi5ALpYtE
UJRUAfKVCzaWJjZ rpgB6WgcF
TcziyRmM8iYvBNwzmUxoPzgkv
ALtRDom1rdpsVDxGveMJu6 pE
80Jdf2sbfXmZd1viiz+RXRvNI
YBNTujz2NPadBATPlUNr0sbQj
Cota iBk=

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN NSE
C3 1 1 1 D399EAAB BGDM7MS
```

```
F9V1T1JFVI8FA2
 A RRSIG

bgca0g0ug0p6o7
qng3p2f.org. 8
IG NSEC3 7 2 8
30150037 20151
448 org. Wg2ha
P7sk04Y/nSp+sR
Vn/Q4DEXqftVYe
0rk7bZ/K+v0+5m
fevV8t4ZmWrS+N
StsWztJ50oxdmZ
sK+PUKaB6dx2Bo
ggs9 MB0=
```

s:

gid
NSE
R6
NOT
SEC

gid

np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201601
06023715 20151216013715 1
448 org. OGLyaFRtHdR6UBeq
EZ1/mnvAG8NJ2z5nBi5ALpYtE
UJRUAfKVCzaWJjZ rpgB6WgcF
TcziyRmM8iYvBNwzmUxoPzgkv
ALtRDom1rdpsVDxGveMJu6 pE
80Jdf2sbfXmZd1viiz+RXRvNI
YBNTujz2NPadBATPlUNr0sbQj
Cota iBk=

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN NSE
C3 1 1 1 D399EAAB BGDM7MS

F9V1T1JFVI8FA211MH4JD7U
A RRSIG

bgca0g0ug0p6o7425emkt9u
qng3p2f.org. 86400 IN R
IG NSEC3 7 2 86400 2015
30150037 20151209140037
448 org. Wg2ha2mg0DnjiV
P7sk04Y/nSp+sR5uhChRWyz
Vn/Q4DEXqftVYeh v/x7Cmz
0rk7bZ/K+v0+5mOMyao6Fod
fevV8t4ZmWrS+NLjNfx/yl
StsWztJ50oxdmZw1EwOALH/
sK+PUKaB6dx2BoEOiFn1plP
ggs9 MBO=

np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201601
06023715 20151216013715 1
448 org. OGLyaFRtHdR6UBeq
EZ1/mnvAG8NJ2z5nBi5ALpYtE
UJRUAfKVCzaWJjZ rpgB6WgcF
TcziyRmM8iYvBNwzmUxoPzgkv
ALtRDom1rdpsVDxGveMJu6 pE
80Jdf2sbfXmZd1viiz+RXRvNI
YBNTujz2NPadBATPlUNr0sbQj
Cota iBk=

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN NSE
C3 1 1 1 D399EAAB BGDM7MS
F9V1T1JFVI8FA211MH4JD7UJ7
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201512
30150037 20151209140037 1
448 org. Wg2ha2mg0DnjiVNl
P7sk04Y/nSp+sR5uhChRWyzqH
Vn/Q4DEXqftVYeh v/x7Cmz2Q
0rk7bZ/K+v0+5mOMyao6Fod8+
fevV8t4ZmWrS+NLjNfx/yl So
StsWztJ50oxdmZw1Ew0ALH/5g
sK+PUKaB6dx2BoE0iFn1plPSf
ggs9 MB0=

```
3h.org. 86400 IN RRS
EC3 7 2 86400 201601
715 20151216013715 1
rg. OGLyaFRtHdR6UBeq
nvAG8NJ2z5nBi5ALpYtE
fKVCzaWJjZ rpgB6WgcF
RmM8iYvBNwzmUxoPzgkv
om1rdpsVDxGveMJu6 pE
2sbfXmZd1viiz+RXRvNI
jz2NPadBATPlUNr0sbQj
iBk=

g0ug0p6o7425emkt9ue4
2f.org. 86400 IN NSE
1 1 D399EAAB BGDM7MS
```

```
F9V1T1JFVI8FA211MH4JD7UJ7
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201512
30150037 20151209140037 1
448 org. Wg2ha2mg0DnjiVNl
P7sk04Y/nSp+sR5uhChRWyzqH
Vn/Q4DEXqftVYeh v/x7Cmz2Q
0rk7bZ/K+v0+5m0Myao6Fod8+
fevV8t4ZmWrS+NLjNfx/yl So
StsWztJ50oxdmZw1Ew0ALH/5g
sK+PUKaB6dx2BoE0iFn1plPSf
ggs9 MB0=
```

Wow, th
Must be

```
$ tcpdu
  host
```

shows pa
dig send
to the .
receives

See mor

```
$ dig +
  org @
```

Sends 8
receives
totalling

```
6400 IN RRS
6400 201601
216013715 1
FRtHdR6UBeq
5nBi5ALpYtE
Z rpgB6WgcF
wzmUxoPzgkv
xGveMJu6 pE
viiz+RXRvNI
TPlUNr0sbQj


425emkt9ue4
6400 IN NSE
AAB BGDM7MS
```

```
F9V1T1JFVI8FA211MH4JD7UJ7
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201512
30150037 20151209140037 1
448 org. Wg2ha2mg0DnjiVNl
P7sk04Y/nSp+sR5uhChRWyzqH
Vn/Q4DEXqftVYeh v/x7Cmz2Q
0rk7bZ/K+v0+5mOMyao6Fod8+
fevV8t4ZmWrS+NLjNfx/yl So
StsWztJ50oxdmZw1EwOALH/5g
sK+PUKaB6dx2BoE0iFn1plPSf
ggs9 MB0=
```

Wow, that's a lot
Must be strong cr

```
$ tcpdump -n -e
  host 199.19.54
```

shows packet sizes
dig sends 89-byte
to the `.org` DNS
receives 696-byte

See more DNSSEC

```
$ dig +dnssec an
  org @199.19.54
```

Sends 89-byte IP
receives two IP fra
totalling 2362 byte

```
RS
301
5 1
3eq
YtE
gcF
gkv
pE
+NI
5Qj


1e4
NSE
VMS
```

```
F9V1T1JFVI8FA211MH4JD7UJ7
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201512
30150037 20151209140037 1
448 org. Wg2ha2mg0DnjiVNl
P7sk04Y/nSp+sR5uhChRWyzqH
Vn/Q4DEXqftVYeh v/x7Cmz2Q
0rk7bZ/K+v0+5m0Myao6Fod8+
fevV8t4ZmWrS+NLjNfx/yl So
StsWztJ50oxdmZw1Ew0ALH/5g
sK+PUKaB6dx2BoE0iFn1plPSf
ggs9 MB0=
```

Wow, that's a lot of data.
Must be strong cryptograph

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
dig sends 89-byte IP packet
to the `.org` DNS server,
receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet,
receives two IP fragments
totalling 2362 bytes.

```
F9V1T1JFVI8FA211MH4JD7UJ7
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201512
30150037 20151209140037 1
448 org. Wg2ha2mg0DnjiVNl
P7sk04Y/nSp+sR5uhChRWyzqH
Vn/Q4DEXqftVYeh v/x7Cmz2Q
0rk7bZ/K+v0+5m0Myao6Fod8+
fevV8t4ZmWrS+NLjNfx/yl So
StsWztJ50oxdmZw1Ew0ALH/5g
sK+PUKaB6dx2BoE0iFn1plPSf
ggs9 MB0=
```

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet to the `.org` DNS server, receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet, receives two IP fragments totalling 2362 bytes.

1JFVI8FA211MH4JD7UJ7
SIG

g0ug0p6o7425emkt9ue4
2f.org. 86400 IN RRS
EC3 7 2 86400 201512
037 20151209140037 1
rg. Wg2ha2mg0DnjiVNl
4Y/nSp+sR5uhChRWyzqH
DEXqftVYeh v/x7Cmz2Q
Z/K+v0+5mOMyao6Fod8+
t4ZmWrS+NLjNfx/yl So
tJ50oxdmZw1Ew0ALH/5g
KaB6dx2BoE0iFn1plPSf
MB0=

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet to the `.org` DNS server, receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet, receives two IP fragments totalling 2362 bytes.

Interlude

What ha
this data

11MH4JD7UJ7

425emkt9ue4
6400 IN RRS
6400 201512
209140037 1
2mgODnjiVNl
5uhChRWyzqH
h v/x7Cmz2Q
OMyao6Fod8+
LjNfx/yl So
w1EwOALH/5g
EOiFn1plPSf

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet to the `.org` DNS server, receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet, receives two IP fragments totalling 2362 bytes.

Interlude: the atta

What happens if w
this data at somec

J7
e4
RS
12
1
Nl
qH
2Q
8+
So
5g
Sf

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet to the `.org` DNS server, receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet, receives two IP fragments totalling 2362 bytes.

Interlude: the attacker's vie

What happens if we aim this data at someone else?

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet
to the `.org` DNS server,
receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet,
receives two IP fragments
totalling 2362 bytes.

Interlude: the attacker's view

What happens if we aim
this data at someone else?

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet
to the `.org` DNS server,
receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet,
receives two IP fragments
totalling 2362 bytes.

Interlude: the attacker's view

What happens if we aim
this data at someone else?

Wow, that's a lot of data.
Must be strong cryptography!

```
$ tcpdump -n -e \
  host 199.19.54.1 &
```

shows packet sizes:
`dig` sends 89-byte IP packet to the `.org` DNS server, receives 696-byte IP packet.

See more DNSSEC data:

```
$ dig +dnssec any \
  org @199.19.54.1
```

Sends 89-byte IP packet, receives two IP fragments totalling 2362 bytes.

Interlude: the attacker's view

What happens if we aim this data at someone else?

Let's see what DNSSEC can do as an amplification tool for denial-of-service attacks.

hat's a lot of data.
strong cryptography!

```
mp -n -e \
199.19.54.1 &
```

acket sizes:
ds 89-byte IP packet
org DNS server,
696-byte IP packet.

e DNSSEC data:

```
dnssec any \
199.19.54.1
```

9-byte IP packet,
two IP fragments
2362 bytes.

Interlude: the attacker's view

What happens if we aim this data at someone else?

Let's see what DNSSEC can do as an amplification tool for denial-of-service attacks.

Downloa

```
wget -m
  secsp
cd secs
awk '
  /GREE
    spl
    sub
    pri
  }
' ./*--
| sort
```

of data.

yptography!

```
\
.1 &
```

s:

IP packet

server,

IP packet.

C data:

```
y \
.1
```

packet,

gments

es.

Interlude: the attacker's view

What happens if we aim this data at someone else?

Let's see what DNSSEC can do as an amplification tool for denial-of-service attacks.

Download DNSSE

```
wget -m -k -I /
  secspider.cs.u
cd secspider.cs.
awk '
  /GREEN.*GREEN.
    split($0,x,/
    sub(/<\/TD>/
    print x[5]
  }
' ./*--zone.html
| sort -u | wc -
```

y!

t

Interlude: the attacker's view

What happens if we aim
this data at someone else?

Let's see what DNSSEC can do
as an amplification tool for
denial-of-service attacks.

Download DNSSEC zone lis

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Y
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

Interlude: the attacker's view

What happens if we aim this data at someone else?

Let's see what DNSSEC can do as an amplification tool for denial-of-service attacks.

Download DNSSEC zone list:

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Yes/ {
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

e: the attacker's view

appens if we aim
a at someone else?

e what DNSSEC can do
nplification tool for
f-service attacks.

Download DNSSEC zone list:

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Yes/ {
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

Make lis

```
( cd sec
  echo
  | xarg
    /^Zo
      su
      su
    }
    /GRE
      sp
      su
      pr
    }'
) | sort
| awk '-
```

acker's view

ve aim
one else?

ISSEC can do
n tool for
ttacks.

Download DNSSEC zone list:

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Yes/ {
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

Make list of DNSS

```
( cd secspider.c
  echo ./*--zone
  | xargs awk '
    /^Zone <STRO
      sub(/<STRO
      sub(/<\/ST
    }
    /GREEN.*GREE
      split($0,x
      sub(/<\/TD
      print x[5]
    }'
) | sort -k3n \
| awk '{print $1
```

w

n do

Download DNSSEC zone list:

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Yes/ {
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

Make list of DNSSEC names

```
( cd secspider.cs.ucla.ed
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z
      sub(/<STRONG>/,"",z
      sub(/<\/STRONG>/,""
    }
    /GREEN.*GREEN.*GREEN.
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > S
```

Download DNSSEC zone list:

```
wget -m -k -I / \
  secspider.cs.ucla.edu
cd secspider.cs.ucla.edu
awk '
  /GREEN.*GREEN.*GREEN.*Yes/ {
    split($0,x,/<TD>/)
    sub(/<\/TD>/,"",x[5])
    print x[5]
  }
' ./*--zone.html \
| sort -u | wc -l
```

Make list of DNSSEC names:

```
( cd secspider.cs.ucla.edu
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z = $2
      sub(/<STRONG>/,"",z)
      sub(/<\/STRONG>/,"",z)
    }
    /GREEN.*GREEN.*GREEN.*Yes/ {
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5])
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > SERVERS
```

ad DNSSEC zone list:

```
 -k -I / \
ider.cs.ucla.edu
pider.cs.ucla.edu

N.*GREEN.*GREEN.*Yes/ {
it($0,x,/<TD>/)
(/<\/TD>/,"",x[5])
nt x[5]

zone.html \
-u | wc -l
```

Make list of DNSSEC names:

```
( cd secspider.cs.ucla.edu
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z = $2
      sub(/<STRONG>/,"",z)
      sub(/<\/STRONG>/,"",z)
    }
    /GREEN.*GREEN.*GREEN.*Yes/ {
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5])
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > SERVERS
```

For each

estimate

```
while re
do
  dig +c
  +time=
  awk -v
    if
    if
    if
    if
    est
    prin
  }'
done < S
```

C zone list:

```
\
cla.edu
ucla.edu

*GREEN.*Yes/ {
<TD>/)
,"",x[5])



 \
l
```

Make list of DNSSEC names:

```
( cd secspider.cs.ucla.edu
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z = $2
      sub(/<STRONG>/,"",z)
      sub(/<\/STRONG>/,"",z)
    }
    /GREEN.*GREEN.*GREEN.*Yes/ {
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5])
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > SERVERS
```

For each domain:
estimate DNSSEC

```
while read ip z
do
  dig +dnssec +i
  +time=1 any "$
  awk -v "z=$z"
    if ($1 != ";
    if ($2 != "M
    if ($3 != "S
    if ($4 != "r
    est = (22+$5
    print est,ip
  }'
done < SERVERS >
```

t:

es/ {

Make list of DNSSEC names:

```
( cd secspider.cs.ucla.edu
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z = $2
      sub(/<STRONG>/,"",z)
      sub(/<\/STRONG>/,"",z)
    }
    /GREEN.*GREEN.*GREEN.*Yes/ {
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5])
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > SERVERS
```

For each domain: Try query

estimate DNSSEC amplifica

```
while read ip z
do
  dig +dnssec +ignore +tr
  +time=1 any "$z" "@$ip"
  awk -v "z=$z" -v "ip=$i
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") nex
    if ($4 != "rcvd:") ne
    est = (22+$5)/(40+len
    print est,ip,z
  }'
done < SERVERS > AMP
```

Make list of DNSSEC names:

```
( cd secspider.cs.ucla.edu
  echo ./*--zone.html \
  | xargs awk '
    /^Zone <STRONG>/ { z = $2
      sub(/<STRONG>/,"",z)
      sub(/<\/STRONG>/,"",z)
    }
    /GREEN.*GREEN.*GREEN.*Yes/ {
      split($0,x,/<TD>/)
      sub(/<\/TD>/,"",x[5])
      print x[5],z,rand()
    }'
) | sort -k3n \
| awk '{print $1,$2}' > SERVERS
```

For each domain: Try query, estimate DNSSEC amplification.

```
while read ip z
do
  dig +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip" | \
  awk -v "z=$z" -v "ip=$ip" '{
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") next
    if ($4 != "rcvd:") next
    est = (22+$5)/(40+length(z))
    print est,ip,z
  }'
done < SERVERS > AMP
```

t of DNSSEC names:

```
cspider.cs.ucla.edu
./*--zone.html \
gs awk '
one <STRONG>/ { z = $2
ub(/<STRONG>/,"",z)
ub(/<\/STRONG>/,"",z)

EEN.*GREEN.*GREEN.*Yes/ {
plit($0,x,/<TD>/)
ub(/<\/TD>/,"",x[5])
rint x[5],z,rand()

t -k3n \
{print $1,$2}' > SERVERS
```

For each domain: Try query, estimate DNSSEC amplification.

```
while read ip z
do
  dig +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip" | \
  awk -v "z=$z" -v "ip=$ip" '{
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") next
    if ($4 != "rcvd:") next
    est = (22+$5)/(40+length(z))
    print est,ip,z
  }'
done < SERVERS > AMP
```

For each
find dom
maximu

```
sort -n
  if (s
  if ($
  print
  seen[
}' > MA
head -1
wc -l M
```

Output

```
95.6279
2326 MA
```

SEC names:

```
s.ucla.edu
.html \

NG>/ { z = $2
NG>/,"",z)
RONG>/,"",z)

N.*GREEN.*Yes/ {
,/<TD>/)
>/,"",x[5])
,z,rand()

,$2}' > SERVERS
```

For each domain: Try query, estimate DNSSEC amplification.

```
while read ip z
do
  dig +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip" | \
  awk -v "z=$z" -v "ip=$ip" '{
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") next
    if ($4 != "rcvd:") next
    est = (22+$5)/(40+length(z))
    print est,ip,z
  }'
done < SERVERS > AMP
```

For each DNSSEC
find domain estim
maximum DNSSE

```
sort -nr AMP | a
  if (seen[$2])
  if ($1 < 30) n
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time

```
95.6279 156.154.
2326 MAXAMP
```

s:

```
u

= $2
)
,z)

*Yes/ {

])

ERVERS
```

For each domain: Try query, estimate DNSSEC amplification.

```
while read ip z
do
  dig +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip" | \
  awk -v "z=$z" -v "ip=$ip" '{
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") next
    if ($4 != "rcvd:") next
    est = (22+$5)/(40+length(z))
    print est,ip,z
  }'
done < SERVERS > AMP
```

For each DNSSEC server, find domain estimated to ha maximum DNSSEC amplific

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi
2326 MAXAMP
```

For each domain: Try query, estimate DNSSEC amplification.

```
while read ip z
do
  dig +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip" | \
  awk -v "z=$z" -v "ip=$ip" '{
    if ($1 != ";;") next
    if ($2 != "MSG") next
    if ($3 != "SIZE") next
    if ($4 != "rcvd:") next
    est = (22+$5)/(40+length(z))
    print est,ip,z
  }'
done < SERVERS > AMP
```

For each DNSSEC server, find domain estimated to have maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

n domain: Try query,
e DNSSEC amplification.

```
ead ip z

dnssec +ignore +tries=1 \
=1 any "$z" "@$ip" | \
v "z=$z" -v "ip=$ip" '{
($1 != ";;") next
($2 != "MSG") next
($3 != "SIZE") next
($4 != "rcvd:") next
 = (22+$5)/(40+length(z))
nt est,ip,z

SERVERS > AMP
```

For each DNSSEC server,
find domain estimated to have
maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

Can tha
>2000 D
around t
providin
of incom

Try query,
amplification.

```
gnore +tries=1 \
z" "@$ip" | \
-v "ip=$ip" '{
;") next
SG") next
IZE") next
cvd:") next
)/(40+length(z))
,z

 AMP
```

For each DNSSEC server, find domain estimated to have maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

Can that really be
>2000 DNSSEC s
around the Interne
providing >30× a
of incoming UDP

,
tion.

```
ries=1 \
 | \
p" '{
.
t
xt
gth(z))
```

For each DNSSEC server,
find domain estimated to have
maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

Can that really be true?
>2000 DNSSEC servers
around the Internet, each
providing >30× amplificatio
of incoming UDP packets?

For each DNSSEC server,
find domain estimated to have
maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

Can that really be true?
$>$2000 DNSSEC servers
around the Internet, each
providing $>30\times$ amplification
of incoming UDP packets?

For each DNSSEC server,
find domain estimated to have
maximum DNSSEC amplification:

```
sort -nr AMP | awk '{
  if (seen[$2]) next
  if ($1 < 30) next
  print $1,$2,$3
  seen[$2] = 1
}' > MAXAMP
head -1 MAXAMP
wc -l MAXAMP
```

Output (last time I tried it):

```
95.6279 156.154.102.26 fi.
2326 MAXAMP
```

Can that really be true?
>2000 DNSSEC servers
around the Internet, each
providing >30× amplification
of incoming UDP packets?

Let's verify this.

Choose quiet test machines
on two different networks
(without egress filters).

e.g. Sender: 1.2.3.4.
Receiver: 5.6.7.8.

n DNSSEC server,
nain estimated to have
m DNSSEC amplification:

```
r AMP | awk '{
een[$2]) next
1 < 30) next
 $1,$2,$3
$2] = 1
XAMP
 MAXAMP
AXAMP
```

(last time I tried it):

```
 156.154.102.26 fi.
XAMP
```

Can that really be true?
>2000 DNSSEC servers
around the Internet, each
providing >30× amplification
of incoming UDP packets?

Let's verify this.

Choose quiet test machines
on two different networks
(without egress filters).

e.g. Sender: 1.2.3.4.
Receiver: 5.6.7.8.

Run netw
on 1.2.3

On 1.2.3
address
and send

```
ifconfi
  5.6.7
  netma
while r
do
  dig -
  +dnss
  +time
done <
```

C server,
ated to have
C amplification:

```
wk '{
next
ext
```

I tried it):

```
102.26 fi.
```

Can that really be true? >2000 DNSSEC servers around the Internet, each providing >30× amplification of incoming UDP packets?

Let's verify this.

Choose quiet test machines on two different networks (without egress filters).

e.g. Sender: 1.2.3.4.
Receiver: 5.6.7.8.

Run network-traffi
on 1.2.3.4 and 5.6

On 1.2.3.4, set res
address to 5.6.7.8,
and send 1 query/

```
ifconfig eth0:1
  5.6.7.8 \
  netmask 255.25
while read est i
do
  dig -b 5.6.7.8
  +dnssec +ignor
  +time=1 any "$
done < MAXAMP >/
```

ve
ation:

.

Can that really be true?
>2000 DNSSEC servers
around the Internet, each
providing >30× amplification
of incoming UDP packets?

Let's verify this.

Choose quiet test machines
on two different networks
(without egress filters).

e.g. Sender: 1.2.3.4.
Receiver: 5.6.7.8.

Run network-traffic monitors
on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response
address to 5.6.7.8,
and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null
```

Can that really be true?
>2000 DNSSEC servers around the Internet, each providing >30× amplification of incoming UDP packets?

Let's verify this.

Choose quiet test machines on two different networks (without egress filters).

e.g. Sender: 1.2.3.4.
Receiver: 5.6.7.8.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

t really be true?
DNSSEC servers
the Internet, each
g >30× amplification
ning UDP packets?

rify this.

quiet test machines
different networks
t egress filters).

nder: 1.2.3.4.
r: 5.6.7.8.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustain
of actua
in a US-
on typica
at the er

true?
ervers
et, each
mplification
packets?

machines
etworks
ters).

.4.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustained 51× ar
of actual network
in a US-to-Europe
on typical universi
at the end of 201

on

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustained 51$\times$ amplificatio
of actual network traffic
in a US-to-Europe experime
on typical university comput
at the end of 2010.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustained 51$\times$ amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustained 51× amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Run network-traffic monitors on 1.2.3.4 and 5.6.7.8.

On 1.2.3.4, set response address to 5.6.7.8, and send 1 query/second:

```
ifconfig eth0:1 \
  5.6.7.8 \
  netmask 255.255.255.255
while read est ip z
do
  dig -b 5.6.7.8 \
  +dnssec +ignore +tries=1 \
  +time=1 any "$z" "@$ip"
done < MAXAMP >/dev/null 2>&1
```

I sustained 51× amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

work-traffic monitors
.4 and 5.6.7.8.

3.4, set response
to 5.6.7.8,
d 1 query/second:

```
g eth0:1 \
.8 \
sk 255.255.255.255
ead est ip z

b 5.6.7.8 \
ec +ignore +tries=1 \
=1 any "$z" "@$ip"
MAXAMP >/dev/null 2>&1
```

I sustained 51× amplification
of actual network traffic
in a US-to-Europe experiment
on typical university computers
at the end of 2010.

Attacker sending 10Mbps
can trigger 500Mbps flood
from the DNSSEC drone pool,
taking down typical site.

Attacker sending 200Mbps
can trigger 10Gbps flood,
taking down very large site.

Attack c
total DN
Mid-201
Can't ta

c monitors
.7.8.

sponse

second:

```
\

5.255.255
p z

 \
e +tries=1 \
z" "@$ip"
dev/null 2>&1
```

I sustained 51× amplification
of actual network traffic
in a US-to-Europe experiment
on typical university computers
at the end of 2010.

Attacker sending 10Mbps
can trigger 500Mbps flood
from the DNSSEC drone pool,
taking down typical site.

Attacker sending 200Mbps
can trigger 10Gbps flood,
taking down very large site.

Attack capacity is
total DNSSEC ser
Mid-2012 estimate
Can't take down C

s

s

```
=1 \

 2>&1
```

I sustained 51× amplification
of actual network traffic
in a US-to-Europe experiment
on typical university computers
at the end of 2010.

Attacker sending 10Mbps
can trigger 500Mbps flood
from the DNSSEC drone pool,
taking down typical site.

Attacker sending 200Mbps
can trigger 10Gbps flood,
taking down very large site.

Attack capacity is limited by
total DNSSEC server bandw
Mid-2012 estimate: <100Gb
Can't take down Google this

I sustained 51$\times$ amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: $<$100Gbps. Can't take down Google this way.

I sustained 51$\times$ amplification
of actual network traffic
in a US-to-Europe experiment
on typical university computers
at the end of 2010.

Attacker sending 10Mbps
can trigger 500Mbps flood
from the DNSSEC drone pool,
taking down typical site.

Attacker sending 200Mbps
can trigger 10Gbps flood,
taking down very large site.

Attack capacity is limited by
total DNSSEC server bandwidth.
Mid-2012 estimate: $<$100Gbps.
Can't take down Google this way.

Logical attacker response:
Tell people to install DNSSEC.

I sustained 51$\times$ amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: $<$100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

I sustained 51$\times$ amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: $<$100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

2011.12.14 DNSSEC servers: 3393 IP addresses worldwide.

I sustained 51× amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: $<$100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

2011.12.14 DNSSEC servers: 3393 IP addresses worldwide.

2015: No SecSpider downloads???

I sustained 51$\times$ amplification of actual network traffic in a US-to-Europe experiment on typical university computers at the end of 2010.

Attacker sending 10Mbps can trigger 500Mbps flood from the DNSSEC drone pool, taking down typical site.

Attacker sending 200Mbps can trigger 10Gbps flood, taking down very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: $<$100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

2011.12.14 DNSSEC servers: 3393 IP addresses worldwide.

2015: No SecSpider downloads??? Exercise: Collect+publish data.

ed 51× amplification
l network traffic
to-Europe experiment
al university computers
nd of 2010.

r sending 10Mbps
ger 500Mbps flood
e DNSSEC drone pool,
own typical site.

r sending 200Mbps
ger 10Gbps flood,
own very large site.

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: <100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

2011.12.14 DNSSEC servers: 3393 IP addresses worldwide.

2015: No SecSpider downloads??? Exercise: Collect+publish data.

RFC 403
"DNSSE
against

mplification
traffic
experiment
ty computers
0.

10Mbps
ps flood
drone pool,
al site.

200Mbps
s flood,
large site.

Attack capacity is limited by
total DNSSEC server bandwidth.
Mid-2012 estimate: <100Gbps.
Can't take down Google this way.

Logical attacker response:
Tell people to install DNSSEC.

2010.12.24 DNSSEC servers:
2536 IP addresses worldwide.

2011.12.14 DNSSEC servers:
3393 IP addresses worldwide.

2015: No SecSpider downloads???
Exercise: Collect+publish data.

RFC 4033 says
"DNSSEC provide
against denial of s

n

nt

ers

ol,

Attack capacity is limited by
total DNSSEC server bandwidth.
Mid-2012 estimate: $<$100Gbps.
Can’t take down Google this way.

Logical attacker response:
Tell people to install DNSSEC.

2010.12.24 DNSSEC servers:
2536 IP addresses worldwide.

2011.12.14 DNSSEC servers:
3393 IP addresses worldwide.

2015: No SecSpider downloads???
Exercise: Collect+publish data.

RFC 4033 says
“DNSSEC provides no prote
against denial of service atta

Attack capacity is limited by
total DNSSEC server bandwidth.
Mid-2012 estimate: <100Gbps.
Can't take down Google this way.

Logical attacker response:
Tell people to install DNSSEC.

2010.12.24 DNSSEC servers:
2536 IP addresses worldwide.

2011.12.14 DNSSEC servers:
3393 IP addresses worldwide.

2015: No SecSpider downloads???
Exercise: Collect+publish data.

RFC 4033 says
"DNSSEC provides no protection
against denial of service attacks."

Attack capacity is limited by total DNSSEC server bandwidth. Mid-2012 estimate: <100Gbps. Can't take down Google this way.

Logical attacker response: Tell people to install DNSSEC.

2010.12.24 DNSSEC servers: 2536 IP addresses worldwide.

2011.12.14 DNSSEC servers: 3393 IP addresses worldwide.

2015: No SecSpider downloads??? Exercise: Collect+publish data.

RFC 4033 says "DNSSEC provides no protection against denial of service attacks."

RFC 4033 doesn't say "DNSSEC is a pool of remote-controlled attack drones, the worst DDoS amplifier on the Internet."

Attack capacity is limited by
total DNSSEC server bandwidth.
Mid-2012 estimate: <100Gbps.
Can't take down Google this way.

Logical attacker response:
Tell people to install DNSSEC.

2010.12.24 DNSSEC servers:
2536 IP addresses worldwide.

2011.12.14 DNSSEC servers:
3393 IP addresses worldwide.

2015: No SecSpider downloads???
Exercise: Collect+publish data.

RFC 4033 says
"DNSSEC provides no protection
against denial of service attacks."

RFC 4033 doesn't say
"DNSSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exercise: investigate
other types of DoS attacks.
e.g. DNSSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

capacity is limited by
NSSEC server bandwidth.
2 estimate: <100Gbps.
ke down Google this way.

attacker response:
ple to install DNSSEC.

24 DNSSEC servers:
addresses worldwide.

14 DNSSEC servers:
addresses worldwide.

o SecSpider downloads???
: Collect+publish data.

RFC 4033 says
"DNSSEC provides no protection against denial of service attacks."

RFC 4033 doesn't say
"DNSSEC is a pool of remote-controlled attack drones, the worst DDoS amplifier on the Internet."

Exercise: investigate other types of DoS attacks.
e.g. DNSSEC advertising says zero server-CPU-time cost.
How much server CPU time can attackers actually consume?

Back to

Let's pre
care abo
This is r

limited by
ver bandwidth.
e: <100Gbps.
Google this way.

esponse:
all DNSSEC.

EC servers:
worldwide.

EC servers:
worldwide.

er downloads???
-publish data.

RFC 4033 says
“DNSSEC provides no protection
against denial of service attacks.”

RFC 4033 doesn’t say
“DNSSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet.”

Exercise: investigate
other types of DoS attacks.
e.g. DNSSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

Back to integrity

Let’s pretend we d
care about availab
This is not an atta

y
width.
ops.
s way.

EC.

:
e.

:
e.

ads???
ata.

RFC 4033 says
"DNSSEC provides no protection
against denial of service attacks."

RFC 4033 doesn't say
"DNSSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exercise: investigate
other types of DoS attacks.
e.g. DNSSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

Back to integrity

Let's pretend we don't
care about availability.
This is not an attack:

RFC 4033 says
"DNSSEC provides no protection against denial of service attacks."

RFC 4033 doesn't say
"DNSSEC is a pool of remote-controlled attack drones, the worst DDoS amplifier on the Internet."

Exercise: investigate other types of DoS attacks.
e.g. DNSSEC advertising says zero server-CPU-time cost.
How much server CPU time can attackers actually consume?

Back to integrity

Let's pretend we don't care about availability.
This is not an attack:

33 says
EC provides no protection
denial of service attacks."

33 doesn't say
EC is a pool of
controlled attack drones,
st DDoS amplifier
nternet."

: investigate
pes of DoS attacks.
SSEC advertising says
ver-CPU-time cost.
ch server CPU time
ckers actually consume?

Back to integrity

Let's pretend we don't
care about availability.
This is not an attack:

All we ca

s no protection
ervice attacks."

say
ol of
attack drones,
mplifier

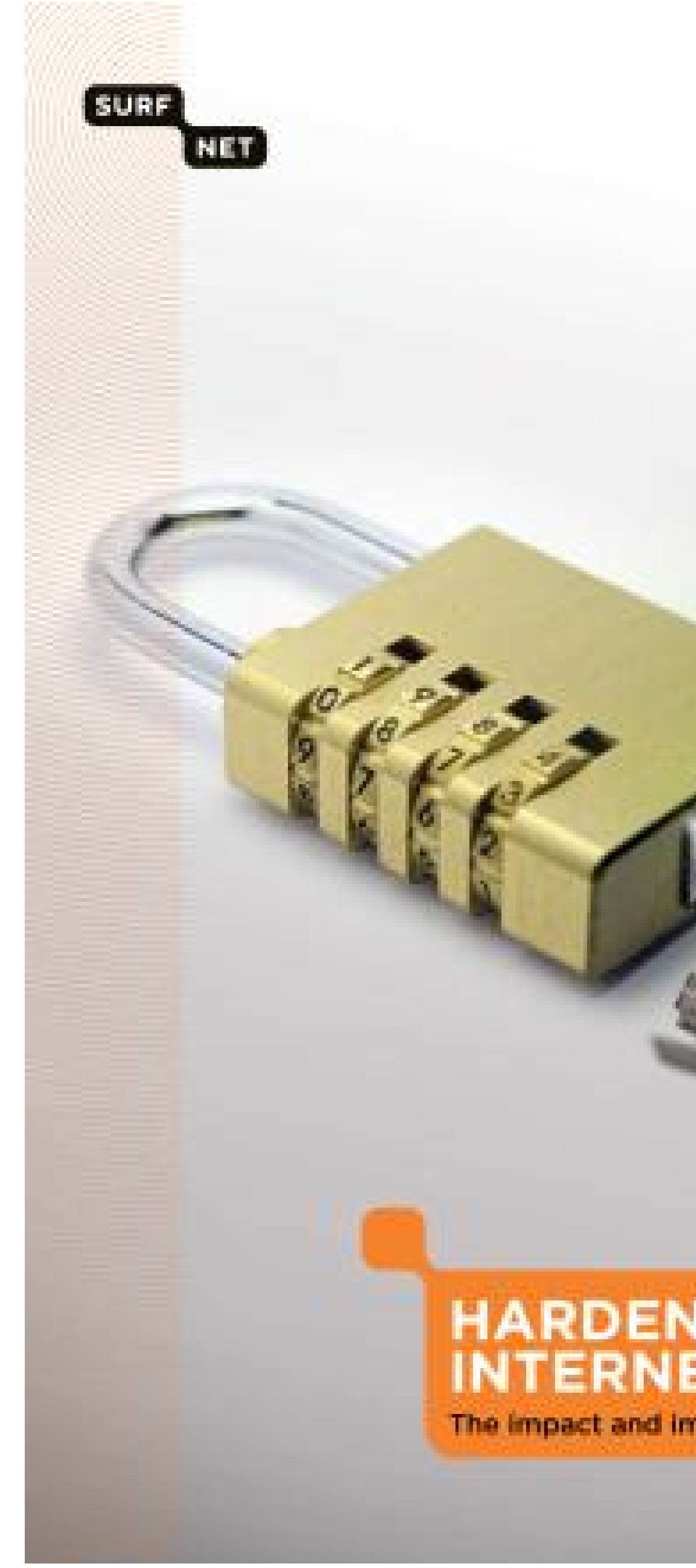
ate
S attacks.
ertising says
ime cost.
CPU time
ally consume?

Back to integrity

Let's pretend we don't care about availability.

This is not an attack:

All we care about



ection
acks.”

ones,

ys

me?

Back to integrity

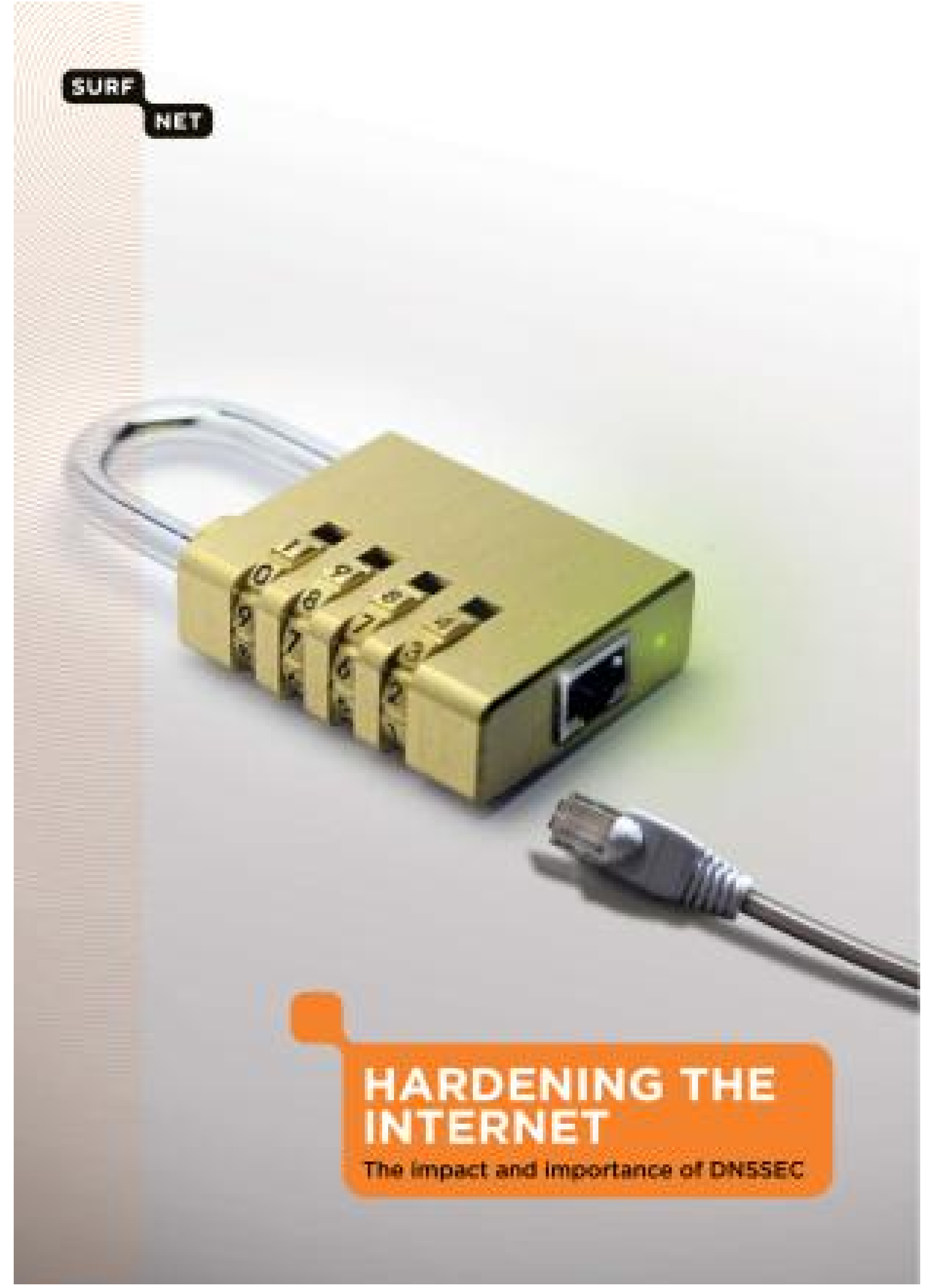
Let’s pretend we don’t care about availability.
This is not an attack:

All we care about is integrity

<u>Back to integrity</u>

Let's pretend we don't care about availability. This is not an attack:

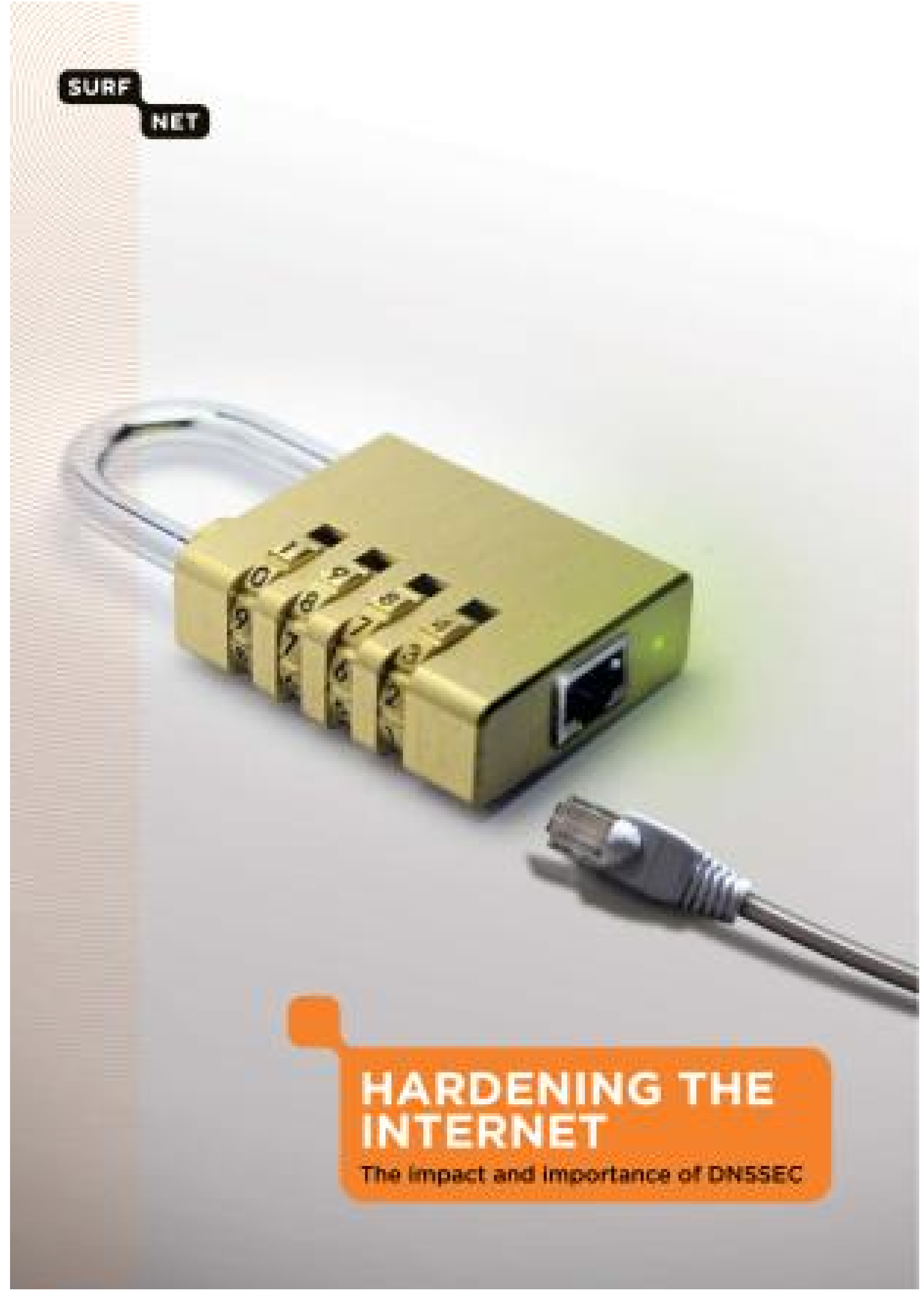
All we care about is integrity:



integrity

etend we don't
out availability.
not an attack:

All we care about is integrity:



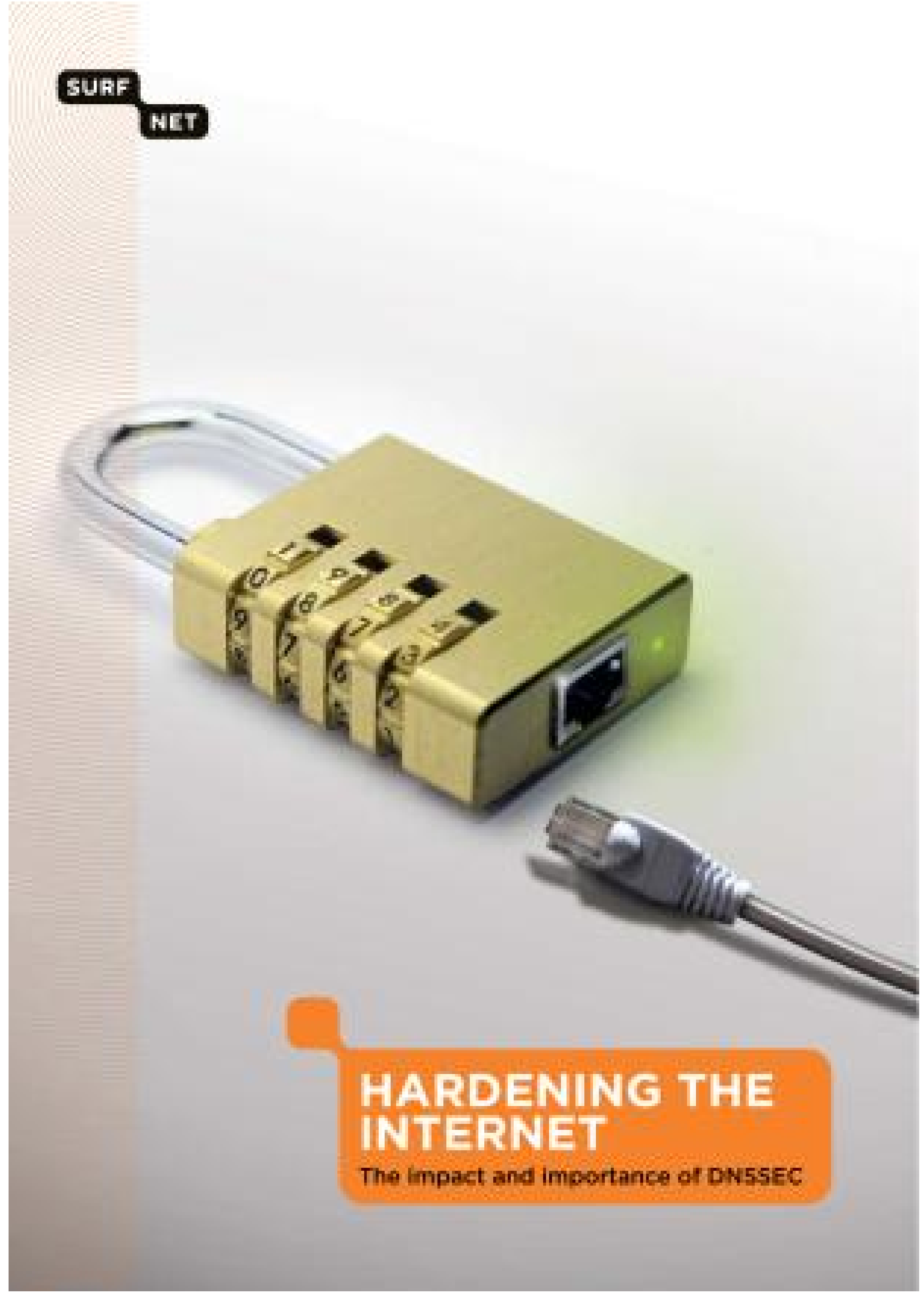
The .or
are 1024

2003: Sh
conclude
was alre
large cor
$10 milli
$120 mi

2003: R
recomme
2048-bit
of this d
made th

lon't
bility.
ack:

All we care about is integrity:

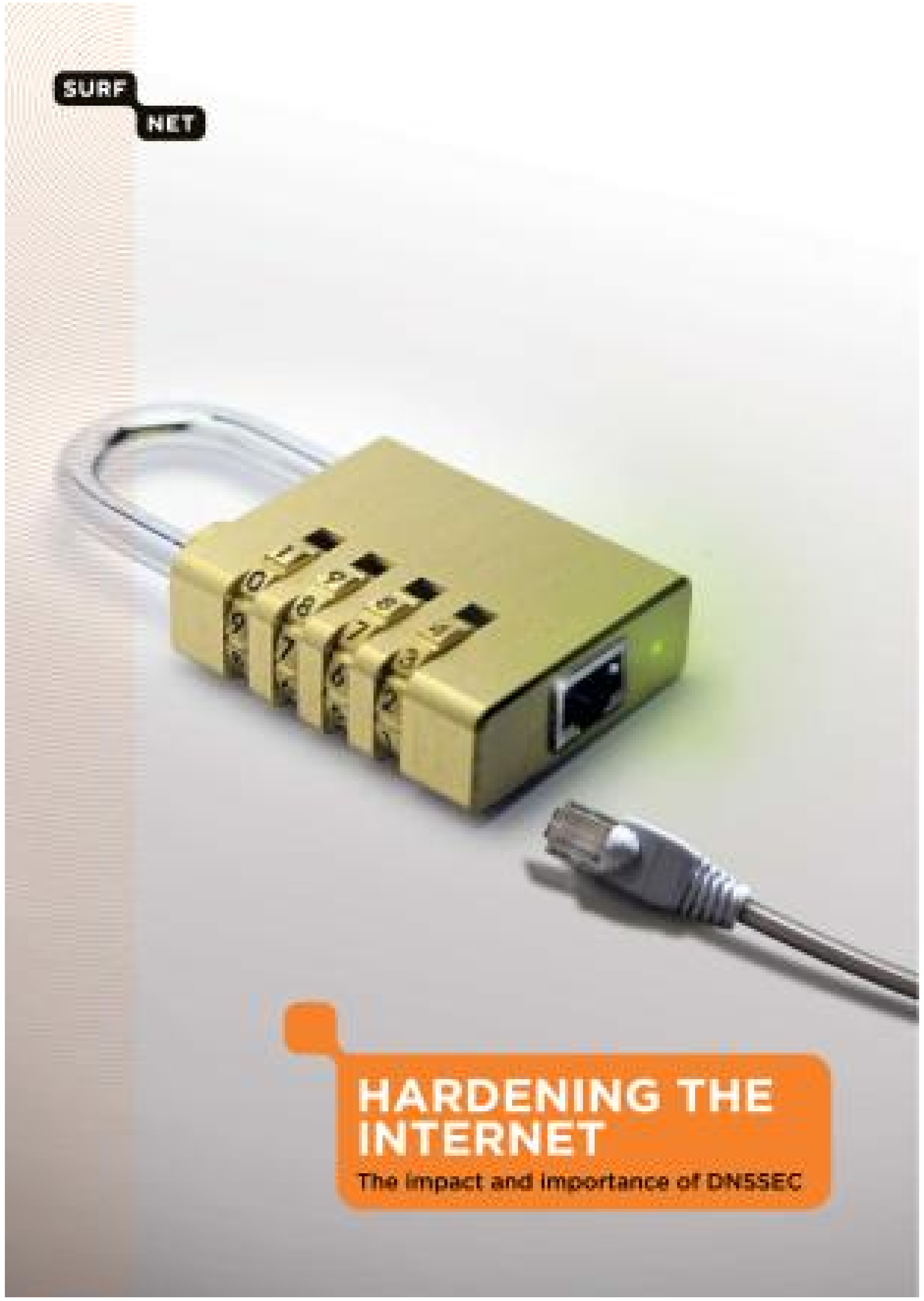


The .org signatur
are 1024-bit RSA

2003: Shamir–Tro
concluded that 10
was already breaka
large companies ar
$10 million: 1 key
$120 million: 1 ke

2003: RSA Labora
recommended a tr
2048-bit keys "ove
of this decade." 2
made the same re

All we care about is integrity:

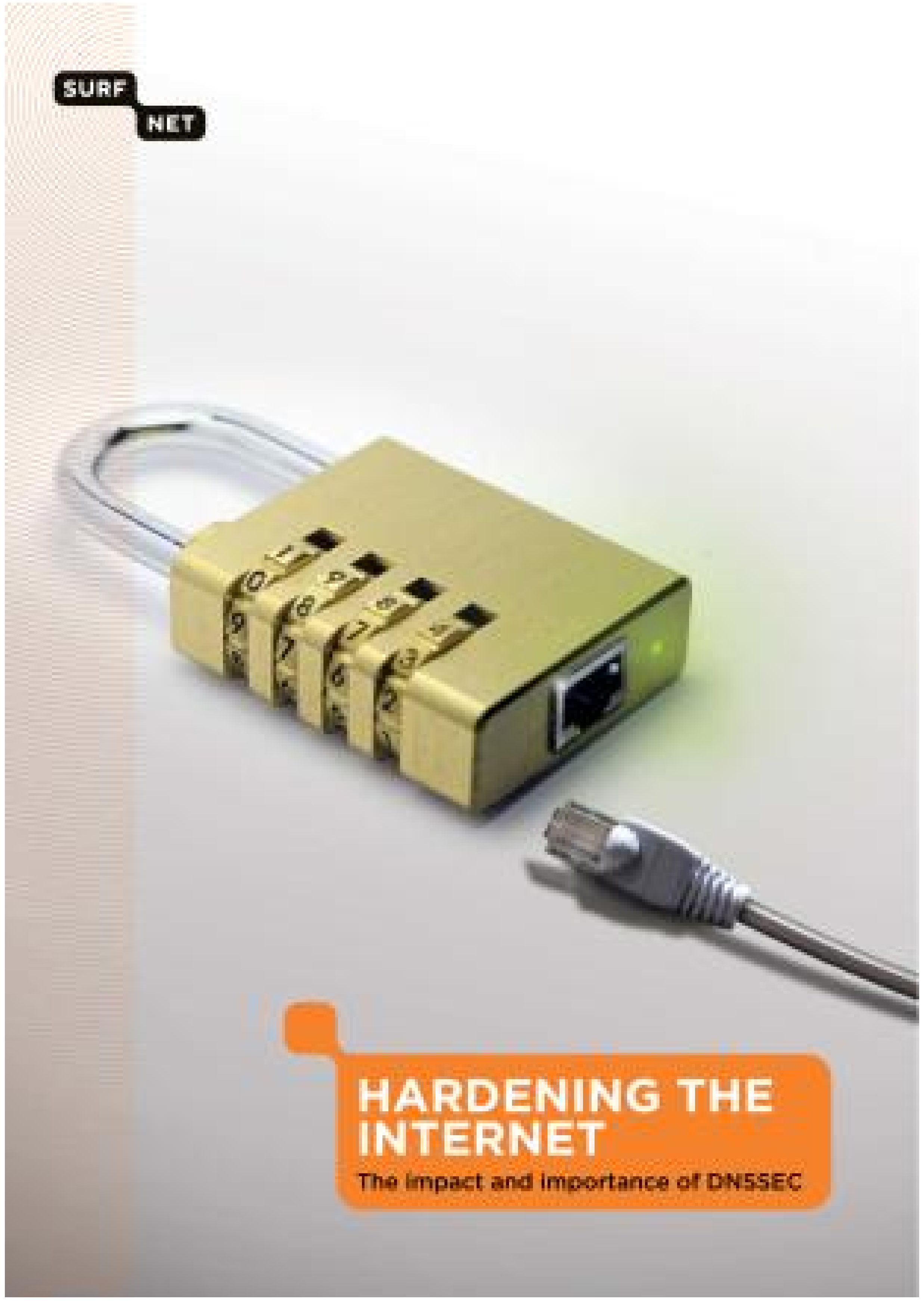


The .org signatures
are 1024-bit RSA signatures

2003: Shamir–Tromer et al.
concluded that 1024-bit RSA
was already breakable by
large companies and botnets
\$10 million: 1 key/year.
\$120 million: 1 key/month.

2003: RSA Laboratories
recommended a transition to
2048-bit keys "over the rem
of this decade." 2007: NIST
made the same recommenda

All we care about is integrity:



The `.org` signatures are 1024-bit RSA signatures.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets. \$10 million: 1 key/year. \$120 million: 1 key/month.

2003: RSA Laboratories recommended a transition to 2048-bit keys "over the remainder of this decade." 2007: NIST made the same recommendation.

are about is integrity:

The `.org` signatures are 1024-bit RSA signatures.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets.
$10 million: 1 key/year.
$120 million: 1 key/month.

2003: RSA Laboratories recommended a transition to 2048-bit keys "over the remainder of this decade." 2007: NIST made the same recommendation.

Academi
factored
Still no
of break

is integrity:

The `.org` signatures
are 1024-bit RSA signatures.

2003: Shamir–Tromer et al.
concluded that 1024-bit RSA
was already breakable by
large companies and botnets.
$10 million: 1 key/year.
$120 million: 1 key/month.

2003: RSA Laboratories
recommended a transition to
2048-bit keys "over the remainder
of this decade." 2007: NIST
made the same recommendation.

Academics in smal
factored RSA-768
Still no public ann
of breaks of 1024-

y:

The `.org` signatures
are 1024-bit RSA signatures.

2003: Shamir–Tromer et al.
concluded that 1024-bit RSA
was already breakable by
large companies and botnets.
$10 million: 1 key/year.
$120 million: 1 key/month.

2003: RSA Laboratories
recommended a transition to
2048-bit keys "over the remainder
of this decade." 2007: NIST
made the same recommendation.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcemen
of breaks of 1024-bit RSA.

The `.org` signatures
are 1024-bit RSA signatures.

2003: Shamir–Tromer et al.
concluded that 1024-bit RSA
was already breakable by
large companies and botnets.
\$10 million: 1 key/year.
\$120 million: 1 key/month.

2003: RSA Laboratories
recommended a transition to
2048-bit keys "over the remainder
of this decade." 2007: NIST
made the same recommendation.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

The `.org` signatures are 1024-bit RSA signatures.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets. \$10 million: 1 key/year. \$120 million: 1 key/month.

2003: RSA Laboratories recommended a transition to 2048-bit keys "over the remainder of this decade." 2007: NIST made the same recommendation.

Academics in small labs factored RSA-768 in 2009. Still no public announcements of breaks of 1024-bit RSA.

"RSA-1024: still secure against honest attackers."

The `.org` signatures
are 1024-bit RSA signatures.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets.
\$10 million: 1 key/year.
\$120 million: 1 key/month.

2003: RSA Laboratories recommended a transition to 2048-bit keys “over the remainder of this decade.” 2007: NIST made the same recommendation.

Academics in small labs factored RSA-768 in 2009.
Still no public announcements of breaks of 1024-bit RSA.

“RSA-1024: still secure against honest attackers.”

What about serious attackers using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

rg signatures
4-bit RSA signatures.

hamir–Tromer et al.
ed that 1024-bit RSA
ady breakable by
mpanies and botnets.
ion: 1 key/year.
llion: 1 key/month.

SA Laboratories
ended a transition to
keys "over the remainder
ecade." 2007: NIST
e same recommendation.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

"RSA-1024: still secure
against honest attackers."

What about serious attackers
using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that
with the
for `gree`

res
signatures.

mer et al.
24-bit RSA
able by
nd botnets.
/year.
y/month.

atories
ansition to
er the remainder
007: NIST
commendation.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

"RSA-1024: still secure
against honest attackers."

What about serious attackers
using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that's not the
with these DNSSE
for `greenpeace.o`

.

A

s.

o
ainder
T
ation.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

"RSA-1024: still secure
against honest attackers."

What about serious attackers
using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that's not the big probl
with these DNSSEC signatu
for `greenpeace.org`.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

"RSA-1024: still secure
against honest attackers."

What about serious attackers
using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that's not the big problem
with these DNSSEC signatures
for `greenpeace.org`.

Academics in small labs factored RSA-768 in 2009. Still no public announcements of breaks of 1024-bit RSA.

"RSA-1024: still secure against honest attackers."

What about serious attackers using many more computers? e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that's not the big problem with these DNSSEC signatures for `greenpeace.org`.

Suppose an attacker forges a DNS packet from `.org`, including exactly the same DNSSEC signatures but *changing the NS+A records* to point to the attacker's servers.

Academics in small labs
factored RSA-768 in 2009.
Still no public announcements
of breaks of 1024-bit RSA.

"RSA-1024: still secure
against honest attackers."

What about serious attackers
using many more computers?
e.g. botnet operators?

I say:
Using RSA-1024 is irresponsible.

But that's not the big problem
with these DNSSEC signatures
for `greenpeace.org`.

Suppose an attacker forges
a DNS packet from `.org`,
including exactly the same
DNSSEC signatures but
changing the NS+A records to
point to the attacker's servers.

Fact: DNSSEC "verification"
won't notice the change.
The signatures say *nothing*
about the NS+A records.
The forgery will be accepted.

ics in small labs
RSA-768 in 2009.
public announcements
s of 1024-bit RSA.

024: still secure
honest attackers."

bout serious attackers
any more computers?
net operators?

SA-1024 is irresponsible.

But that's not the big problem with these DNSSEC signatures for `greenpeace.org`.

Suppose an attacker forges a DNS packet from `.org`, including exactly the same DNSSEC signatures but *changing the NS+A records* to point to the attacker's servers.

Fact: DNSSEC "verification" won't notice the change. The signatures say *nothing* about the NS+A records. *The forgery will be accepted.*

Here's w
translate
"`.org m`
`with ha`
`h9p7u7tr`
`h9parr66`
`but has`
`that da`

Can che
has a ha

`.org` no
of these
This is .
a "neede

ll labs
in 2009.
ouncements
bit RSA.

ecure
ackers."

us attackers
computers?
ors?

s irresponsible.

But that's not the big problem with these DNSSEC signatures for `greenpeace.org`.

Suppose an attacker forges a DNS packet from `.org`, including exactly the same DNSSEC signatures but *changing the NS+A records* to point to the attacker's servers.

Fact: DNSSEC "verification" won't notice the change. The signatures say *nothing* about the NS+A records. *The forgery will be accepted.*

Here's what `.org`
translated into En
"`.org might hav`
`with hashes bet`
`h9p7u7tr2u91d0v0lj`
`h9parr669t6u8o1gsg`
`but has not sig`
`that data.`"

Can check that gr
has a hash in that

`.org` now has tho
of these useless sig
This is `.org` "imp
a "needed security

ts

rs

?

sible.

But that's not the big problem with these DNSSEC signatures for `greenpeace.org`.

Suppose an attacker forges a DNS packet from `.org`, including exactly the same DNSSEC signatures but *changing the NS+A records* to point to the attacker's servers.

Fact: DNSSEC "verification" won't notice the change. The signatures say *nothing* about the NS+A records. *The forgery will be accepted.*

Here's what `.org` signed, translated into English: "`.org might have data with hashes between h9p7u7tr2u91d0v0ljs9l1gidnp9 h9parr669t6u8o1gsg9e1lmitk4d but has not signed any o that data.`"

Can check that greenpeace has a hash in that range.

`.org` now has thousands of these useless signatures. This is `.org` "implementing a "needed security measure.

But that's not the big problem with these DNSSEC signatures for `greenpeace.org`.

Suppose an attacker forges a DNS packet from `.org`, including exactly the same DNSSEC signatures but *changing the NS+A records* to point to the attacker's servers.

Fact: DNSSEC "verification" won't notice the change. The signatures say *nothing* about the NS+A records. *The forgery will be accepted.*

Here's what `.org` signed, translated into English:

"`.org might have data with hashes between h9p7u7tr2u91d0v0ljs9l1gidnp90u3h, h9parr669t6u8o1gsg9e1lmitk4dem0t but has not signed any of that data.`"

Can check that `greenpeace.org` has a hash in that range.

`.org` now has thousands of these useless signatures. This is `.org` "implementing" a "needed security measure."

t's not the big problem
se DNSSEC signatures
enpeace.org.

an attacker forges
packet from .org,
g exactly the same
C signatures but
g *the NS+A records* to
the attacker's servers.

NSSEC "verification"
otice the change.
natures say *nothing*
he NS+A records.
gery will be accepted.

Here's what .org signed,
translated into English:
"`.org might have data
with hashes between
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h,
h9parr669t6u8o1gsg9e1lmitk4dem0t
but has not signed any of
that data.`"

Can check that greenpeace.org
has a hash in that range.

.org now has thousands
of these useless signatures.
This is .org "implementing"
a "needed security measure."

"DNSSE

big problem
C signatures
org.

er forges
n .org,
he same
es but
A *records* to
ker's servers.

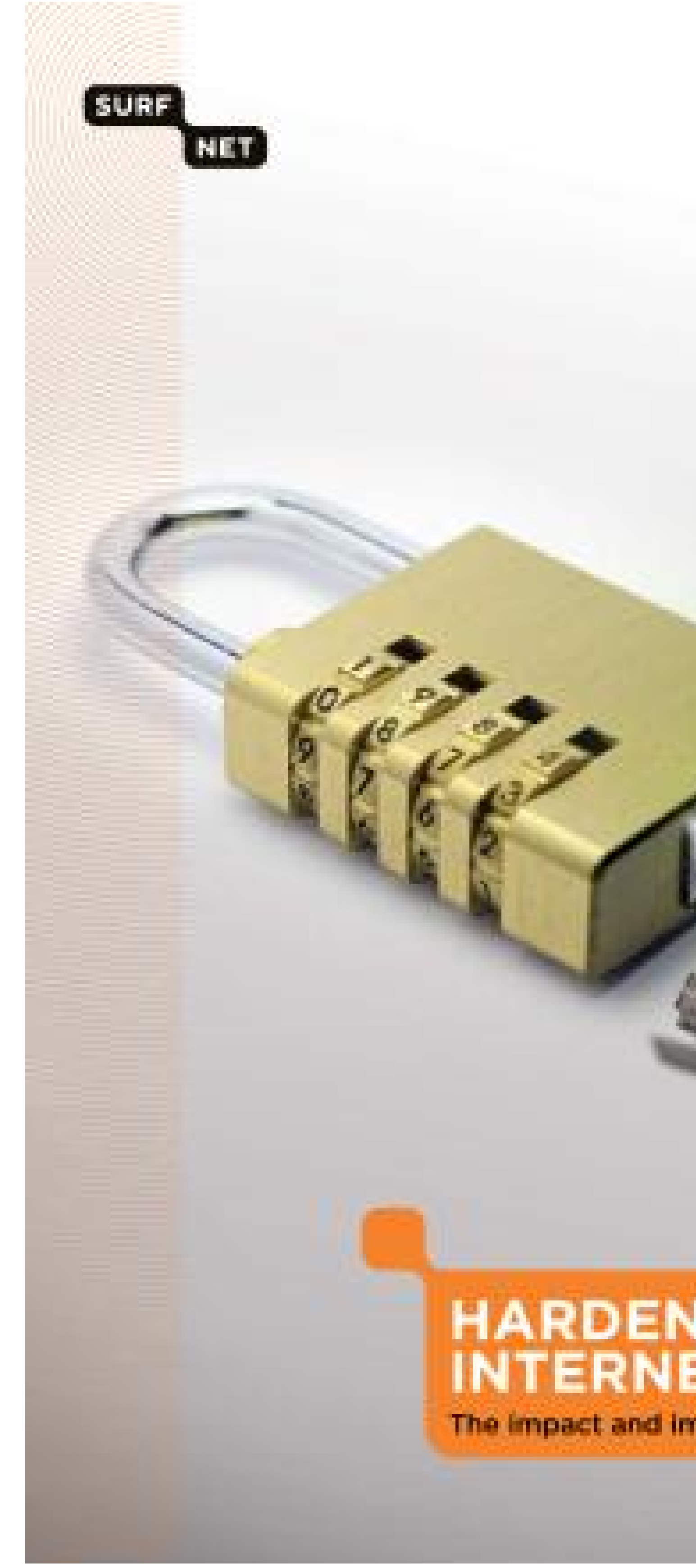
erification"
hange.
nothing
records.
e accepted.

Here's what .org signed,
translated into English:
"`.org might have data with hashes between h9p7u7tr2u91d0v0ljs9l1gidnp90u3h, h9parr669t6u8o1gsg9e1lmitk4dem0t but has not signed any of that data.`"

Can check that greenpeace.org has a hash in that range.

.org now has thousands
of these useless signatures.
This is .org "implementing"
a "needed security measure."

"DNSSEC: Built,



em
res
to
rs.
”
d.

Here’s what `.org` signed,
translated into English:

“`.org might have data with hashes between h9p7u7tr2u91d0v0ljs9l1gidnp90u3h, h9parr669t6u8o1gsg9e1lmitk4dem0t but has not signed any of that data.`”

Can check that `greenpeace.org` has a hash in that range.

`.org` now has thousands of these useless signatures.
This is `.org` “implementing” a “needed security measure.”

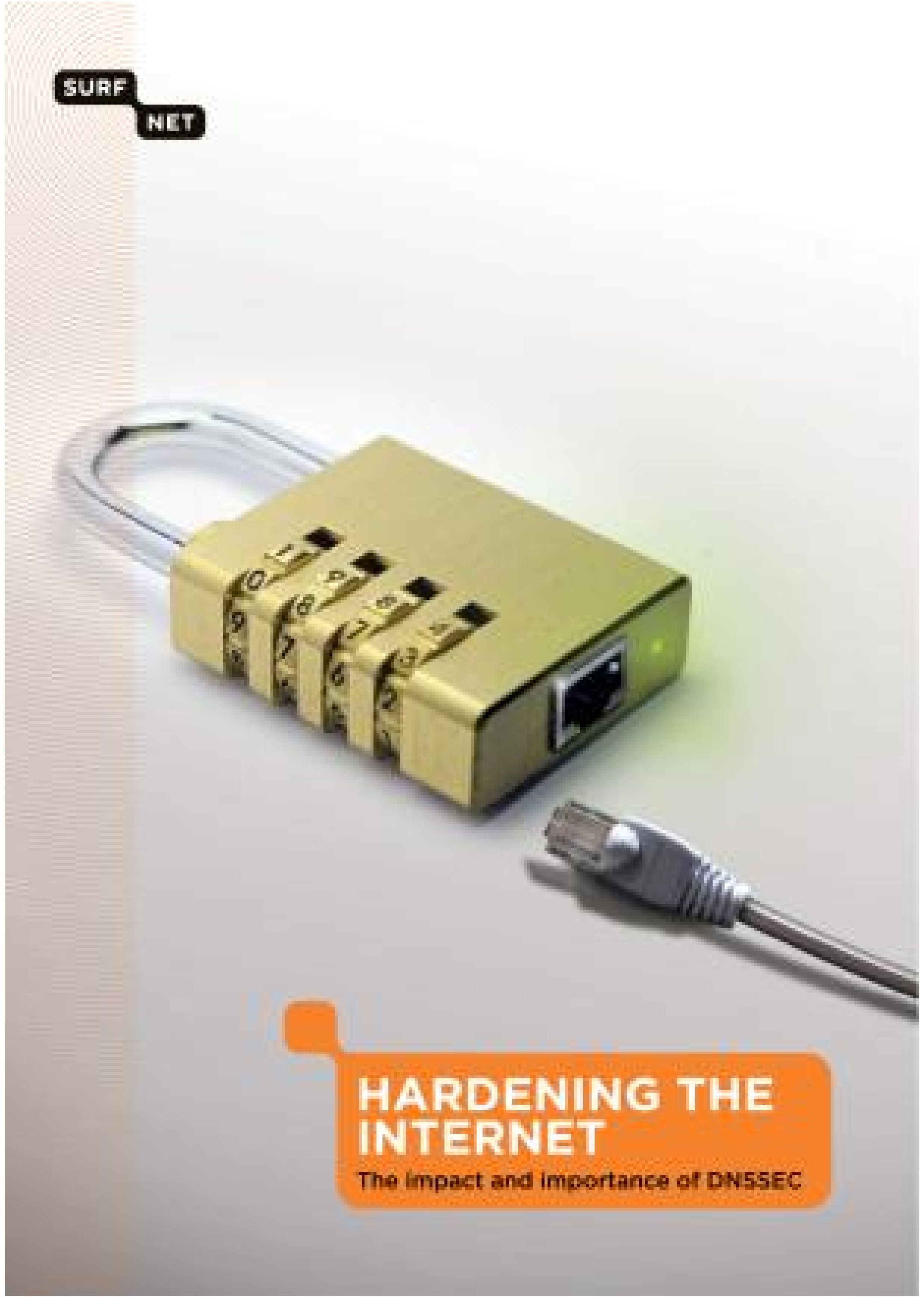
“DNSSEC: Built, not plugge

Here's what .org signed,
translated into English:
"`.org might have data with hashes between h9p7u7tr2u91d0v0ljs9l1gidnp90u3h, h9parr669t6u8o1gsg9e1lmitk4dem0t but has not signed any of that data.`"

Can check that greenpeace.org has a hash in that range.

.org now has thousands of these useless signatures. This is .org "implementing" a "needed security measure."

"DNSSEC: Built, not plugged in."

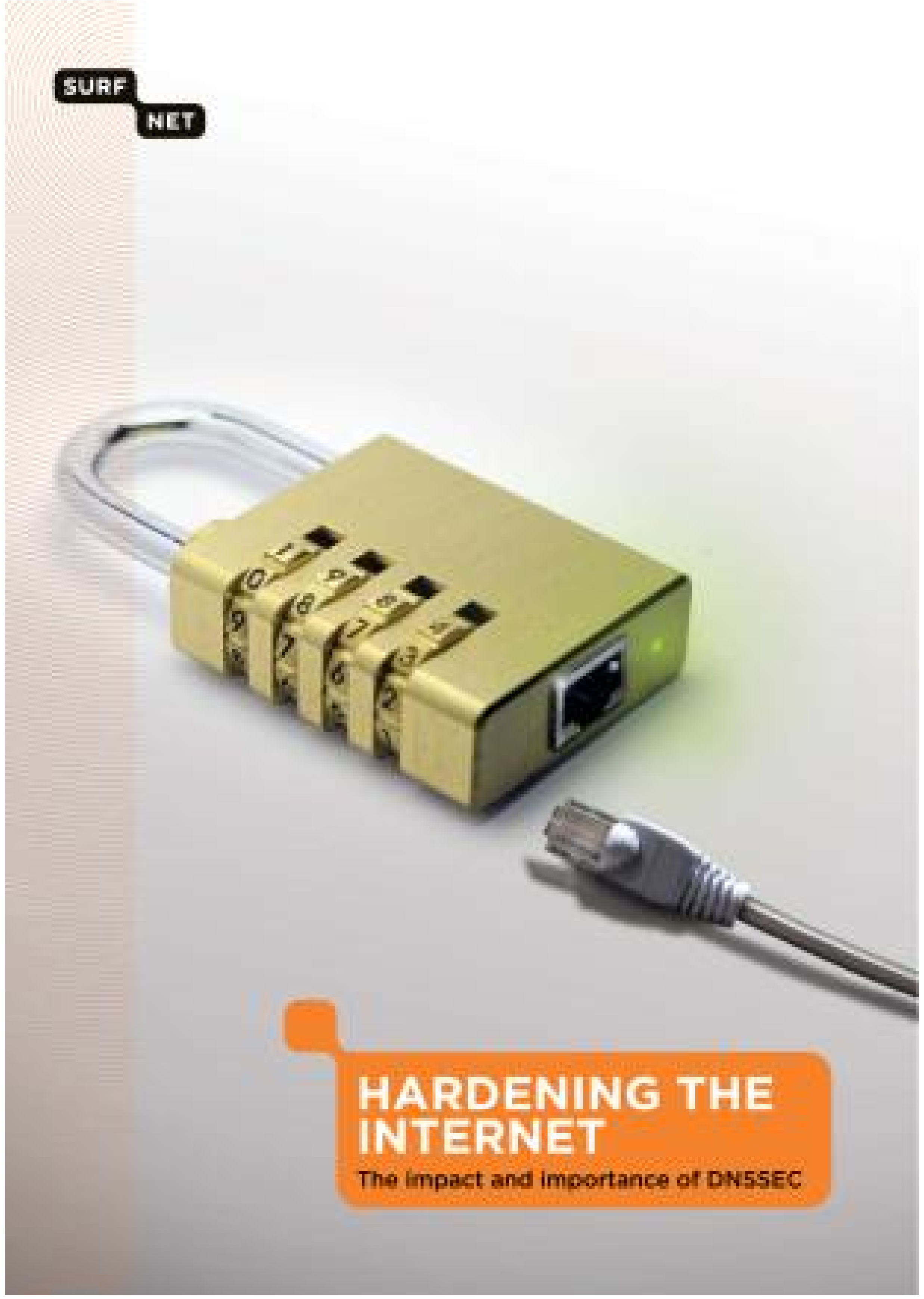


vhat .org signed,
ed into English:
ight have data
shes between
2u91d0v0ljs9l1gidnp90u3h,
9t6u8o1gsg9e1lmitk4dem0t
not signed any of
ta."

ck that greenpeace.org
sh in that range.

w has thousands
useless signatures.
.org "implementing"
ed security measure."

"DNSSEC: Built, not plugged in."



What we

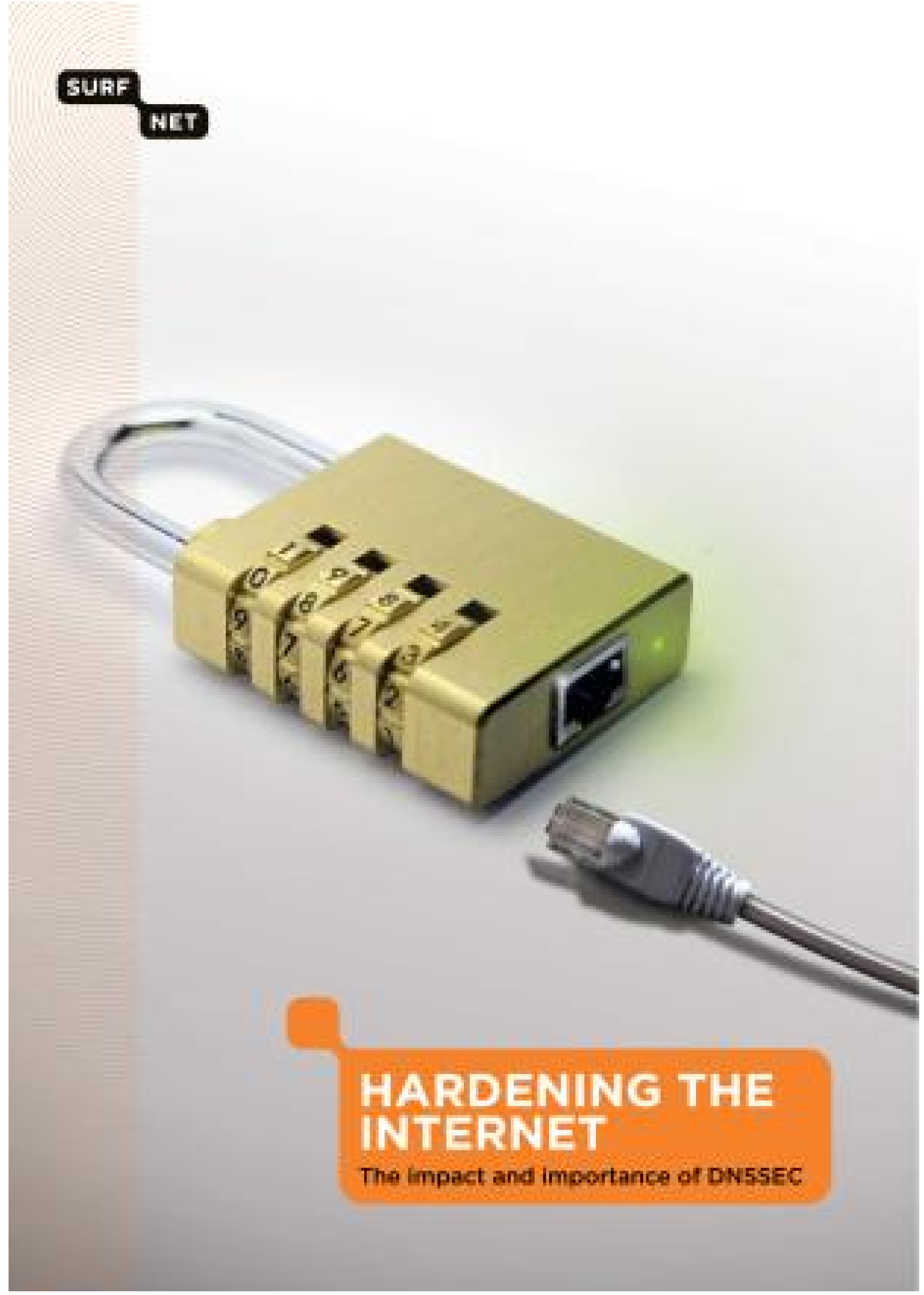
Rushed

signed,
glish:
e data
ween
s9l1gidnp90u3h,
9e1lmitk4dem0t
ned any of

reenpeace.org
range.

usands
gnatures.
lementing"
measure."

"DNSSEC: Built, not plugged in."



What went wrong

Rushed developme

0u3h,

em0t

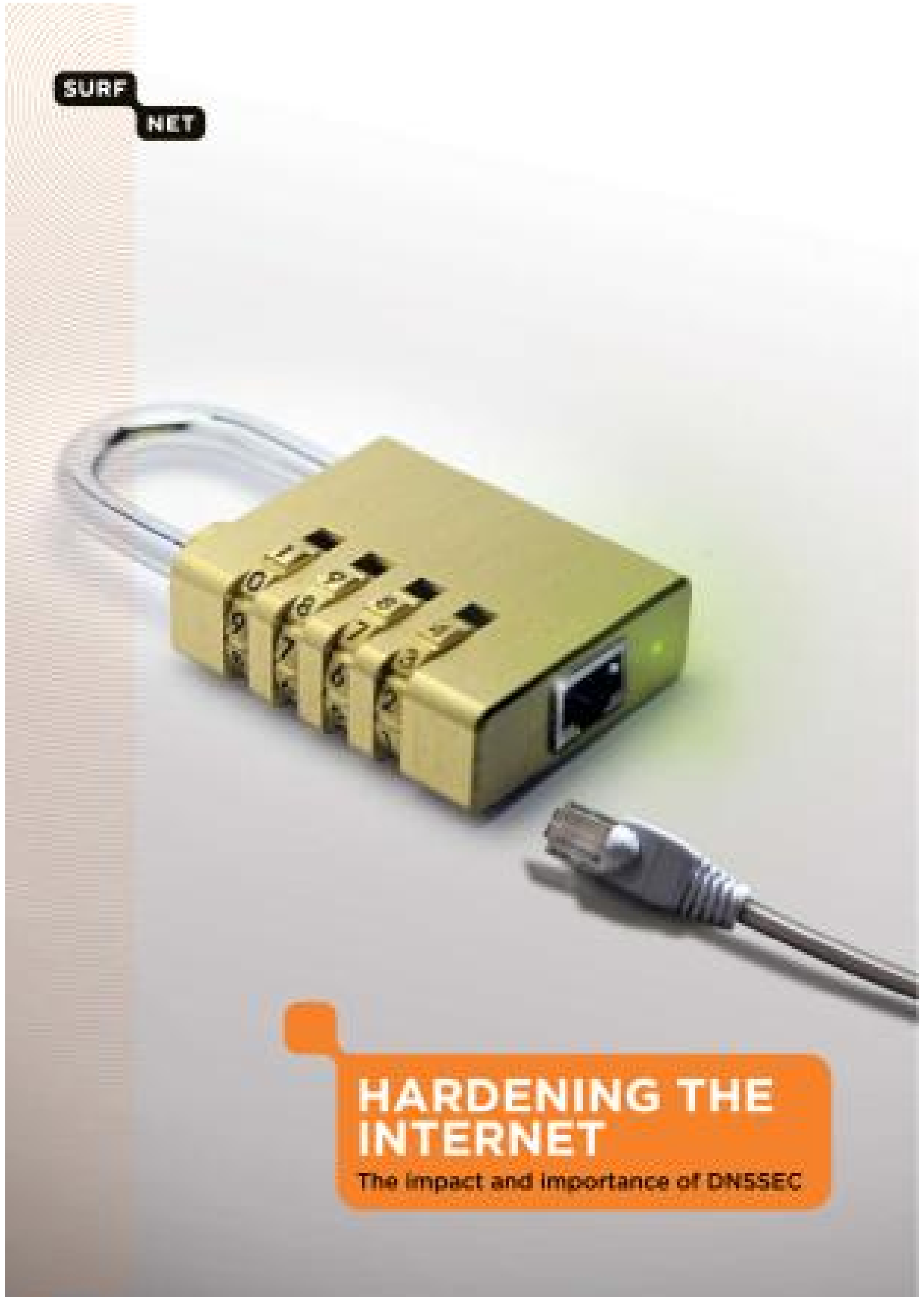
f

e.org

''

''

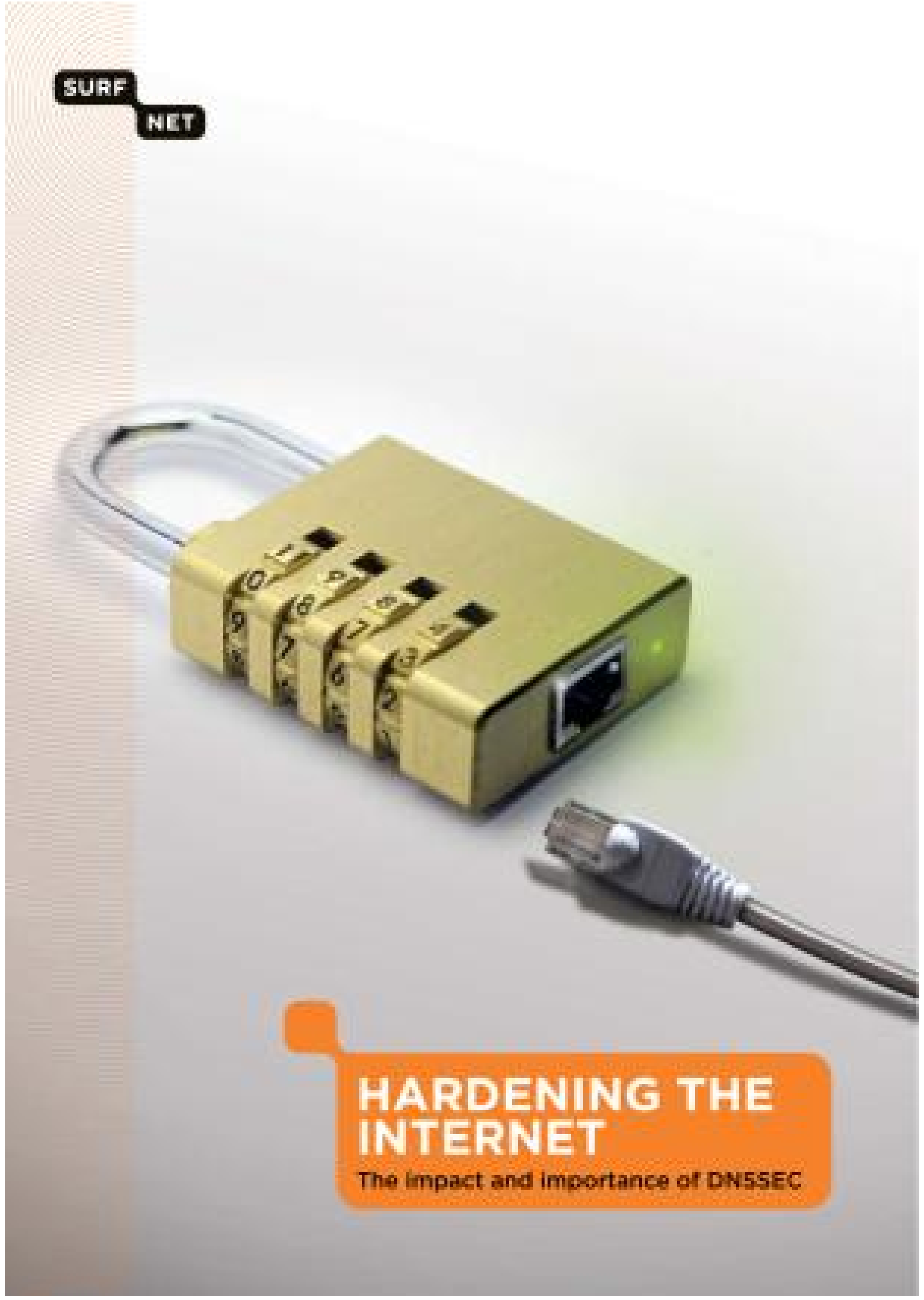
“DNSSEC: Built, not plugged in.”



What went wrong?

Rushed development proces

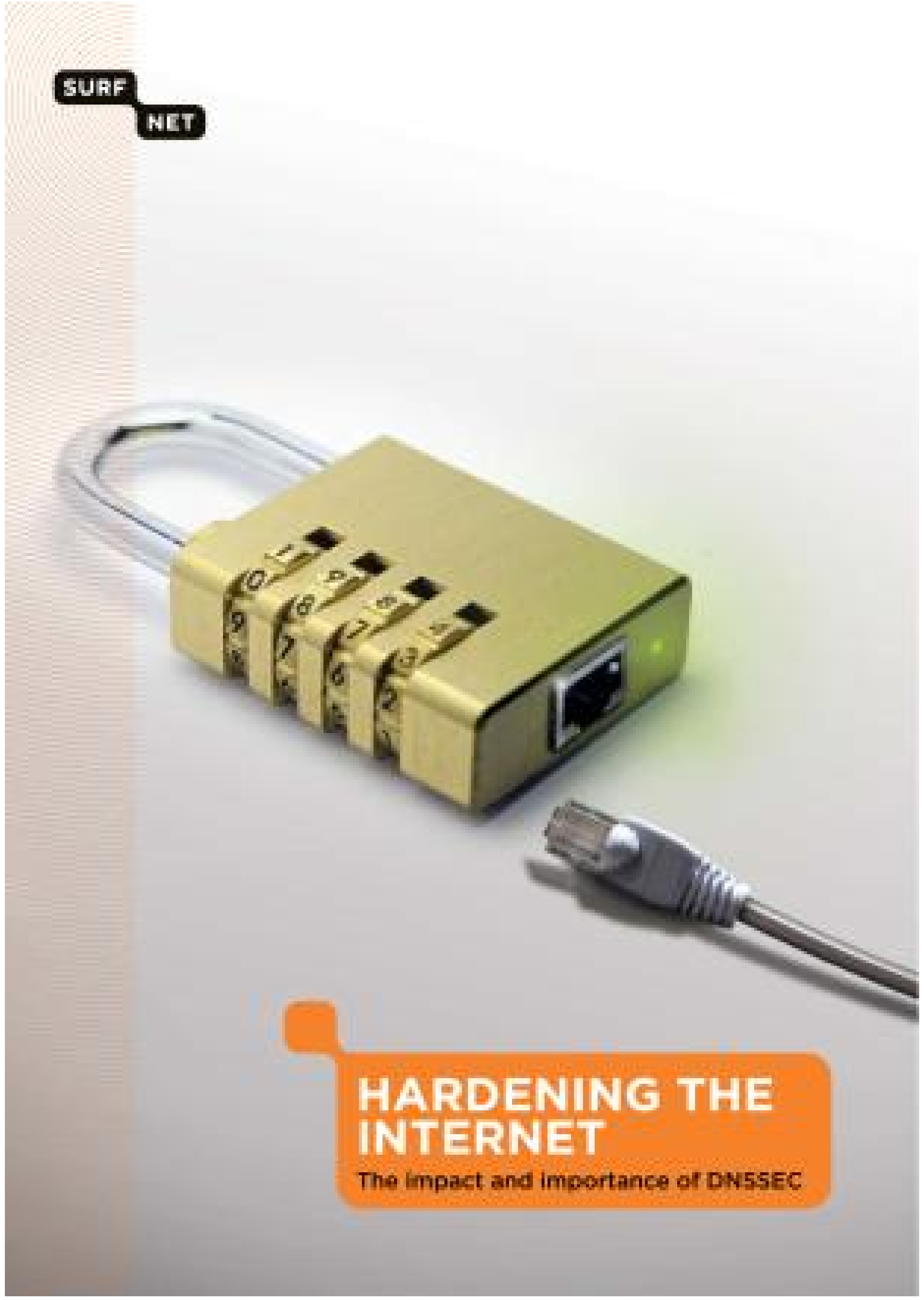
"DNSSEC: Built, not plugged in."



What went wrong?

Rushed development process?

“DNSSEC: Built, not plugged in.”

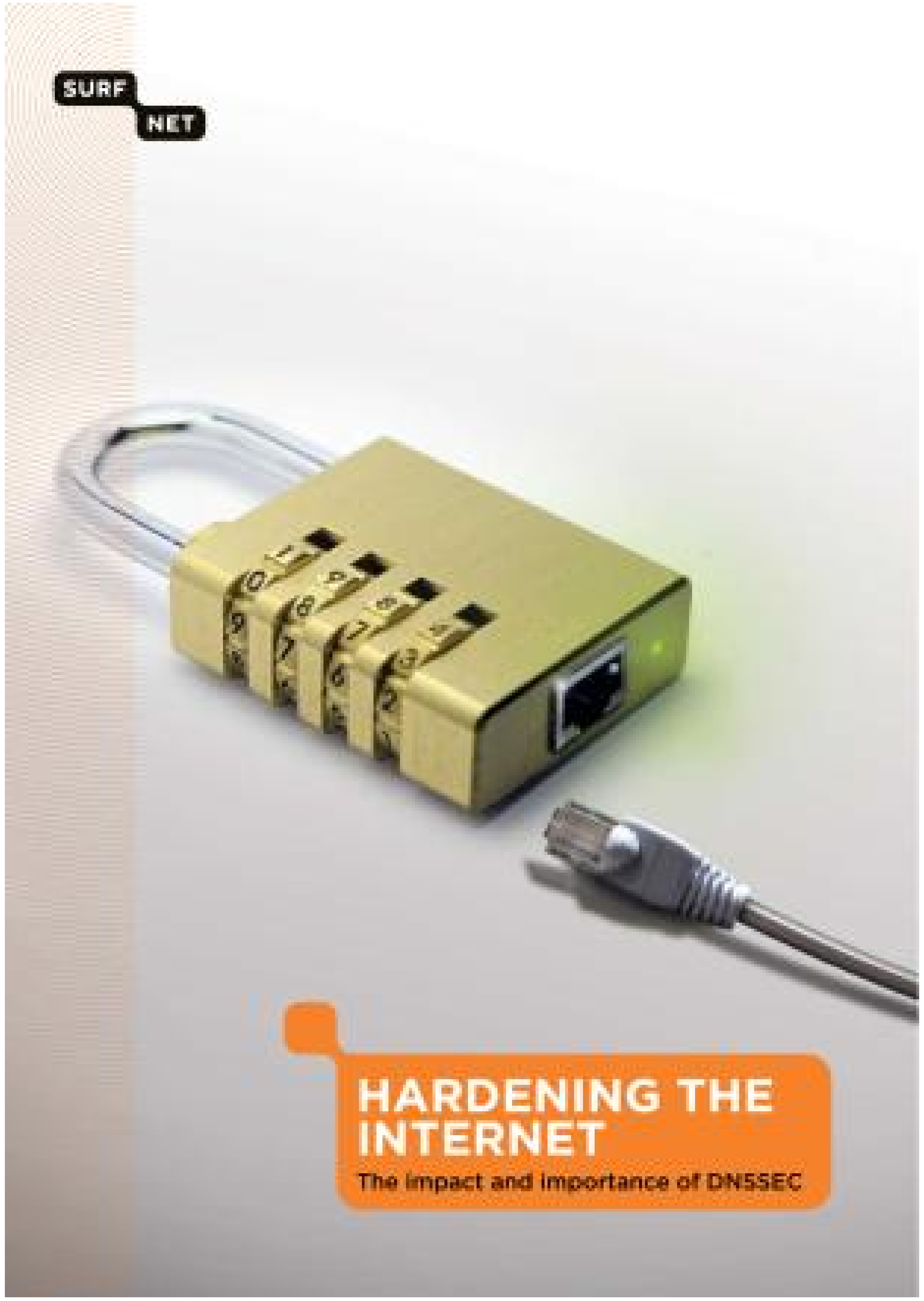


What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

“DNSSEC: Built, not plugged in.”



What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: “The DNS Security design team of the DNS working group met for one morning at the Houston IETF.”

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: “DNSSEC” protocol specification.

EC: Built, not plugged in.”

What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: “The DNS Security design team of the DNS working group met for one morning at the Houston IETF.”

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: “DNSSEC” protocol specification.

Millions
of U.S.
DISA to
NSF to
Secure64

Continui
DNSSEC
IETF DN
protocol
software

not plugged in.”

What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: “The DNS Security design team of the DNS working group met for one morning at the Houston IETF.”

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: “DNSSEC” protocol specification.

Millions of dollars
of U.S. governmen
DISA to BIND co
NSF to UCLA; DH
Secure64 Software

Continuing cycle o
DNSSEC impleme
IETF DNSSEC dis
protocol updates,
software implemen

ed in."

What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: "The DNS Security design team of the DNS working group met for one morning at the Houston IETF."

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: "DNSSEC" protocol specification.

Millions of dollars of U.S. government grants: DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporat

Continuing cycle of DNSSEC implementations, IETF DNSSEC discussions, protocol updates, revised software implementations, e

What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: "The DNS Security design team of the DNS working group met for one morning at the Houston IETF."

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: "DNSSEC" protocol specification.

Millions of dollars of U.S. government grants: e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation.

Continuing cycle of DNSSEC implementations, IETF DNSSEC discussions, protocol updates, revised software implementations, etc.

What went wrong?

Rushed development process?

No: DNSSEC has been under active development for *two decades*.

1993.11 Galvin: "The DNS Security design team of the DNS working group met for one morning at the Houston IETF."

1994.02 Eastlake–Kaufman, after months of discussions on `dns-security` mailing list: "DNSSEC" protocol specification.

Millions of dollars of U.S. government grants: e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation.

Continuing cycle of DNSSEC implementations, IETF DNSSEC discussions, protocol updates, revised software implementations, etc.

Compatibility trap? No. Several DNSSEC updates have broken compatibility with older implementations.

ent wrong?

development process?

SSEC has been
ctive development
decades.

Galvin: "The DNS
design team of the
rking group met for one
at the Houston IETF."

Eastlake–Kaufman,
onths of discussions on
`curity` mailing list:
EC" protocol specification.

Millions of dollars
of U.S. government grants: e.g.,
DISA to BIND company;
NSF to UCLA; DHS to
Secure64 Software Corporation.

Continuing cycle of
DNSSEC implementations,
IETF DNSSEC discussions,
protocol updates, revised
software implementations, etc.

Compatibility trap? No.
Several DNSSEC updates
have broken compatibility
with older implementations.

The perf

Some of
servers a
the root
the goog
Can they

?

ent process?

been
opment

The DNS
am of the
up met for one
ouston IETF."

Kaufman,
scussions on
ailing list:
col specification.

Millions of dollars of U.S. government grants: e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation.

Continuing cycle of DNSSEC implementations, IETF DNSSEC discussions, protocol updates, revised software implementations, etc.

Compatibility trap? No. Several DNSSEC updates have broken compatibility with older implementations.

The performance t

Some of the Intern
servers are extrem
the root servers, t
the `google.com` s
Can they afford cr

s?

one
TF.”

on

cation.

Millions of dollars
of U.S. government grants: e.g.,
DISA to BIND company;
NSF to UCLA; DHS to
Secure64 Software Corporation.

Continuing cycle of
DNSSEC implementations,
IETF DNSSEC discussions,
protocol updates, revised
software implementations, etc.

Compatibility trap? No.
Several DNSSEC updates
have broken compatibility
with older implementations.

The performance trap

Some of the Internet's DNS
servers are extremely busy:
the root servers, the `.com` se
the `google.com` servers.
Can they afford crypto?

Millions of dollars
of U.S. government grants: e.g.,
DISA to BIND company;
NSF to UCLA; DHS to
Secure64 Software Corporation.

Continuing cycle of
DNSSEC implementations,
IETF DNSSEC discussions,
protocol updates, revised
software implementations, etc.

Compatibility trap? No.
Several DNSSEC updates
have broken compatibility
with older implementations.

The performance trap

Some of the Internet's DNS
servers are extremely busy: e.g.,
the root servers, the `.com` servers,
the `google.com` servers.
Can they afford crypto?

Millions of dollars
of U.S. government grants: e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation.

Continuing cycle of DNSSEC implementations, IETF DNSSEC discussions, protocol updates, revised software implementations, etc.

Compatibility trap? No. Several DNSSEC updates have broken compatibility with older implementations.

The performance trap

Some of the Internet's DNS servers are extremely busy: e.g., the root servers, the `.com` servers, the `google.com` servers. Can they afford crypto?

The critical design decision in DNSSEC: *precompute* signatures of DNS records. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the server can afford to sign each DNS record once.

of dollars
government grants: e.g.,
BIND company;
UCLA; DHS to
4 Software Corporation.

ing cycle of
C implementations,
NSSEC discussions,
updates, revised
implementations, etc.

ibility trap? No.
DNSSEC updates
oken compatibility
er implementations.

The performance trap

Some of the Internet's DNS servers are extremely busy: e.g., the root servers, the `.com` servers, the `google.com` servers. Can they afford crypto?

The critical design decision in DNSSEC: *precompute* signatures of DNS records. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the server can afford to sign each DNS record once.

Clients c
of *verify*

DNSSEC
client-sic
precomp
choice o

Many D
640-bit
768-bit
1024-bit
(for "lea
DSA, "1
for verifi
signatur

nt grants: e.g.,
mpany;
HS to
e Corporation.

of
ntations,
scussions,
revised
ntations, etc.

? No.
updates
atibility
entations.

<u>The performance trap</u>

Some of the Internet's DNS servers are extremely busy: e.g., the root servers, the `.com` servers, the `google.com` servers.
Can they afford crypto?

The critical design decision in DNSSEC: *precompute* signatures of DNS records.
"Per-query crypto is bad."

Signature is computed once; saved; sent to many clients.
Hopefully the server can afford to sign each DNS record once.

Clients don't share
of *verifying* a sign

DNSSEC tries to
client-side costs (a
precomputation co
choice of crypto p

Many DNSSEC cr
640-bit RSA, origi
768-bit RSA, man
1024-bit RSA, cur
(for "leaf nodes in
DSA, "10 to 40 ti
for verification" bu
signatures.

e.g.,

ion.

tc.

The performance trap

Some of the Internet's DNS servers are extremely busy: e.g., the root servers, the `.com` servers, the `google.com` servers. Can they afford crypto?

The critical design decision in DNSSEC: *precompute* signatures of DNS records. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the server can afford to sign each DNS record once.

Clients don't share the work of *verifying* a signature.

DNSSEC tries to reduce client-side costs (and precomputation costs) throu choice of crypto primitive.

Many DNSSEC crypto optio
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs
(for "leaf nodes in the DNS'
DSA, "10 to 40 times as slo
for verification" but faster fo
signatures.

The performance trap

Some of the Internet's DNS servers are extremely busy: e.g., the root servers, the `.com` servers, the `google.com` servers. Can they afford crypto?

The critical design decision in DNSSEC: *precompute* signatures of DNS records. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the server can afford to sign each DNS record once.

Clients don't share the work of *verifying* a signature.

DNSSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs (for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow for verification" but faster for signatures.

formance trap

the Internet's DNS
are extremely busy: e.g.,
servers, the `.com` servers,
`gle.com` servers.
y afford crypto?

ical design decision
SEC: *precompute*
es of DNS records.
ery crypto is bad."

re is computed once;
ent to many clients.
y the server can afford
each DNS record once.

Clients don't share the work
of *verifying* a signature.

DNSSEC tries to reduce
client-side costs (and
precomputation costs) through
choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs
(for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow
for verification" but faster for
signatures.

DNSSE
such as
for no re
fear of o

DNSSE
to surviv
More co
including

trap

net's DNS
ely busy: e.g.,
he `.com` servers,
ervers.
ypto?

decision
ompute
records.
is bad."

uted once;
ny clients.
er can afford
record once.

Clients don't share the work of *verifying* a signature.

DNSSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs (for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow for verification" but faster for signatures.

DNSSEC made br
such as 640-bit RS
for no reason othe
fear of overload.

DNSSEC needed r
to survive the inev
More complexity =
including security

e.g.,
ervers,

;

ord
ce.

Clients don't share the work of *verifying* a signature.

DNSSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs (for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow for verification" but faster for signatures.

DNSSEC made breakable ch
such as 640-bit RSA
for no reason other than
fear of overload.

DNSSEC needed more optio
to survive the inevitable brea
More complexity $\Rightarrow$ more bu
including security holes.

Clients don't share the work
of *verifying* a signature.

DNSSEC tries to reduce
client-side costs (and
precomputation costs) through
choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs
(for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow
for verification" but faster for
signatures.

DNSSEC made breakable choices
such as 640-bit RSA
for no reason other than
fear of overload.

DNSSEC needed more options
to survive the inevitable breaks.
More complexity $\Rightarrow$ more bugs,
including security holes.

Clients don't share the work of *verifying* a signature.

DNSSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many DNSSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current RFCs (for "leaf nodes in the DNS");
DSA, "10 to 40 times as slow for verification" but faster for signatures.

DNSSEC made breakable choices such as 640-bit RSA for no reason other than fear of overload.

DNSSEC needed more options to survive the inevitable breaks.
More complexity ⇒ more bugs, including security holes.

Looking beyond the crypto:
Precomputation forced DNSSEC down a path of unreliability, insecurity, and unusability.
Let's see how this happened.

don't share the work
ing a signature.

C tries to reduce
de costs (and
putation costs) through
f crypto primitive.

NSSEC crypto options:
RSA, original specs;
RSA, many docs;
RSA, current RFCs
f nodes in the DNS");
.0 to 40 times as slow
cation" but faster for
es.

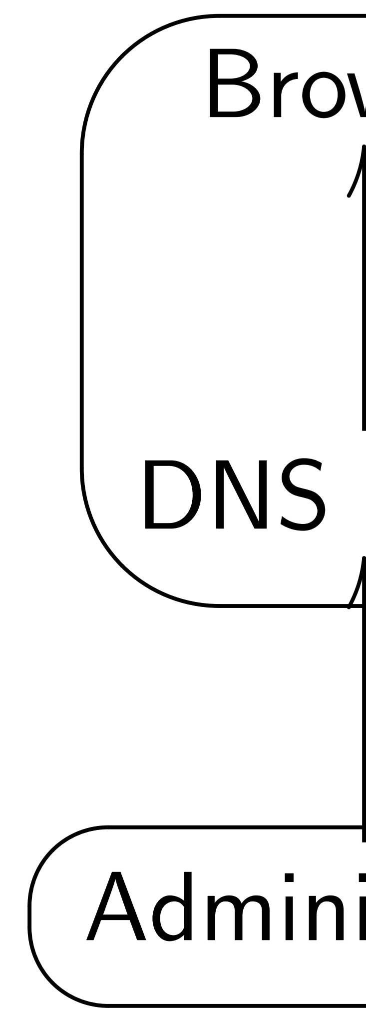
DNSSEC made breakable choices
such as 640-bit RSA
for no reason other than
fear of overload.

DNSSEC needed more options
to survive the inevitable breaks.
More complexity ⇒ more bugs,
including security holes.

Looking beyond the crypto:
Precomputation forced DNSSEC
down a path of unreliability,
insecurity, and unusability.
Let's see how this happened.

DNS arc

Browser
DNS cac



Cache p
administ
doesn't

e the work
ature.

reduce
and
osts) through
rimitive.

ypto options:
nal specs;
y docs;
rent RFCs
the DNS");
mes as slow
ut faster for

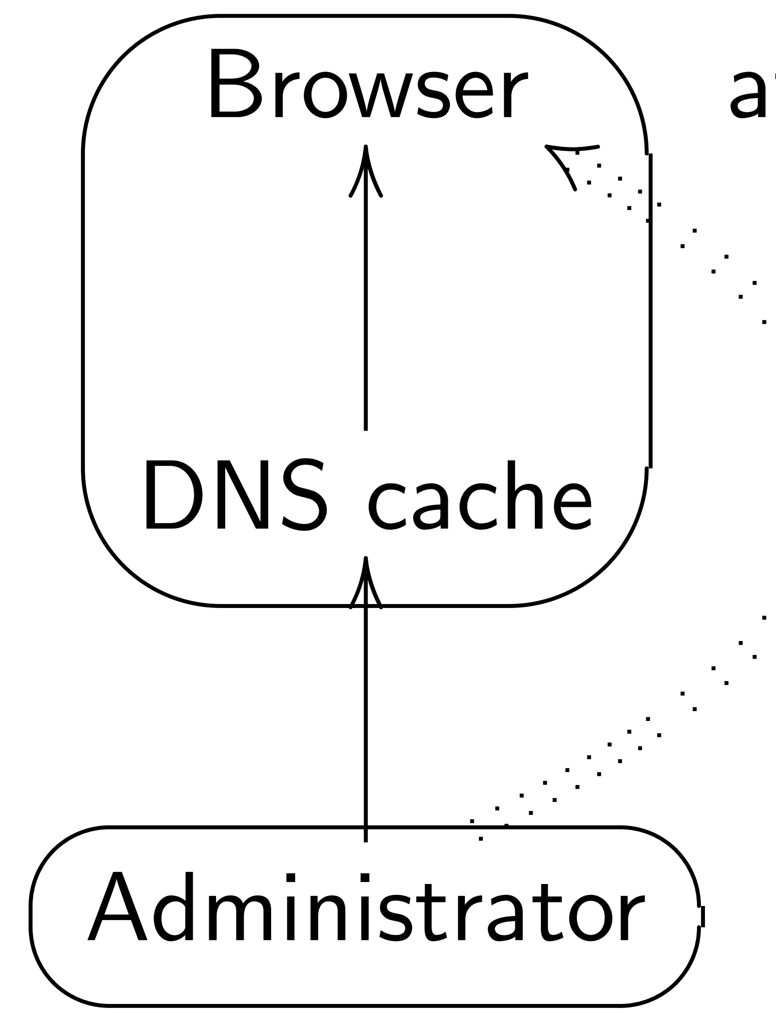
DNSSEC made breakable choices
such as 640-bit RSA
for no reason other than
fear of overload.

DNSSEC needed more options
to survive the inevitable breaks.
More complexity ⇒ more bugs,
including security holes.

Looking beyond the crypto:
Precomputation forced DNSSEC
down a path of unreliability,
insecurity, and unusability.
Let's see how this happened.

DNS architecture

Browser pulls data
DNS cache at `tue`



Cache pulls data f
administrator if it
doesn't already ha

ugh

ons:

,

');

ow

or

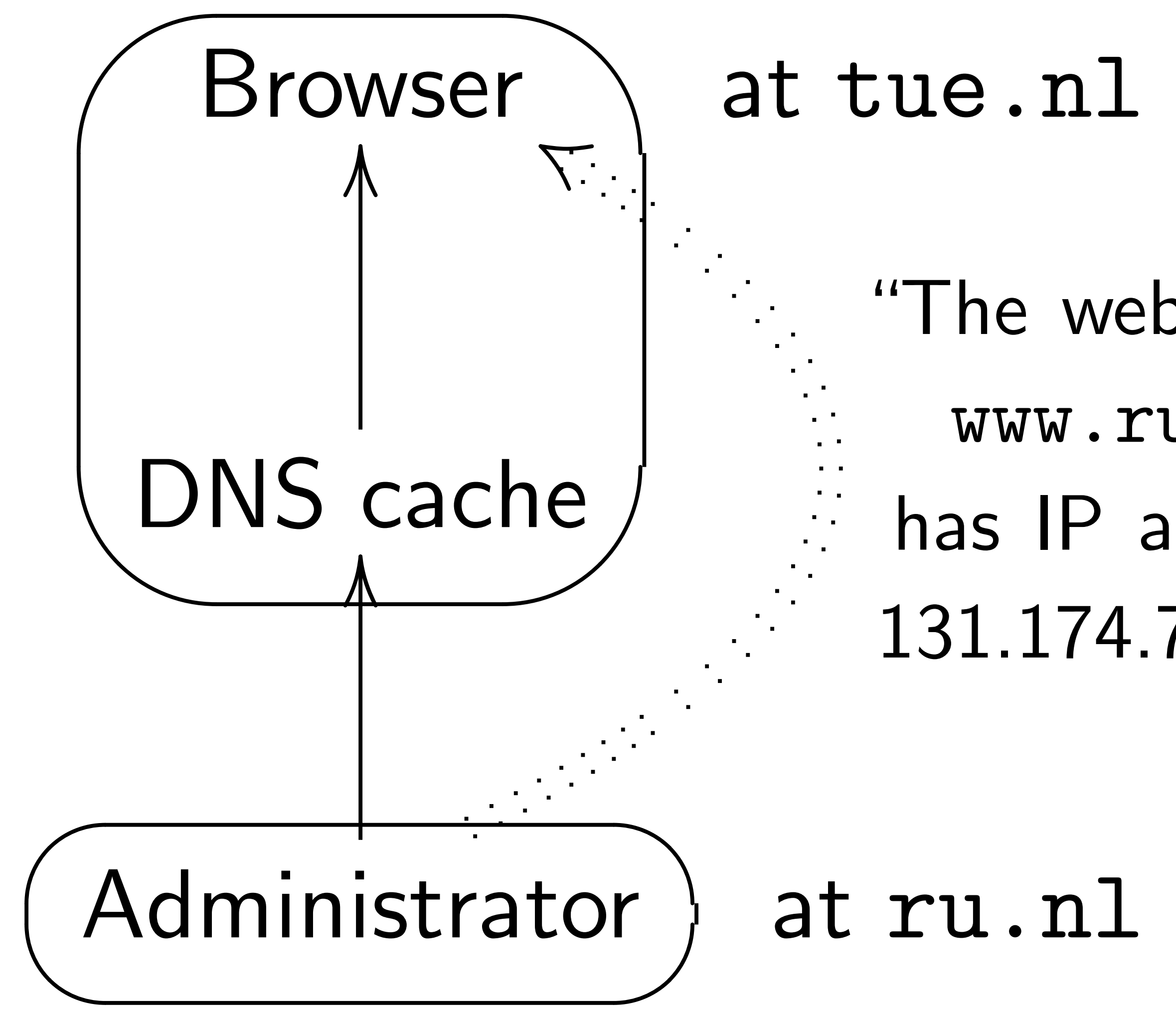
DNSSEC made breakable choices
such as 640-bit RSA
for no reason other than
fear of overload.

DNSSEC needed more options
to survive the inevitable breaks.
More complexity ⇒ more bugs,
including security holes.

Looking beyond the crypto:
Precomputation forced DNSSEC
down a path of unreliability,
insecurity, and unusability.
Let's see how this happened.

DNS architecture

Browser pulls data from
DNS cache at `tue.nl`:



Cache pulls data from
administrator if it
doesn't already have the dat

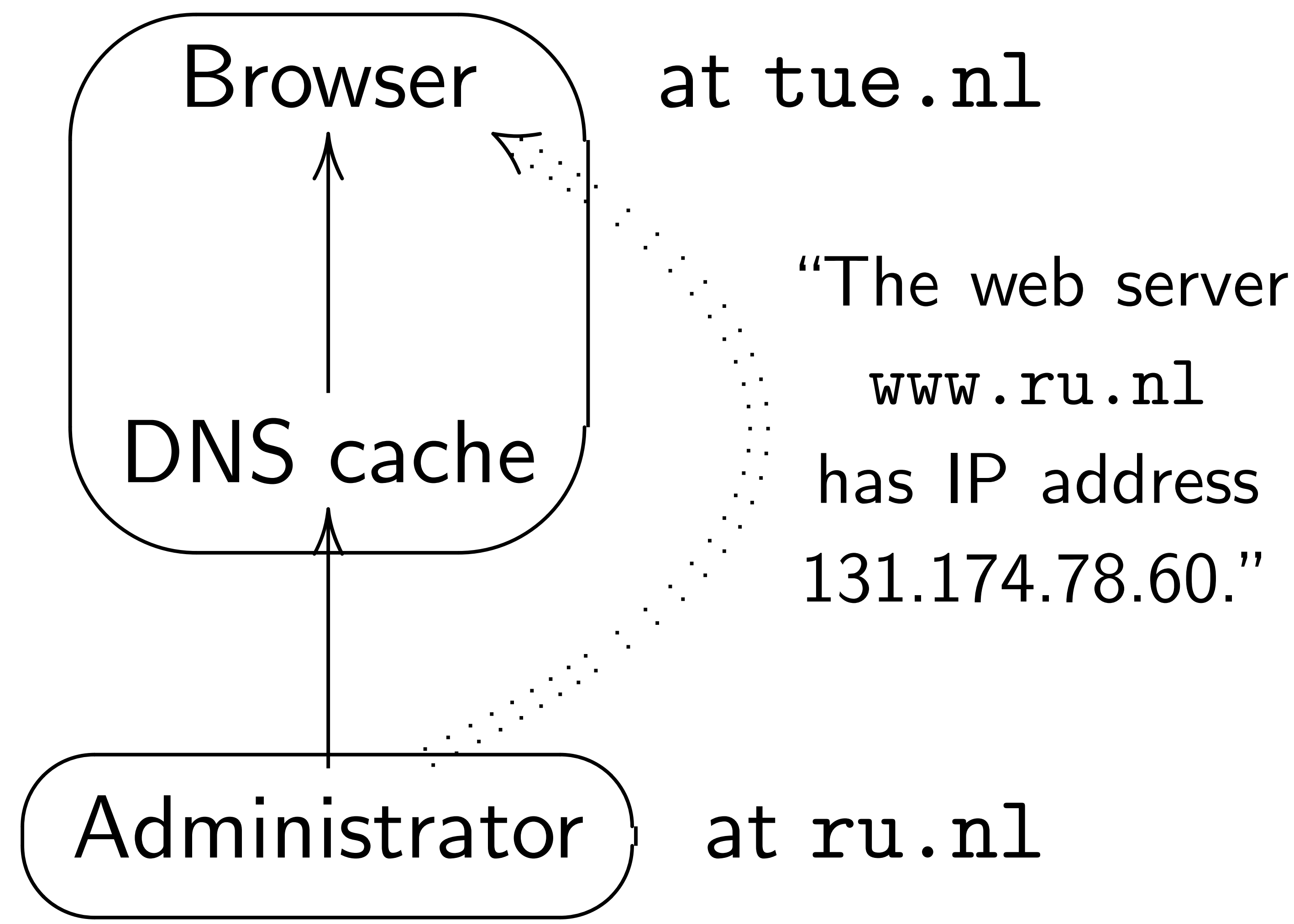
DNSSEC made breakable choices such as 640-bit RSA for no reason other than fear of overload.

DNSSEC needed more options to survive the inevitable breaks. More complexity ⇒ more bugs, including security holes.

Looking beyond the crypto: Precomputation forced DNSSEC down a path of unreliability, insecurity, and unusability. Let's see how this happened.

DNS architecture

Browser pulls data from DNS cache at `tue.nl`:



Cache pulls data from administrator if it doesn't already have the data.

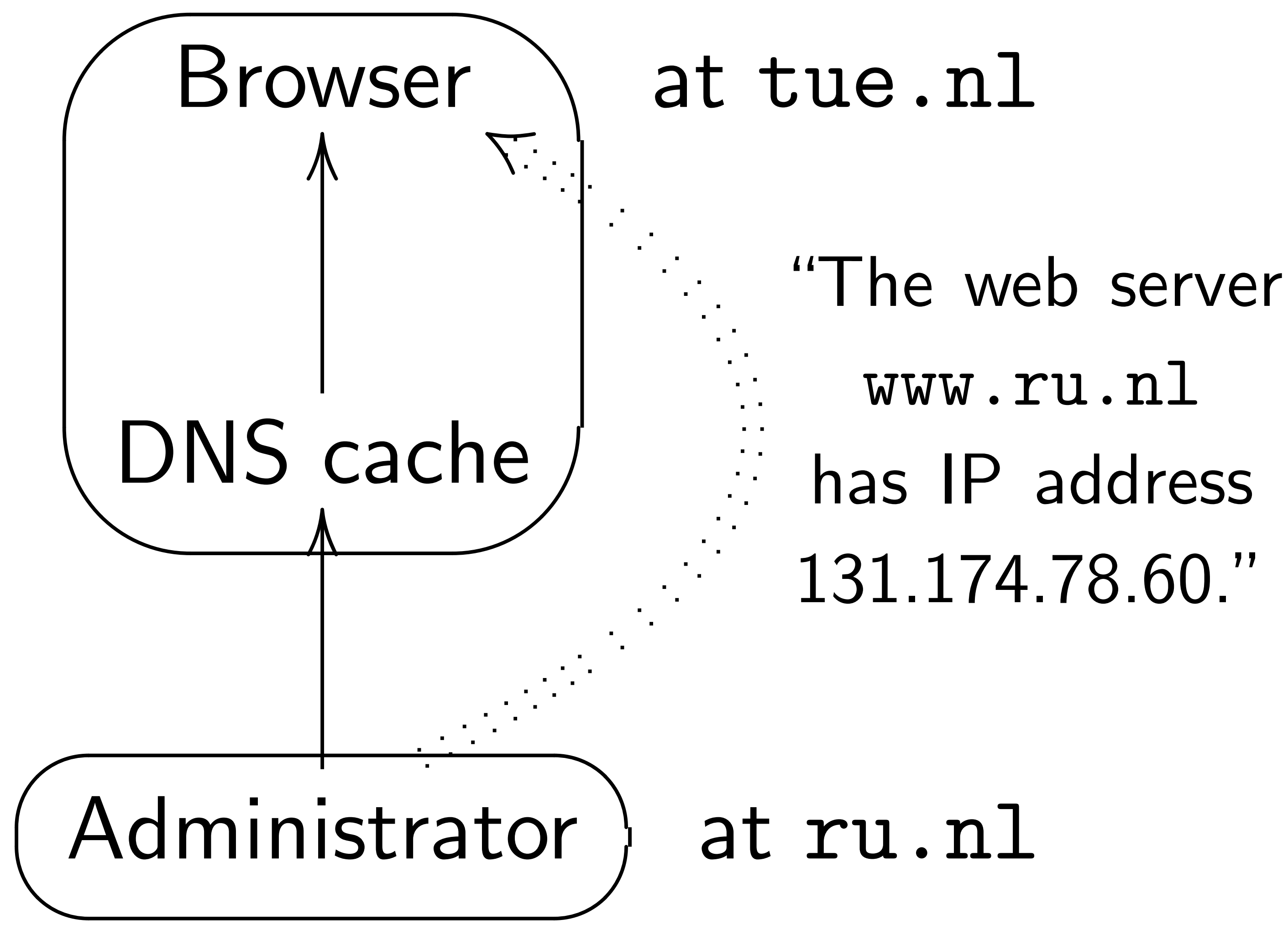
C made breakable choices
640-bit RSA
eason other than
overload.

C needed more options
ve the inevitable breaks.
mplexity ⇒ more bugs,
g security holes.

beyond the crypto:
putation forced DNSSEC
path of unreliability,
y, and unusability.
e how this happened.

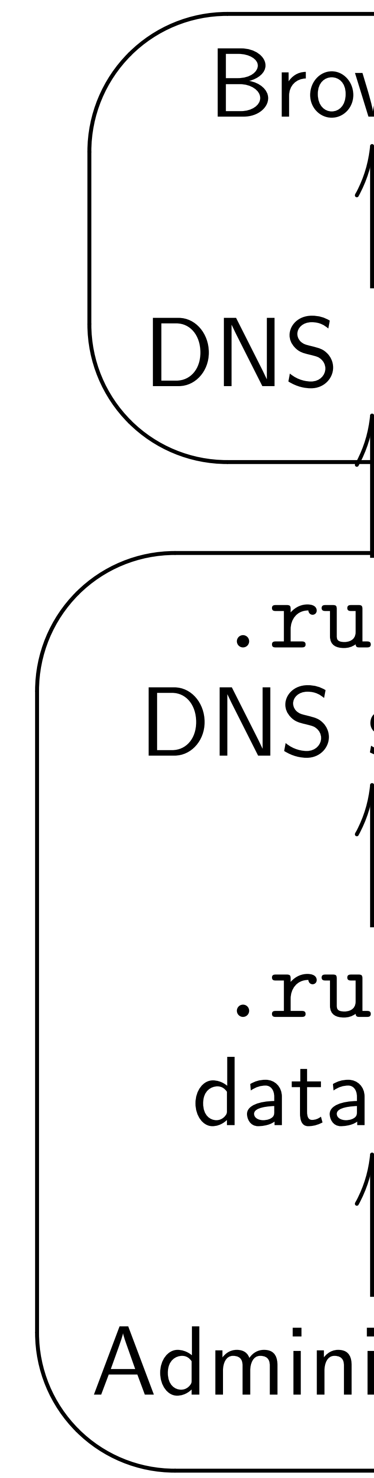
DNS architecture

Browser pulls data from
DNS cache at `tue.nl`:



Cache pulls data from
administrator if it
doesn't already have the data.

Adminis
through
`.ru.nl`



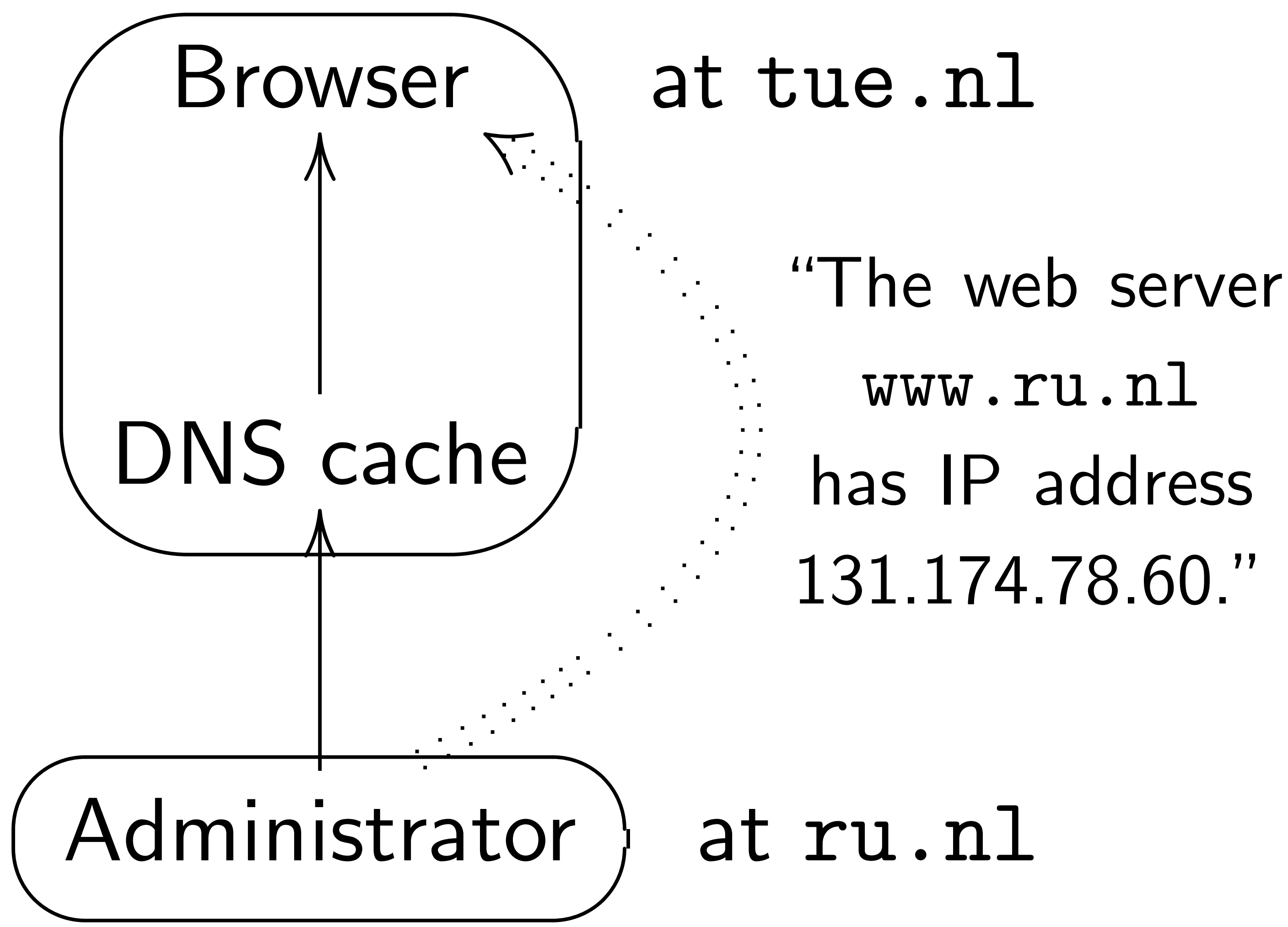
eakable choices
SA
r than

more options
vitable breaks.
⇒ more bugs,
holes.

he crypto:
orced DNSSEC
nreliability,
usability.
happened.

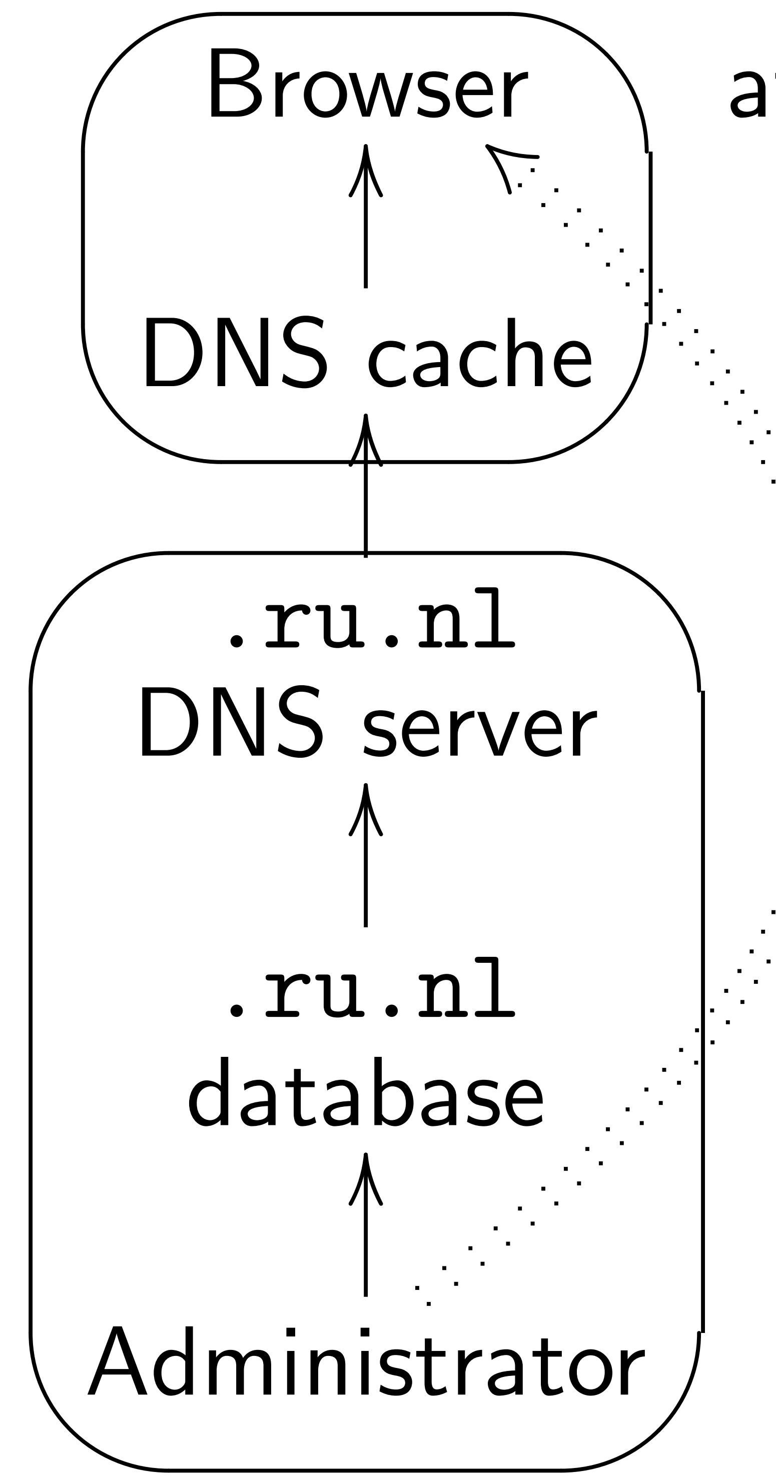
DNS architecture

Browser pulls data from
DNS cache at `tue.nl`:



Cache pulls data from
administrator if it
doesn't already have the data.

Administrator pus
through local data
`.ru.nl` DNS serv



hoices

ons

aks.

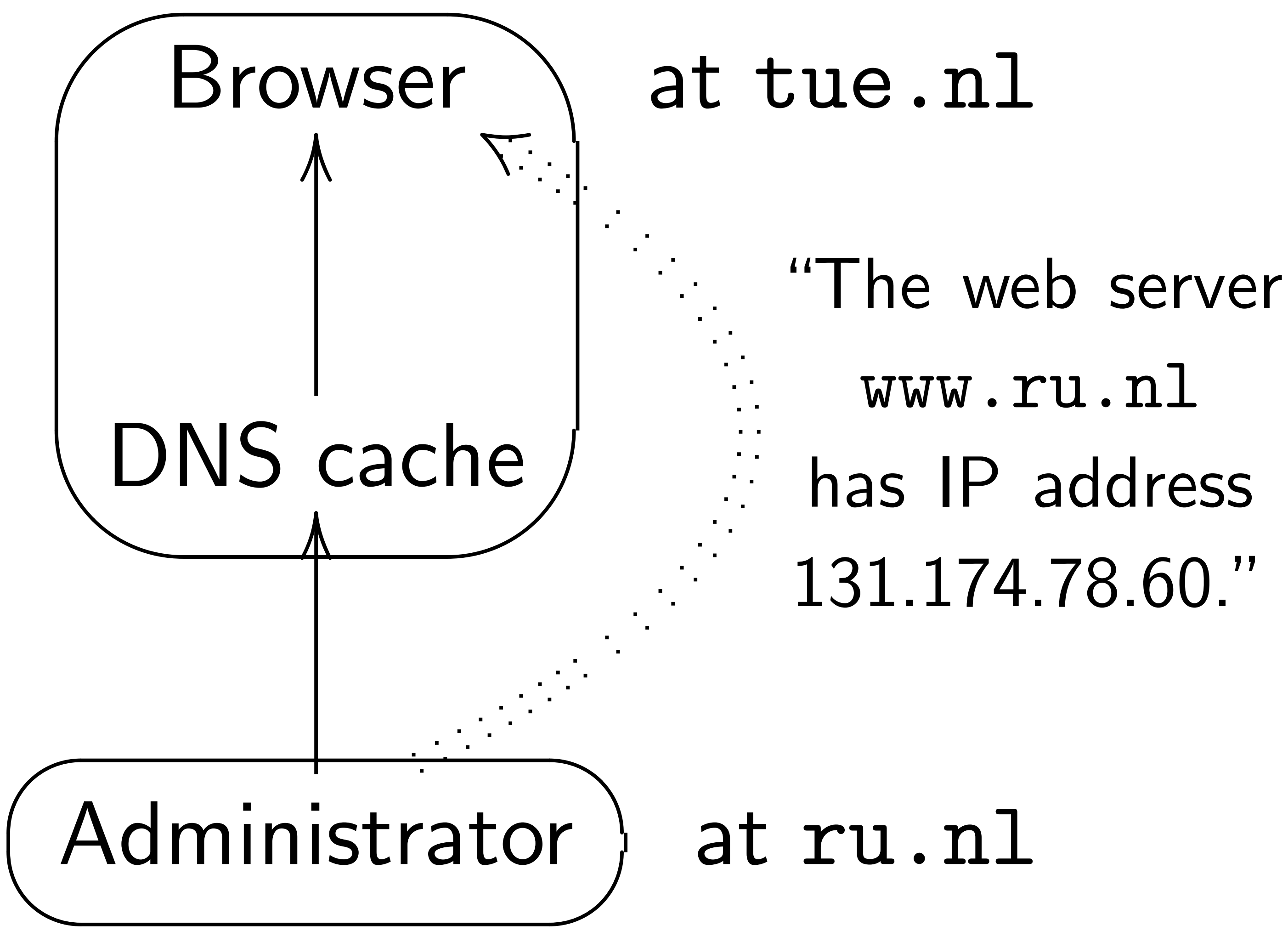
ugs,

SSEC

.

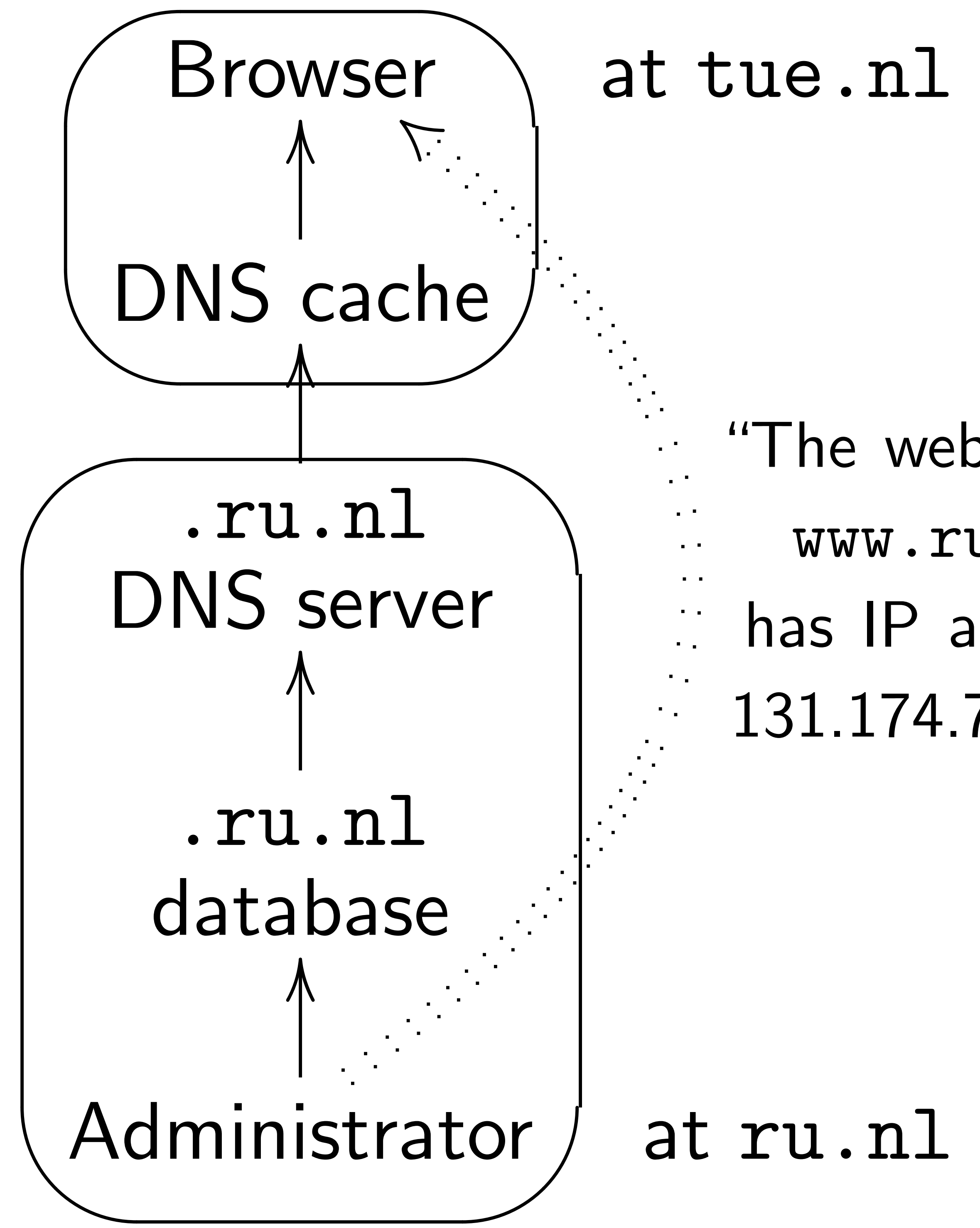
DNS architecture

Browser pulls data from DNS cache at `tue.nl`:



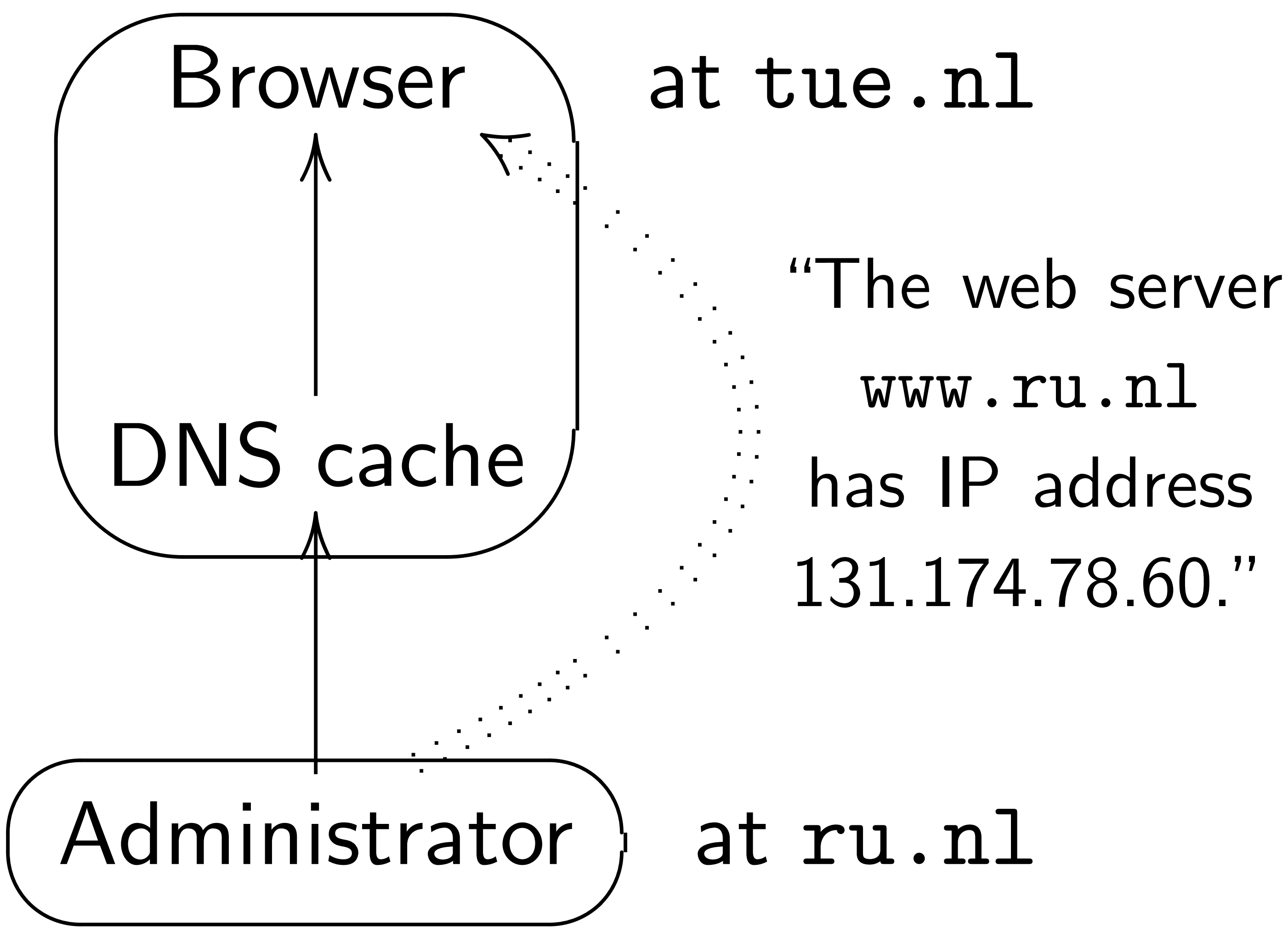
Cache pulls data from administrator if it doesn't already have the data.

Administrator pushes data through local database into `.ru.nl` DNS server:



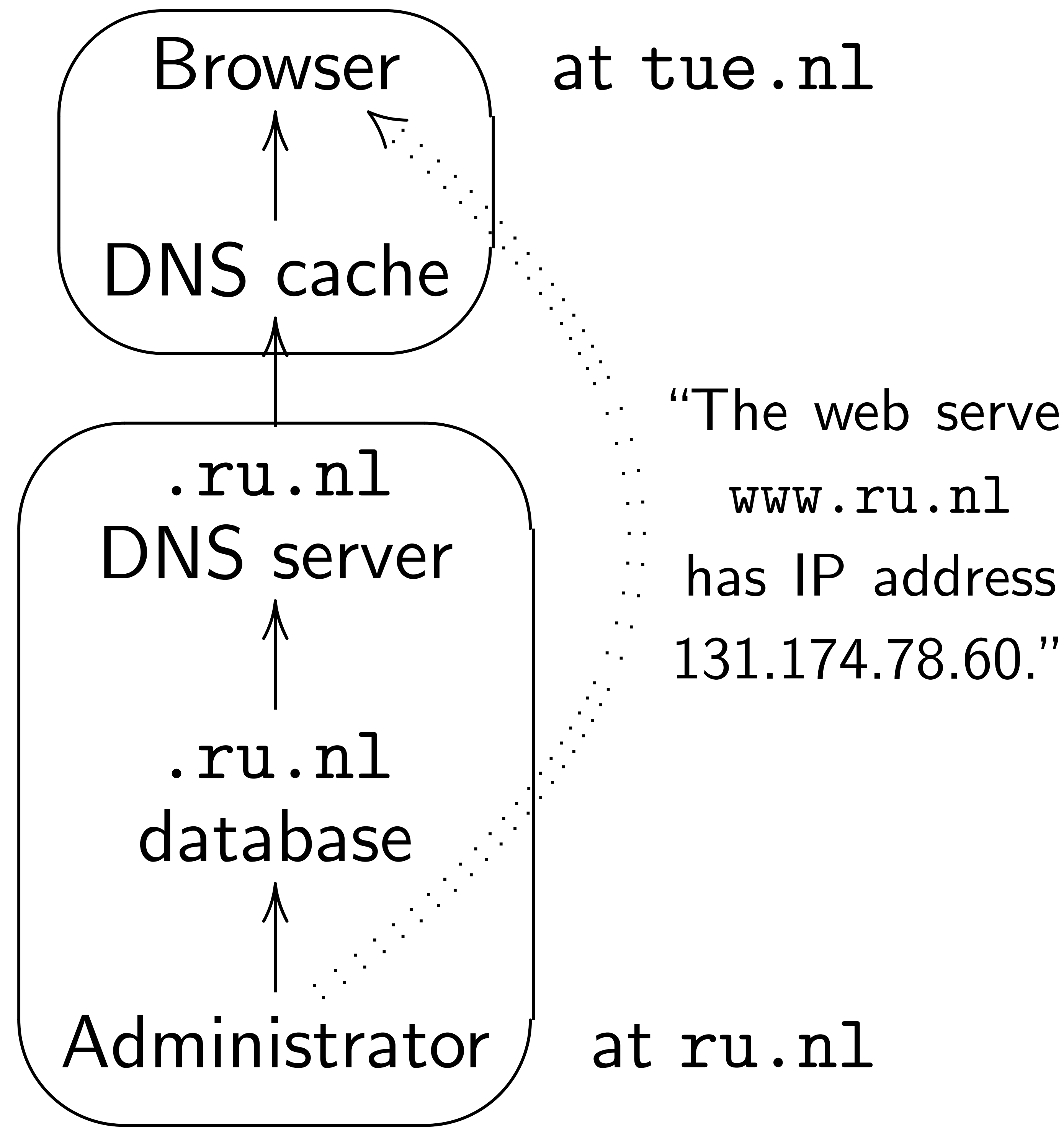
DNS architecture

Browser pulls data from DNS cache at `tue.nl`:



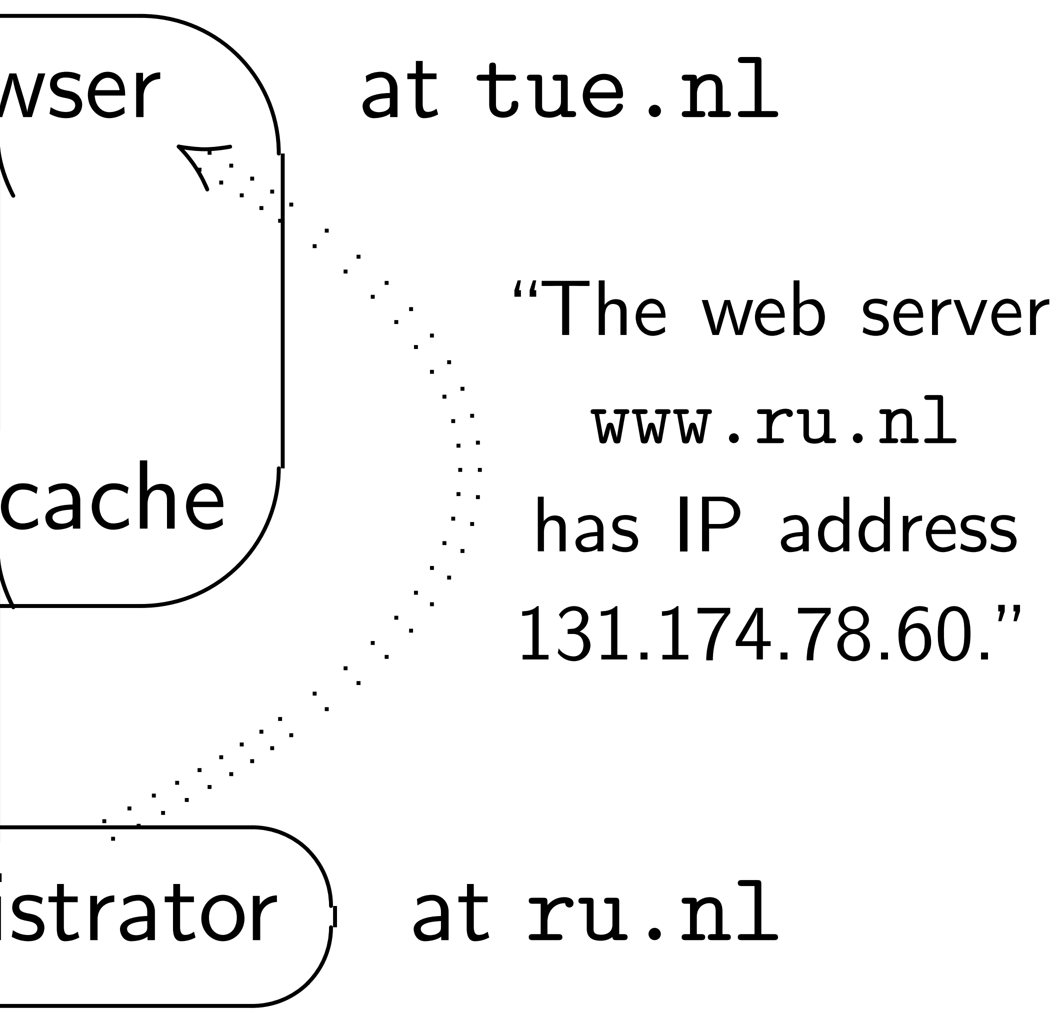
Cache pulls data from administrator if it doesn't already have the data.

Administrator pushes data through local database into `.ru.nl` DNS server:



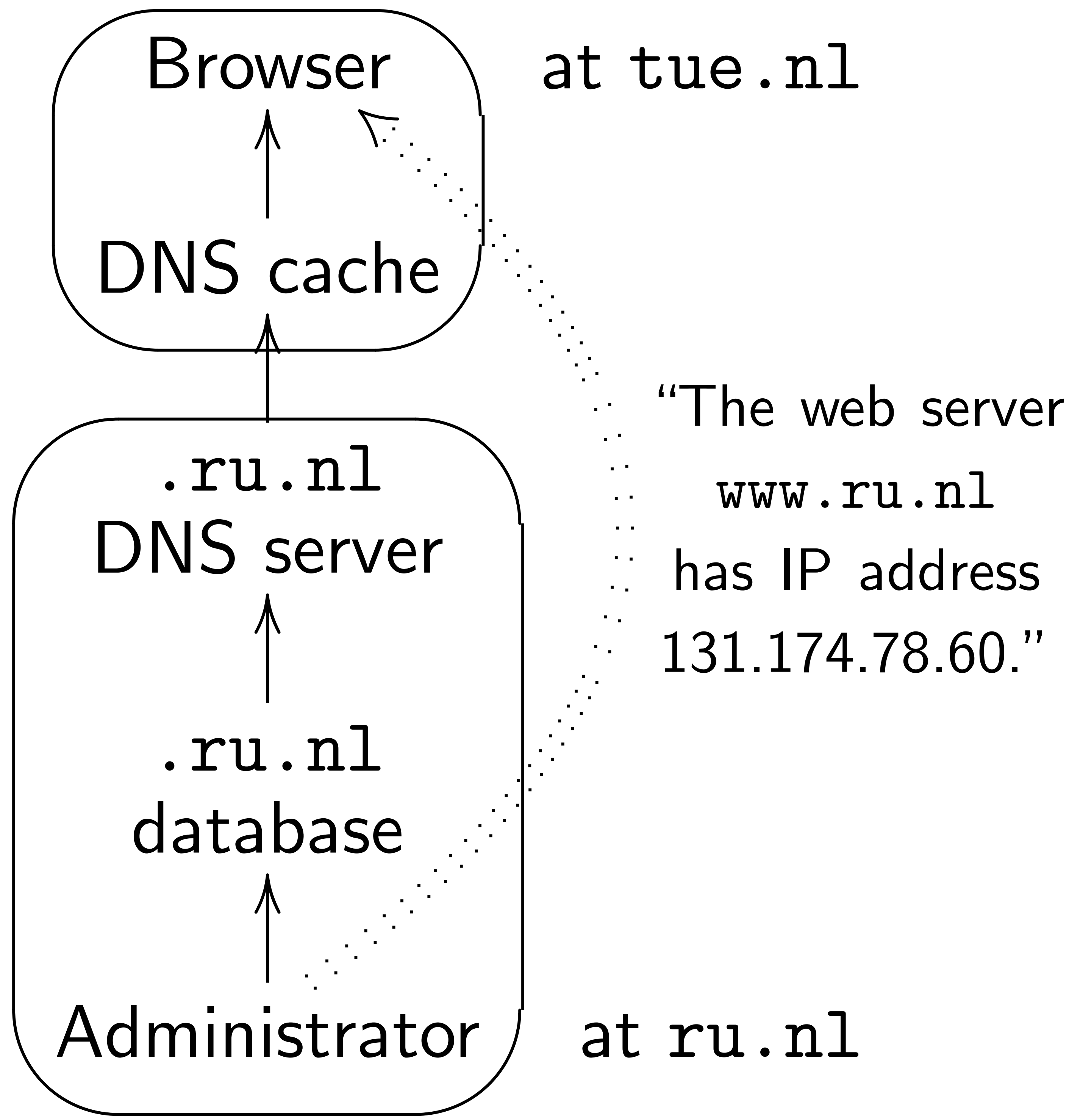
chitecture

pulls data from
che at `tue.nl`:



ulls data from
trator if it
already have the data.

Administrator pushes data
through local database into
`.ru.nl` DNS server:



DNS cac
`.ru.nl`
`.nl` DN

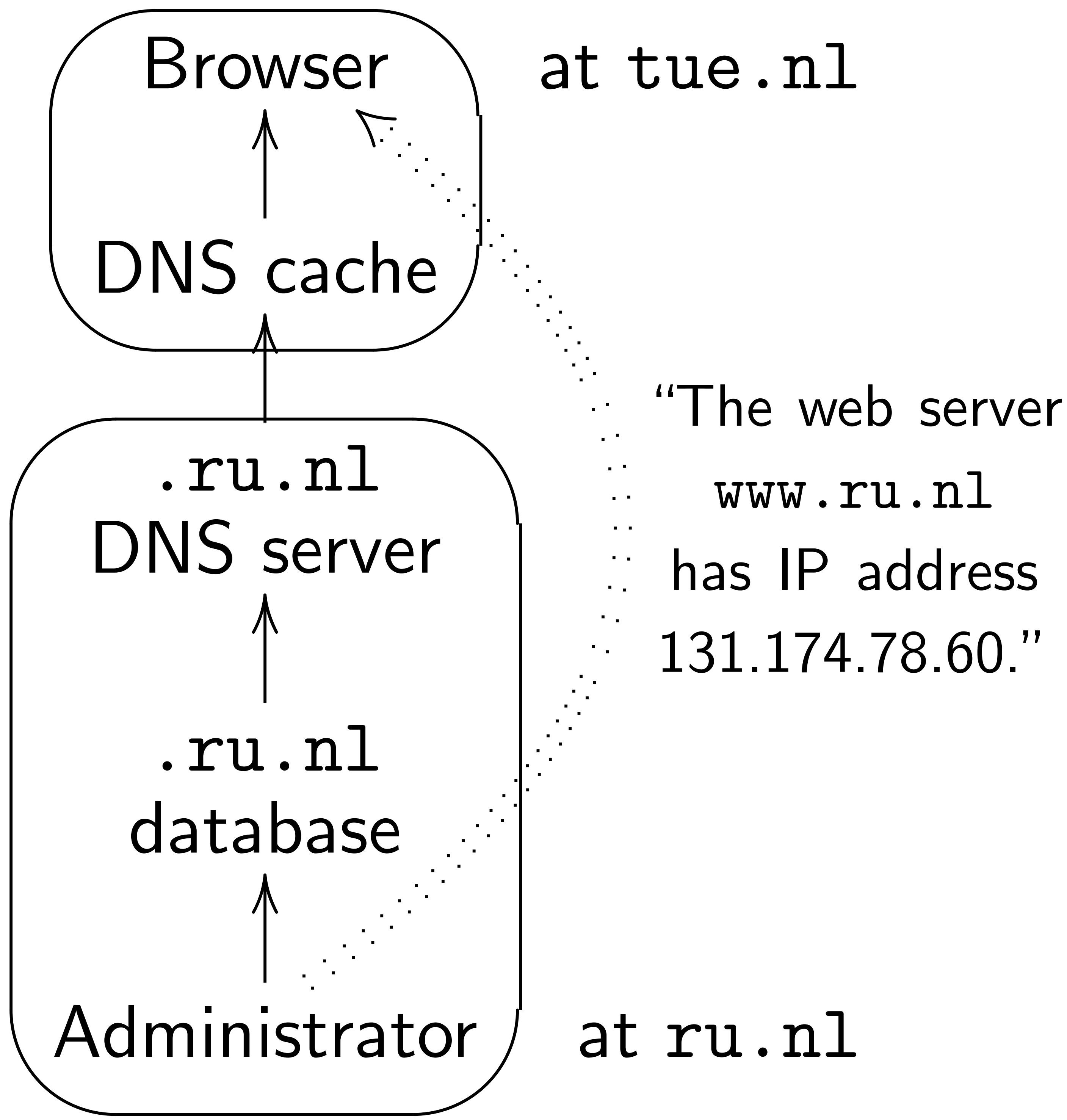
“The DN
for .r
is n
with IP
131.174.

n from
e.nl:

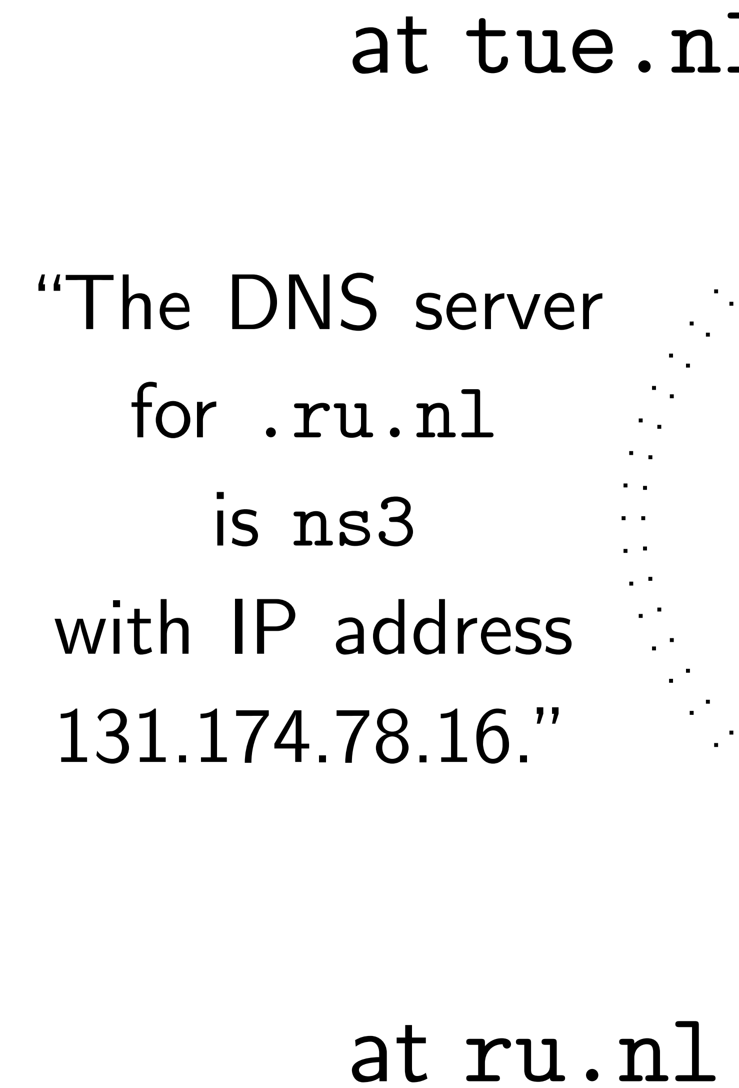
rom

ve the data.

Administrator pushes data through local database into .ru.nl DNS server:



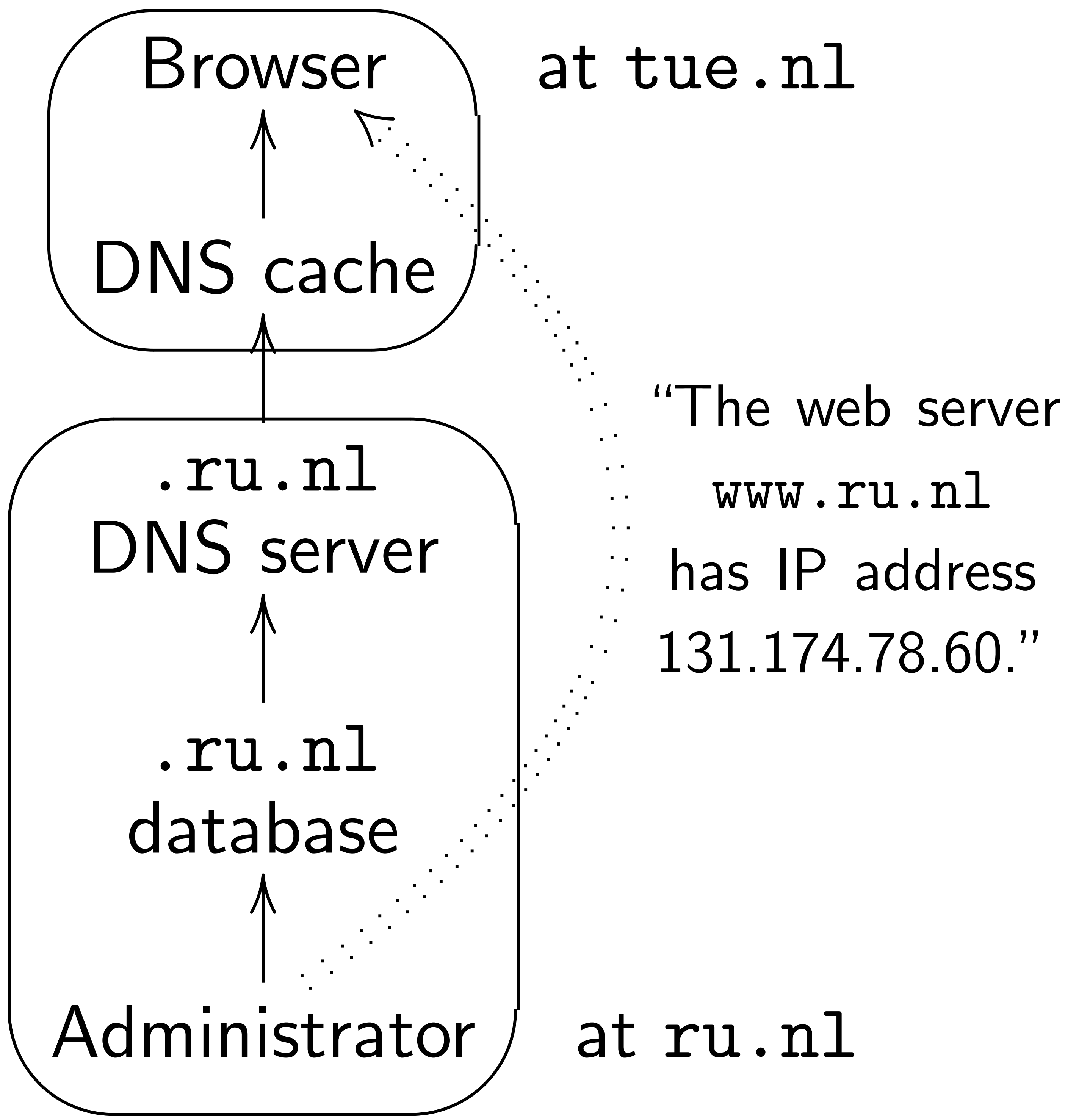
DNS cache learns
.ru.nl DNS serve
.nl DNS server:



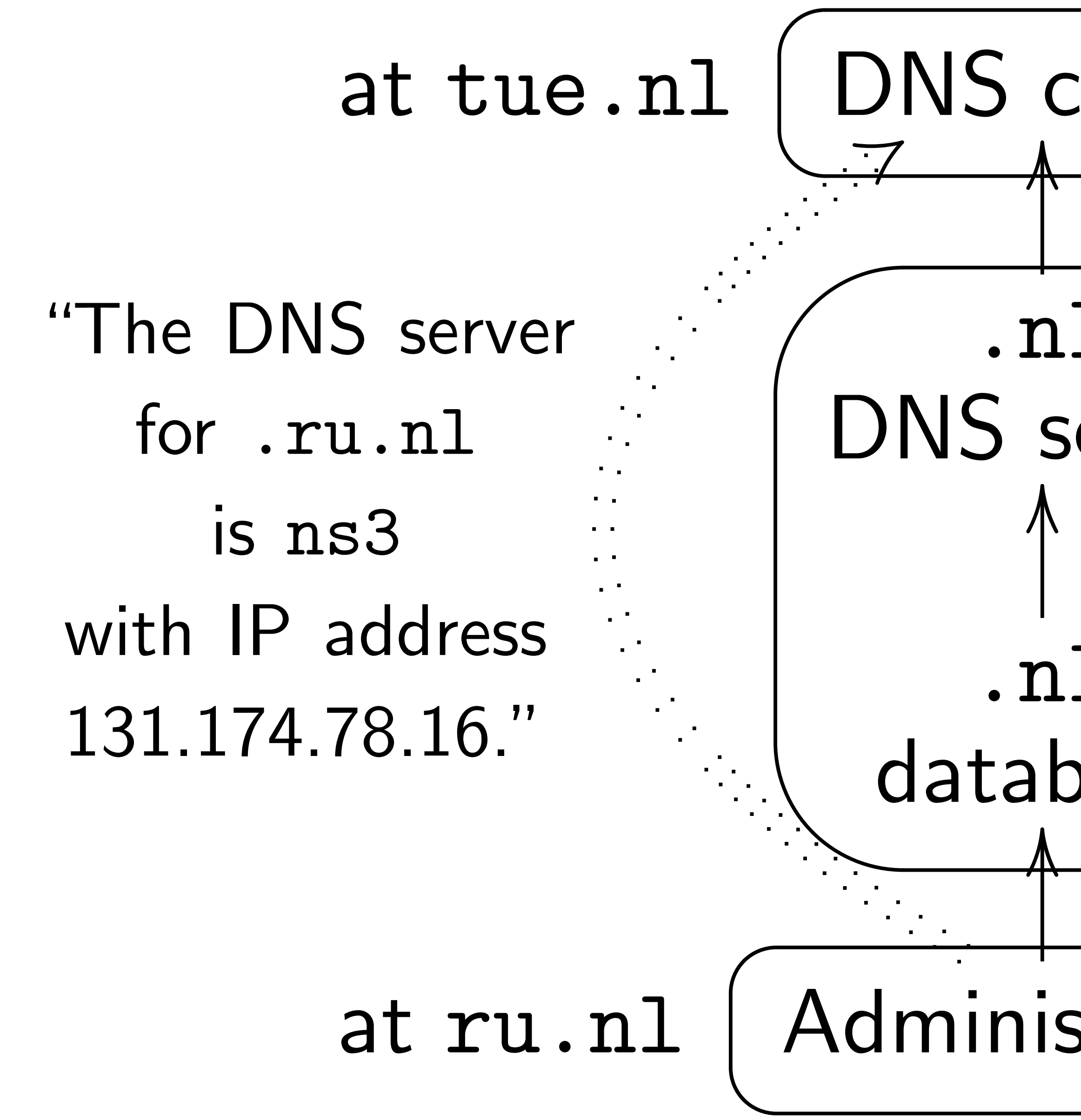
server
u.nl
ddress
78.60.”

ta.

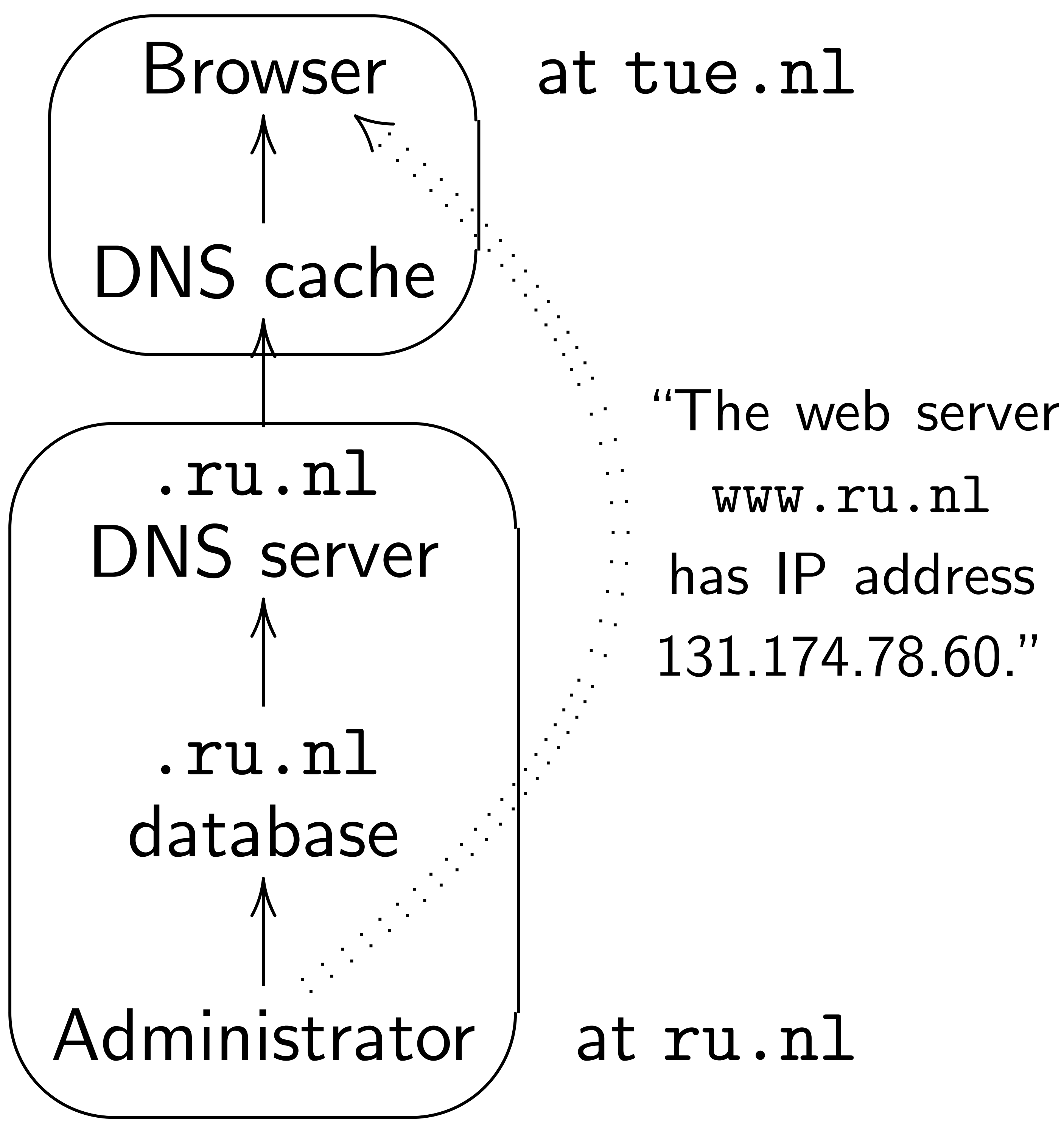
Administrator pushes data through local database into .ru.nl DNS server:



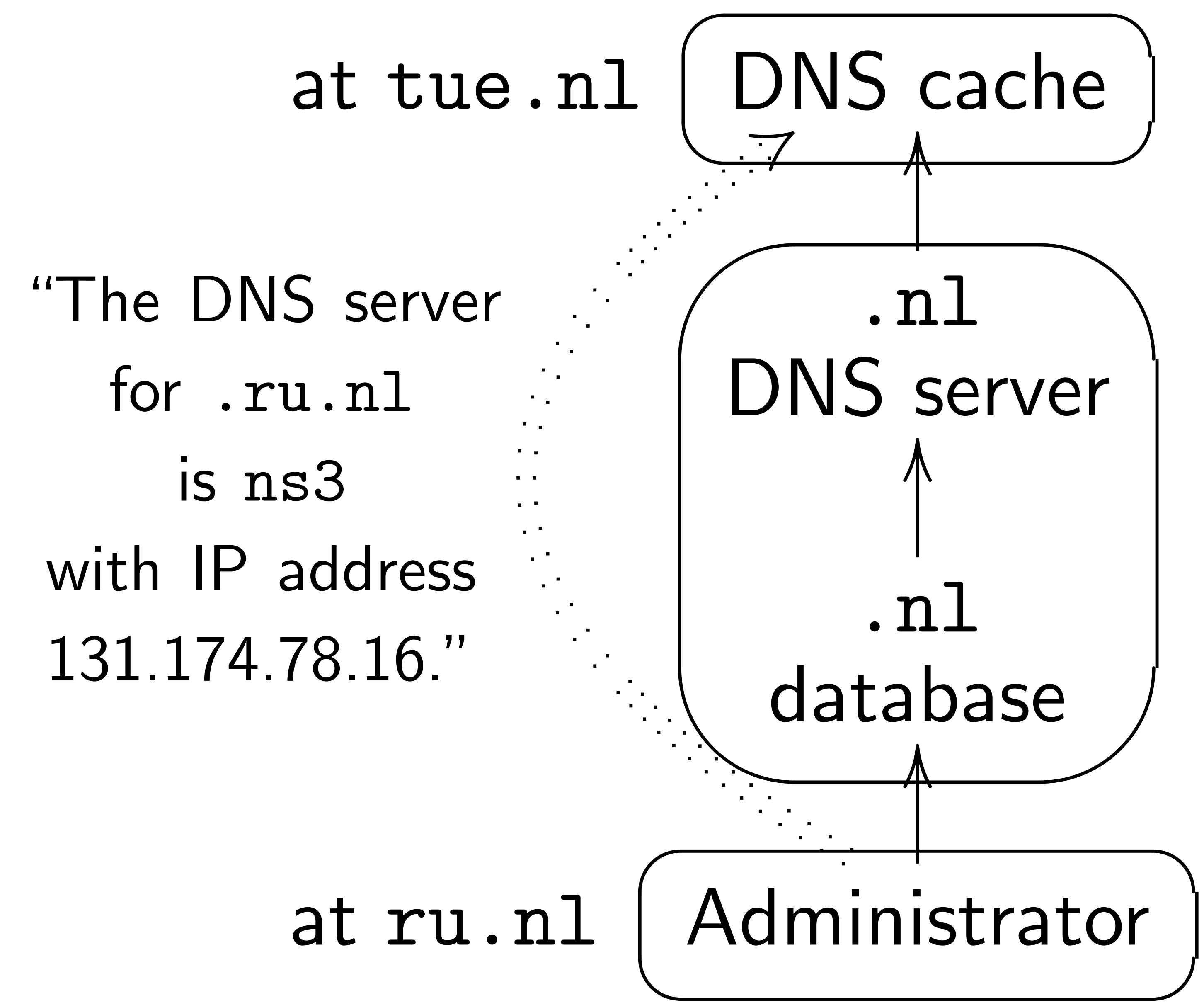
DNS cache learns location o
.ru.nl DNS server from
.nl DNS server:



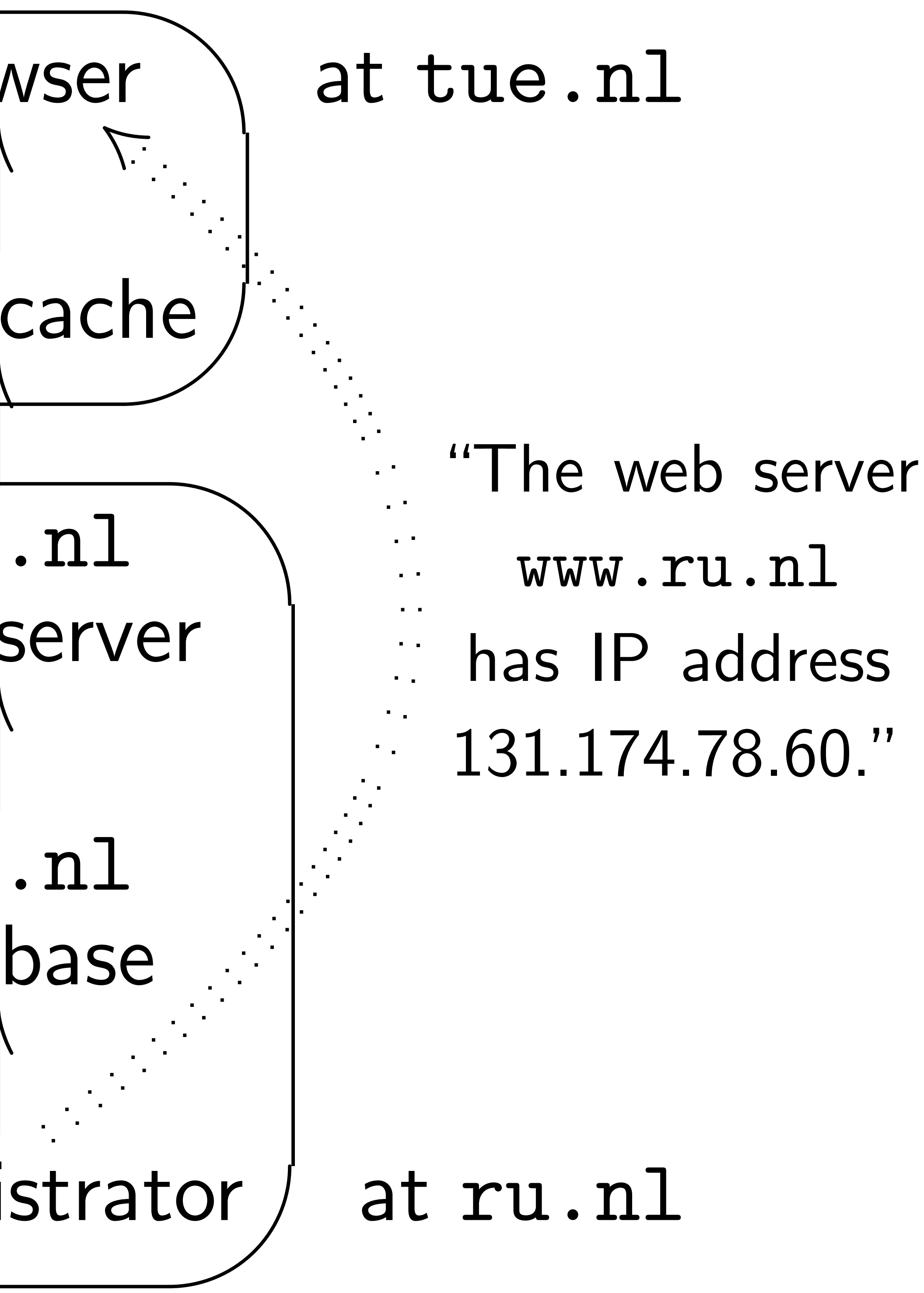
Administrator pushes data through local database into `.ru.nl` DNS server:



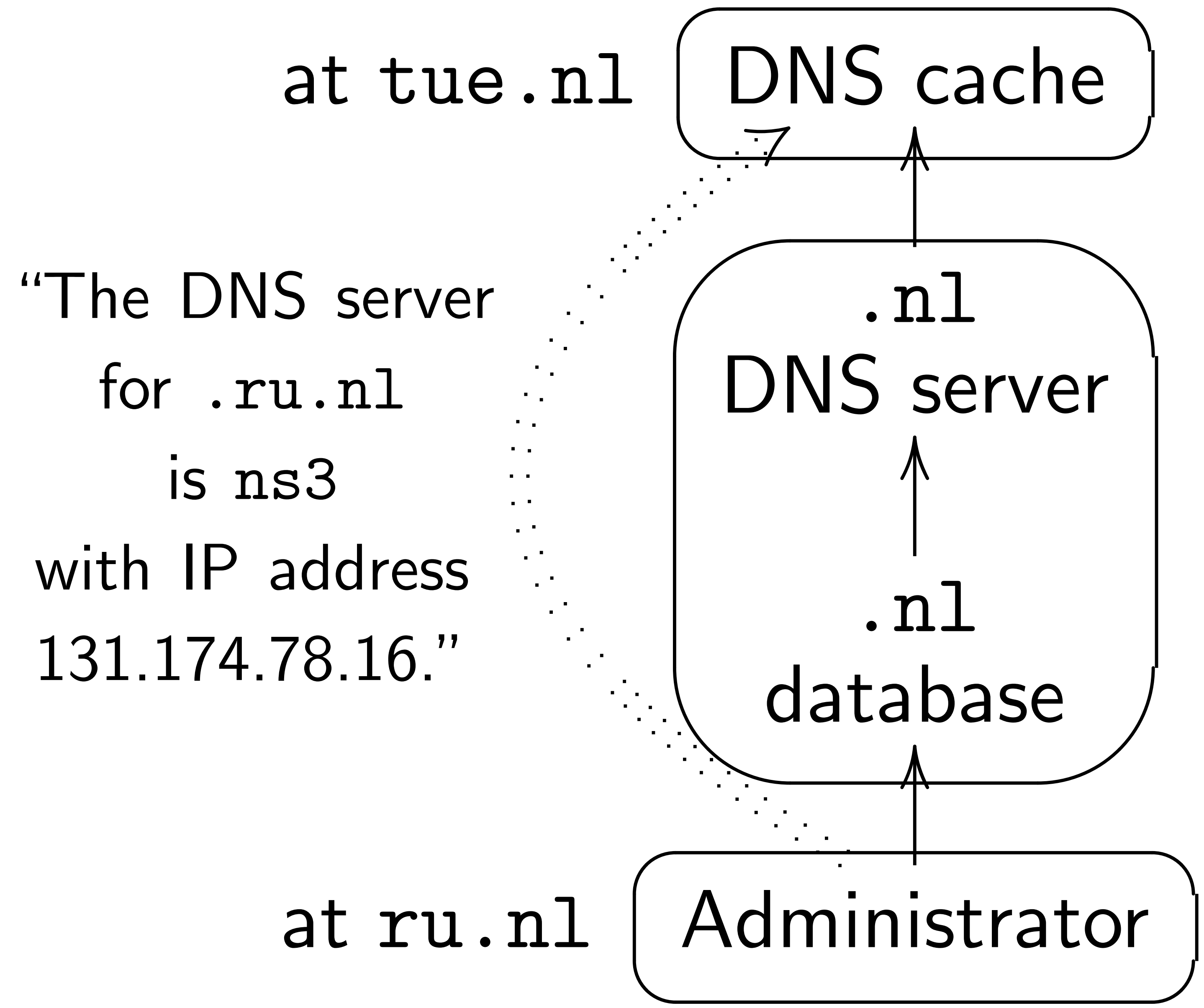
DNS cache learns location of `.ru.nl` DNS server from `.nl` DNS server:

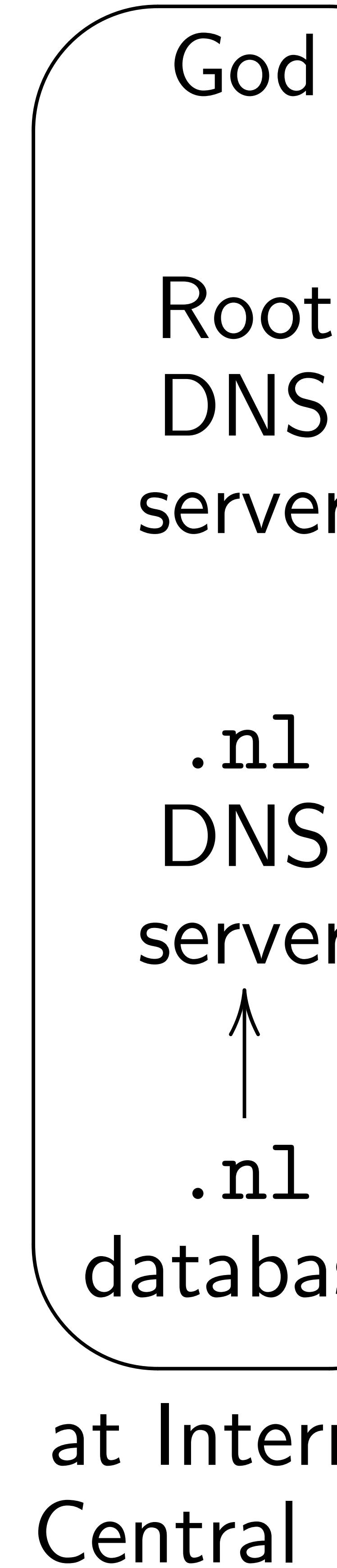


trator pushes data
local database into
DNS server:



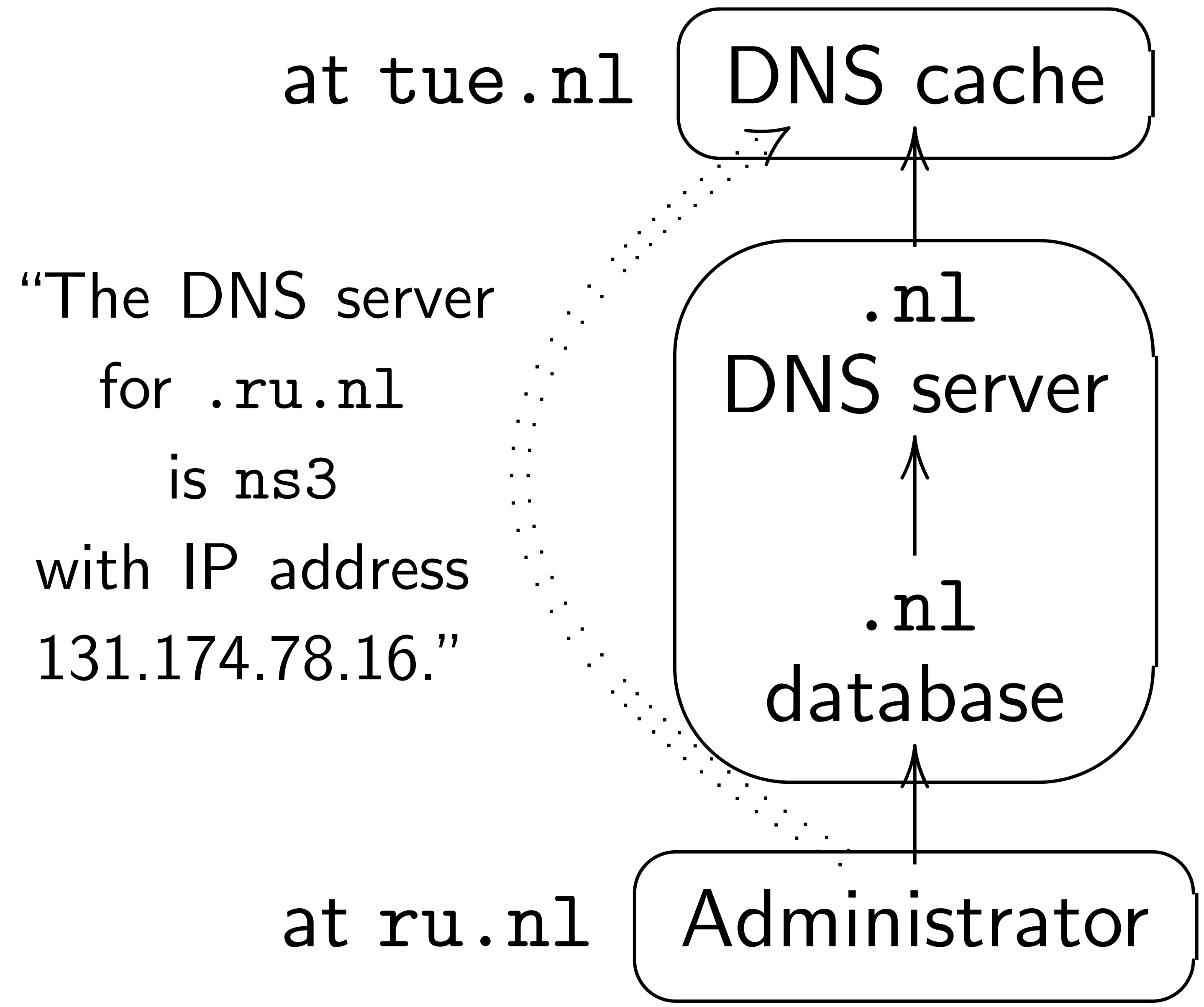
DNS cache learns location of
.ru.nl DNS server from
.nl DNS server:

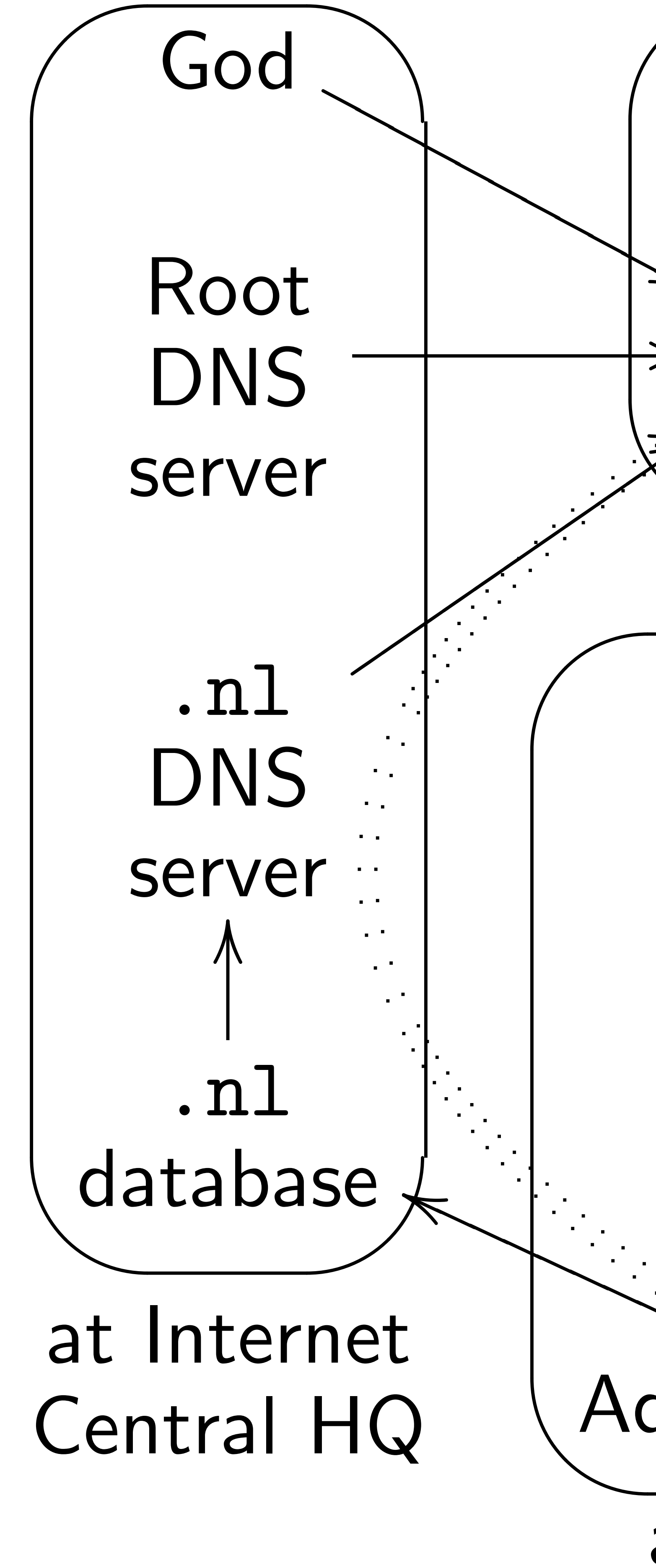




hes data
base into
er:

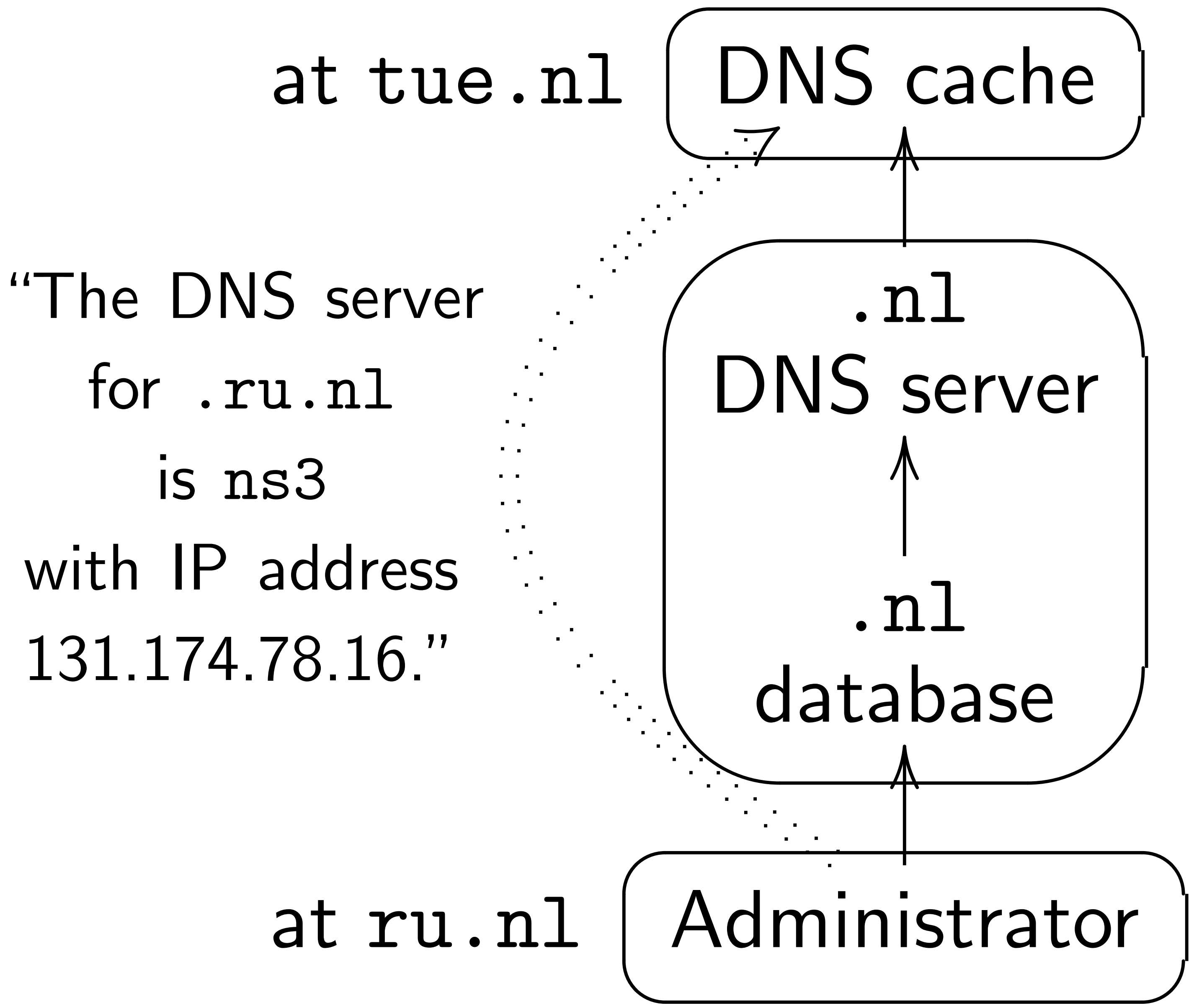
DNS cache learns location of
`.ru.nl` DNS server from
`.nl` DNS server:

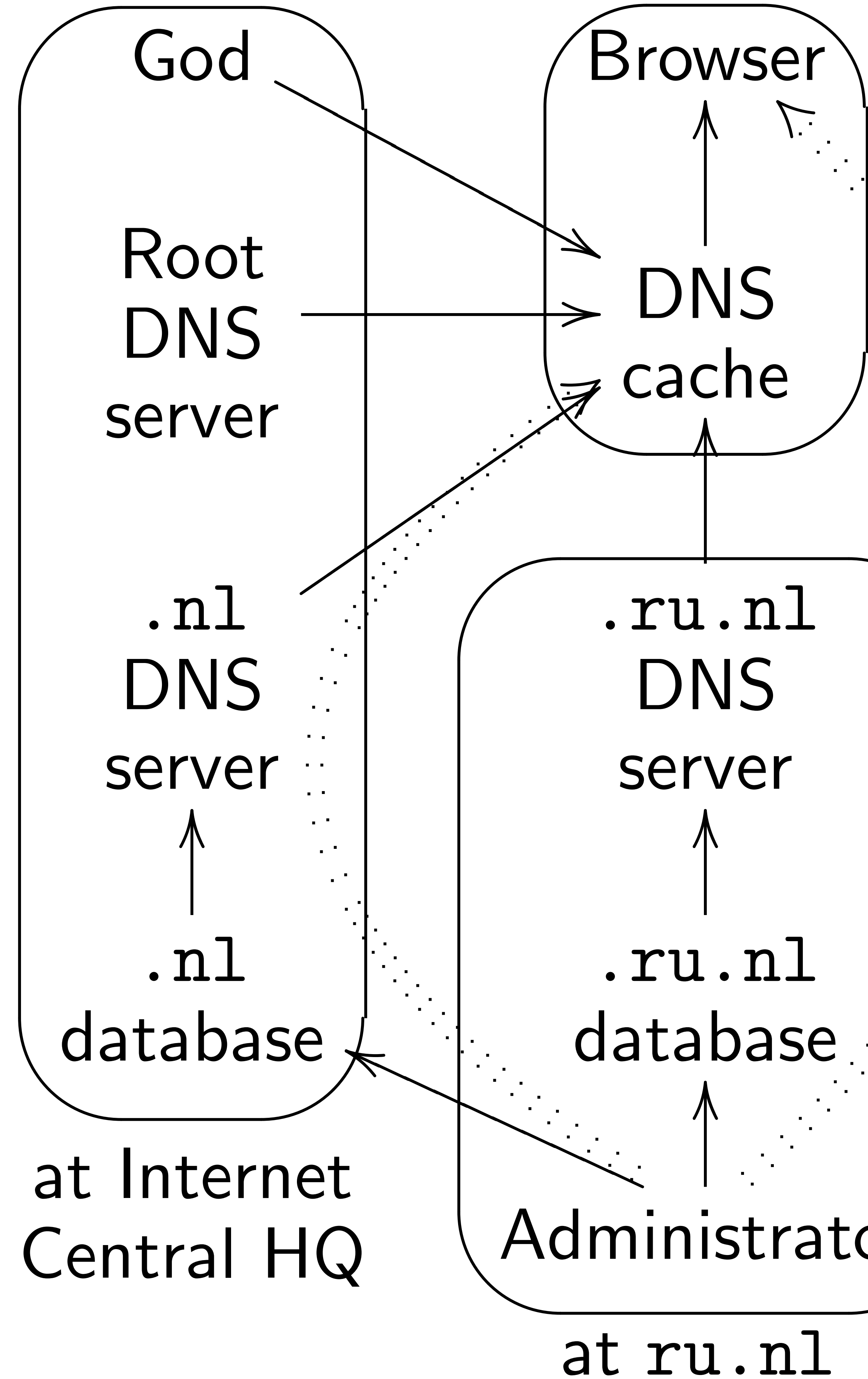




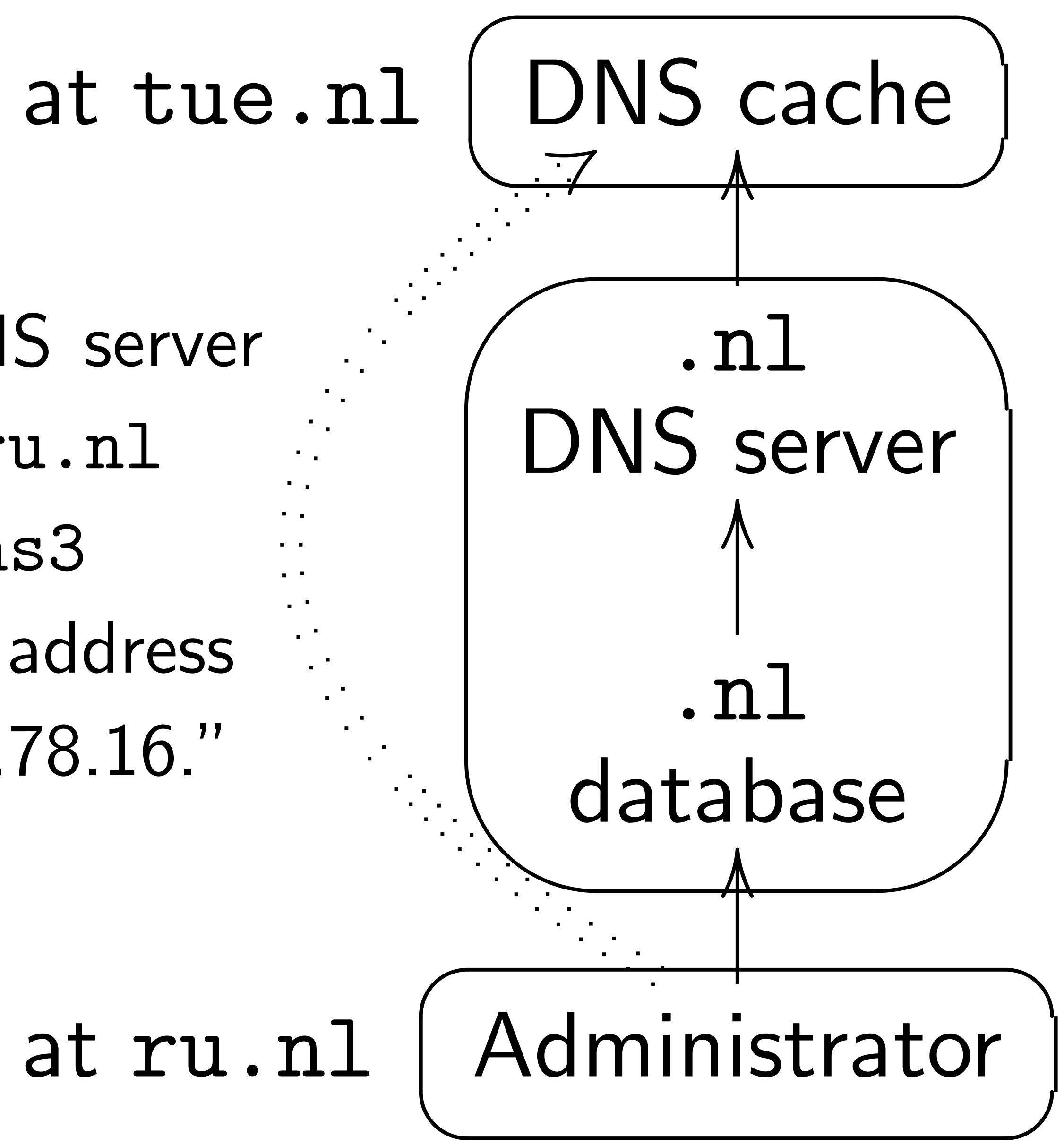
o server
u.nl
ddress
78.60."

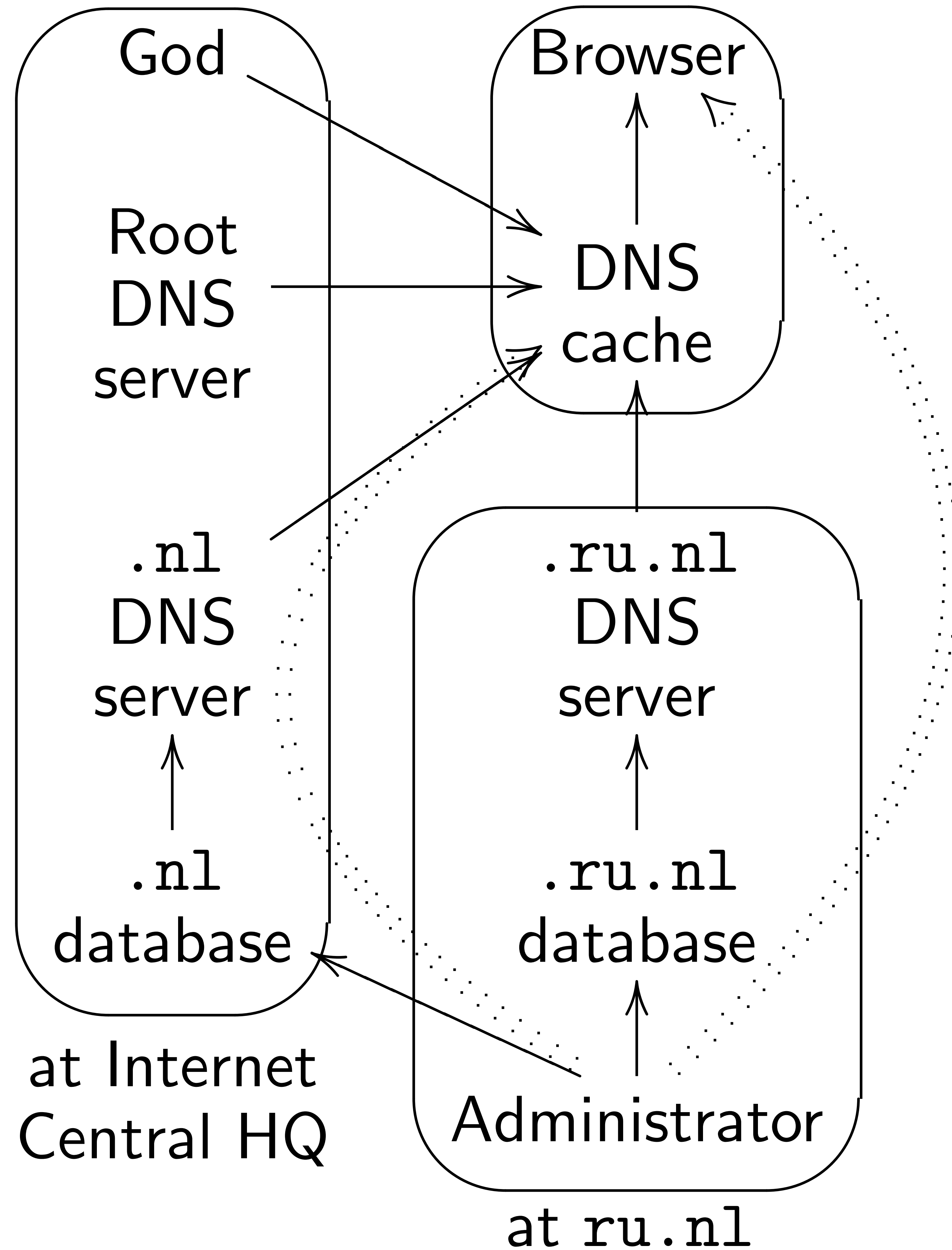
DNS cache learns location of `.ru.nl` DNS server from `.nl` DNS server:





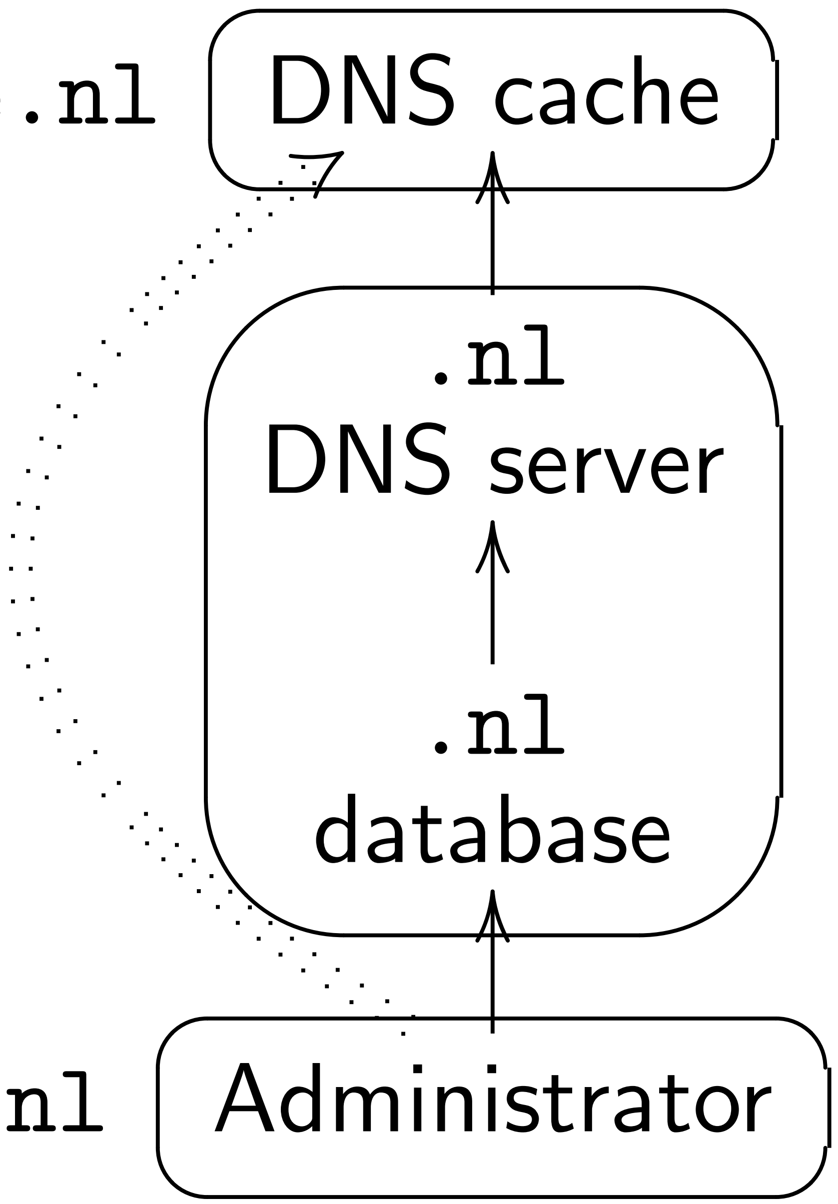
DNS cache learns location of `.ru.nl` DNS server from `.nl` DNS server:

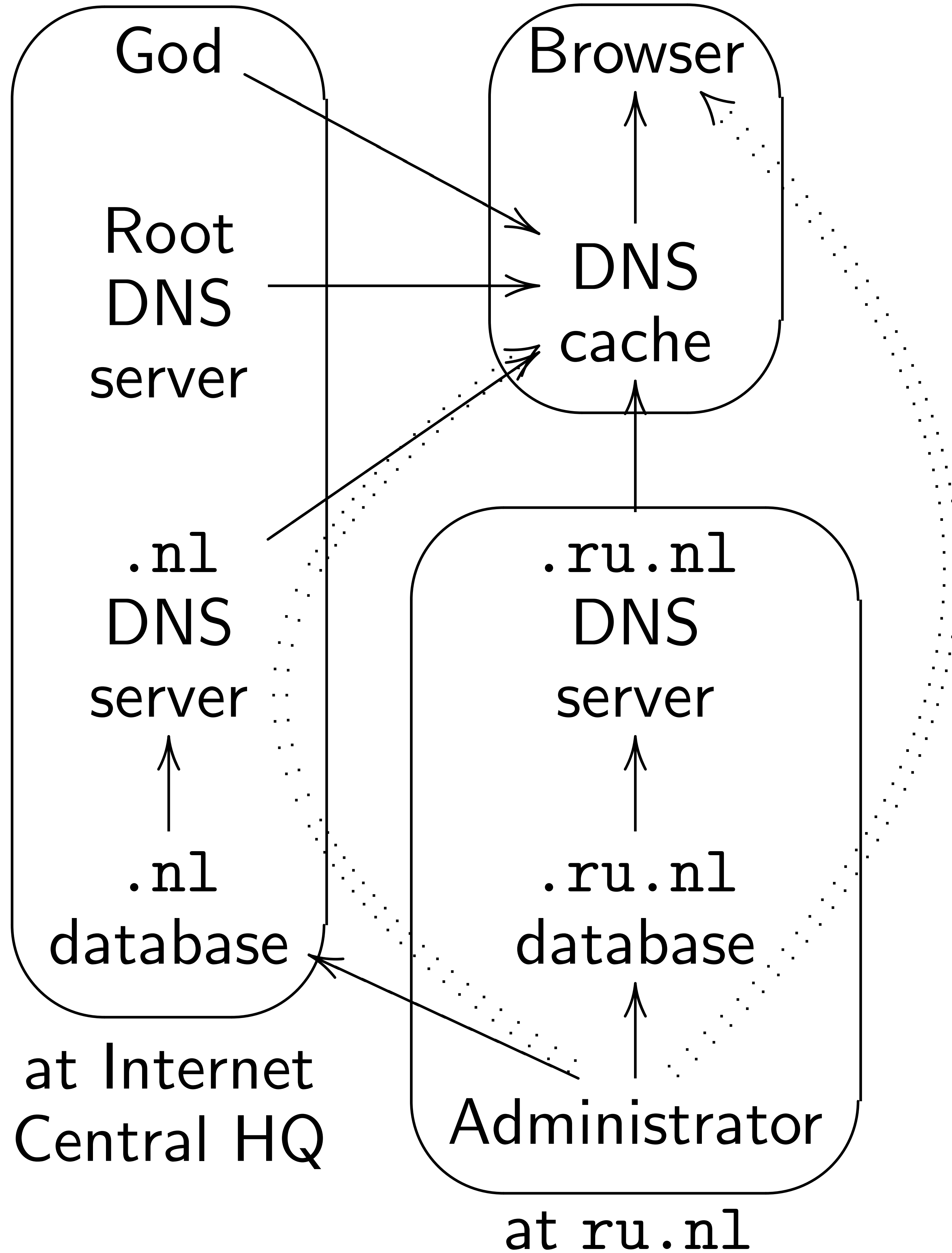




che learns location of
DNS server from
S server:

S server
u.nl
s3
address
78.16.”

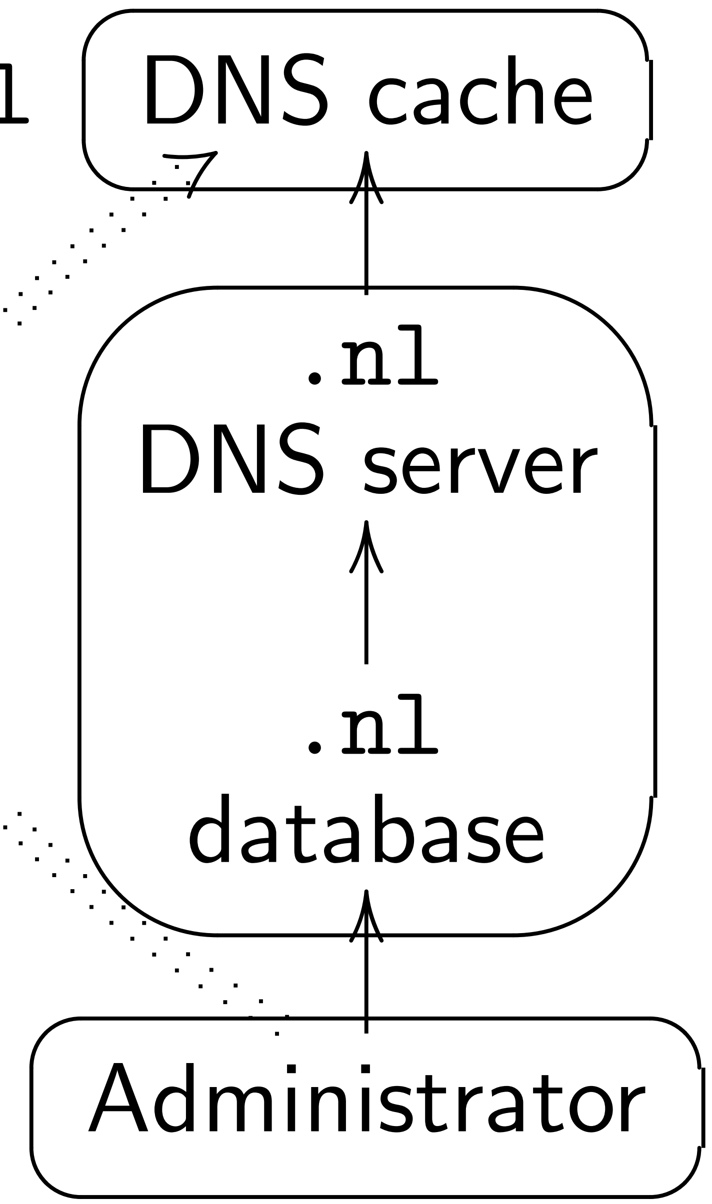


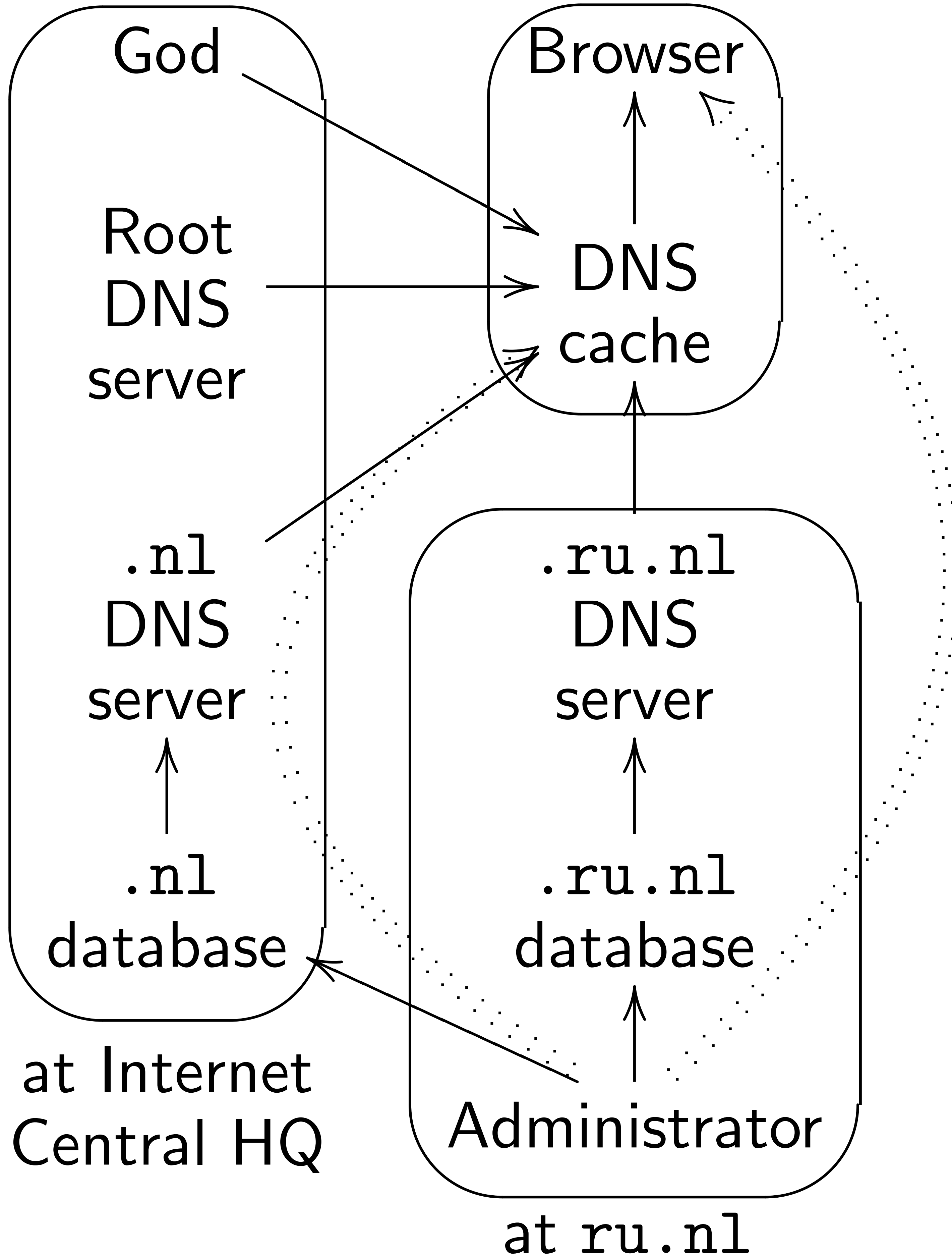


DNS ser
Wikiped
DNS, dj
DNS Plu
PowerDI
Nominur
Posadis,
Registra
yaku-ns,

Much w
database
hundreds
written

location of
er from

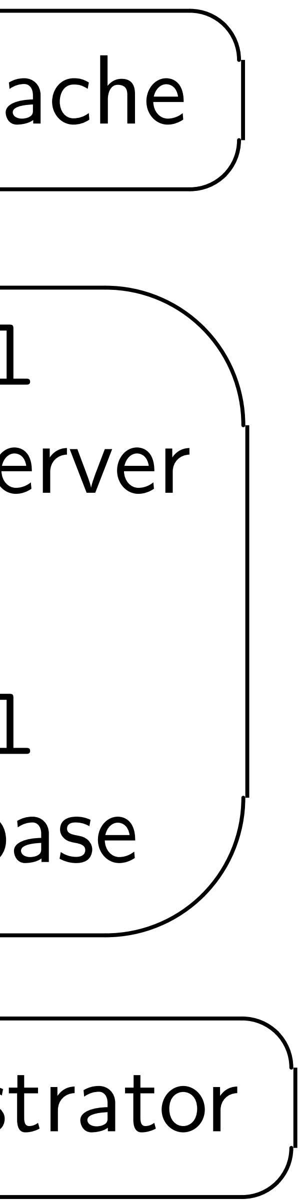


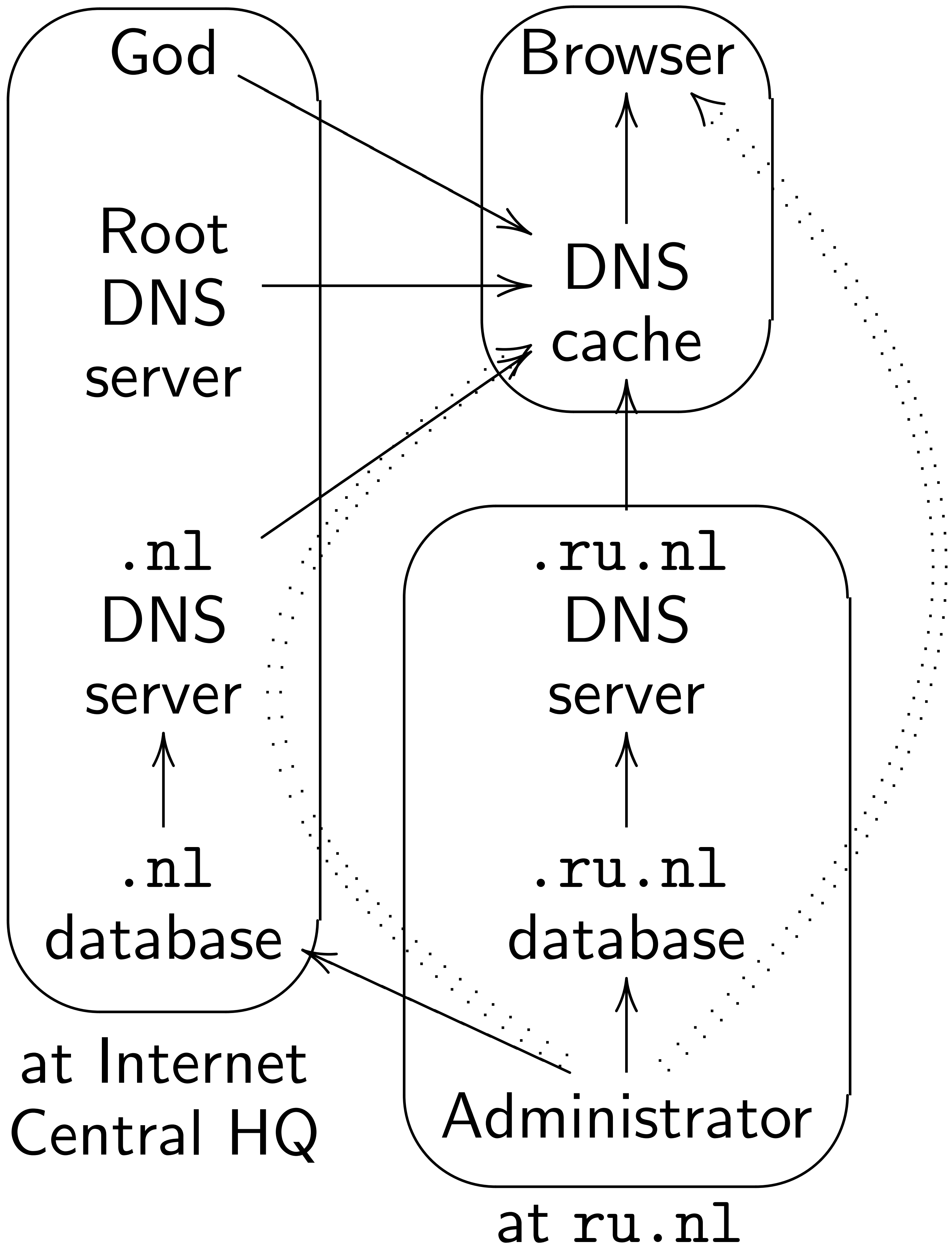


DNS server softwa
Wikipedia: BIND,
DNS, djbdns, Dns
DNS Plus, NSD, K
PowerDNS, MaraD
Nominum ANS, N
Posadis, Unbound
Registrar, dnrd, go
yaku-ns, DNS Blas

Much wider variety
database-managem
hundreds of homeg
written by DNS re

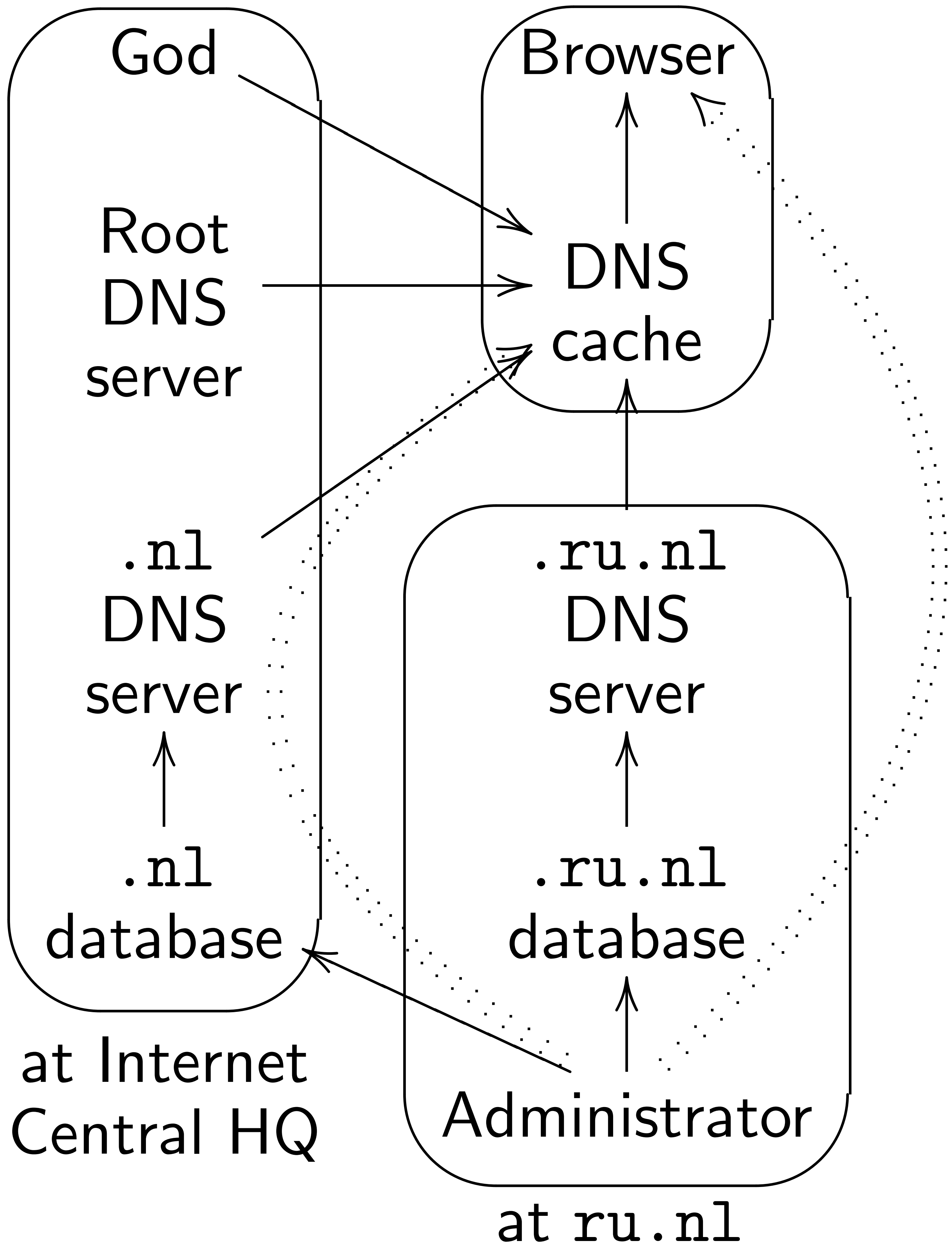
of





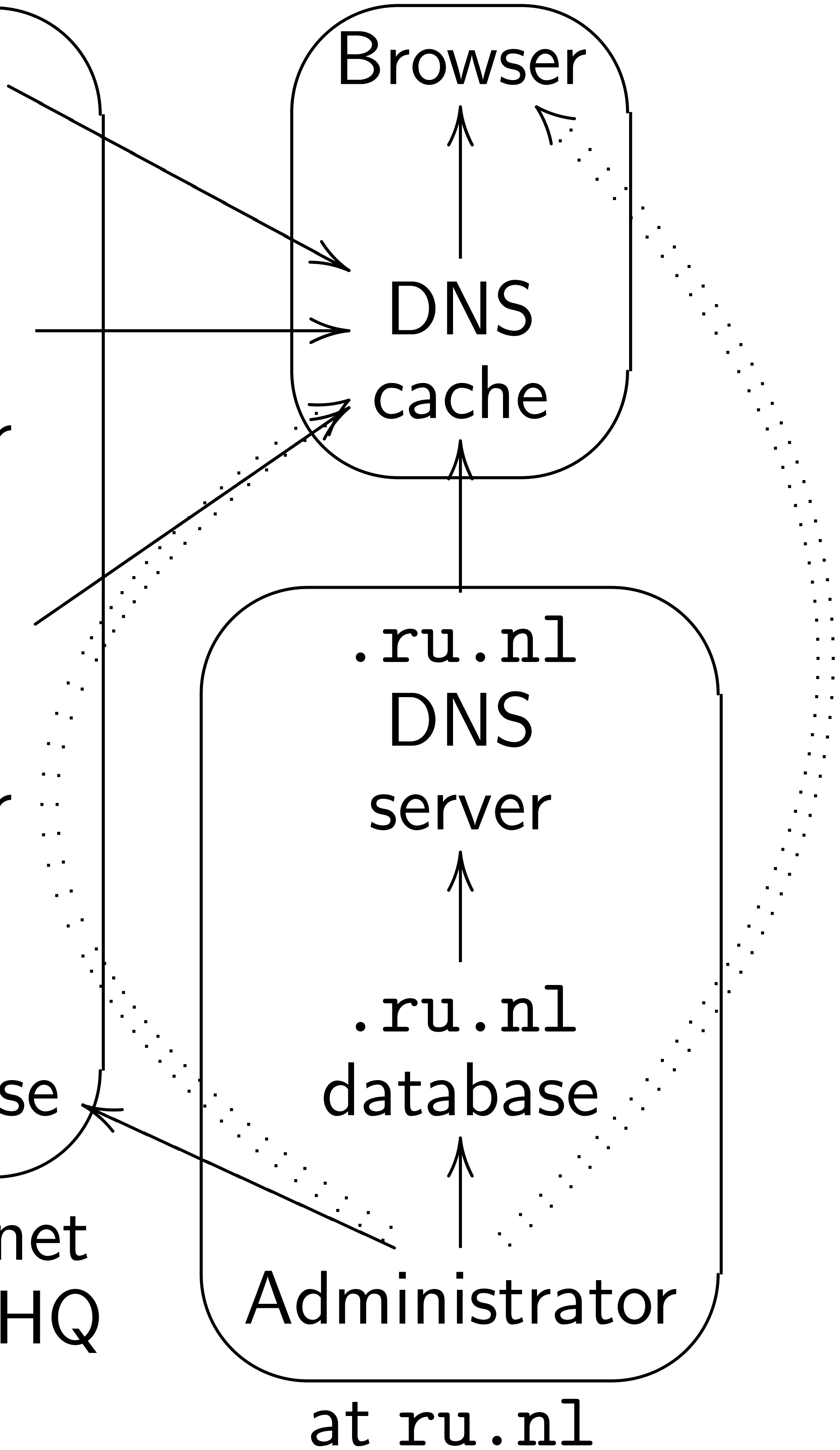
DNS server software listed in
Wikipedia: BIND, Microsoft
DNS, djbdns, Dnsmasq, Sim
DNS Plus, NSD, Knot DNS
PowerDNS, MaraDNS, pdns
Nominum ANS, Nominum V
Posadis, Unbound, Cisco Ne
Registrar, dnrd, gdnsd, YAD
yaku-ns, DNS Blast.

Much wider variety of DNS
database-management tools
hundreds of homegrown too
written by DNS registrars et



DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, Knot DNS, PowerDNS, MaraDNS, pdnsd, Nominum ANS, Nominum Vantio, Posadis, Unbound, Cisco Network Registrar, dnrd, gdnsd, YADIFA, yaku-ns, DNS Blast.

Much wider variety of DNS database-management tools, plus hundreds of homegrown tools written by DNS registrars etc.



DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, Knot DNS, PowerDNS, MaraDNS, pdnsd, Nominum ANS, Nominum Vantio, Posadis, Unbound, Cisco Network Registrar, dnrd, gdnsd, YADIFA, yaku-ns, DNS Blast.

Much wider variety of DNS database-management tools, plus hundreds of homegrown tools written by DNS registrars etc.

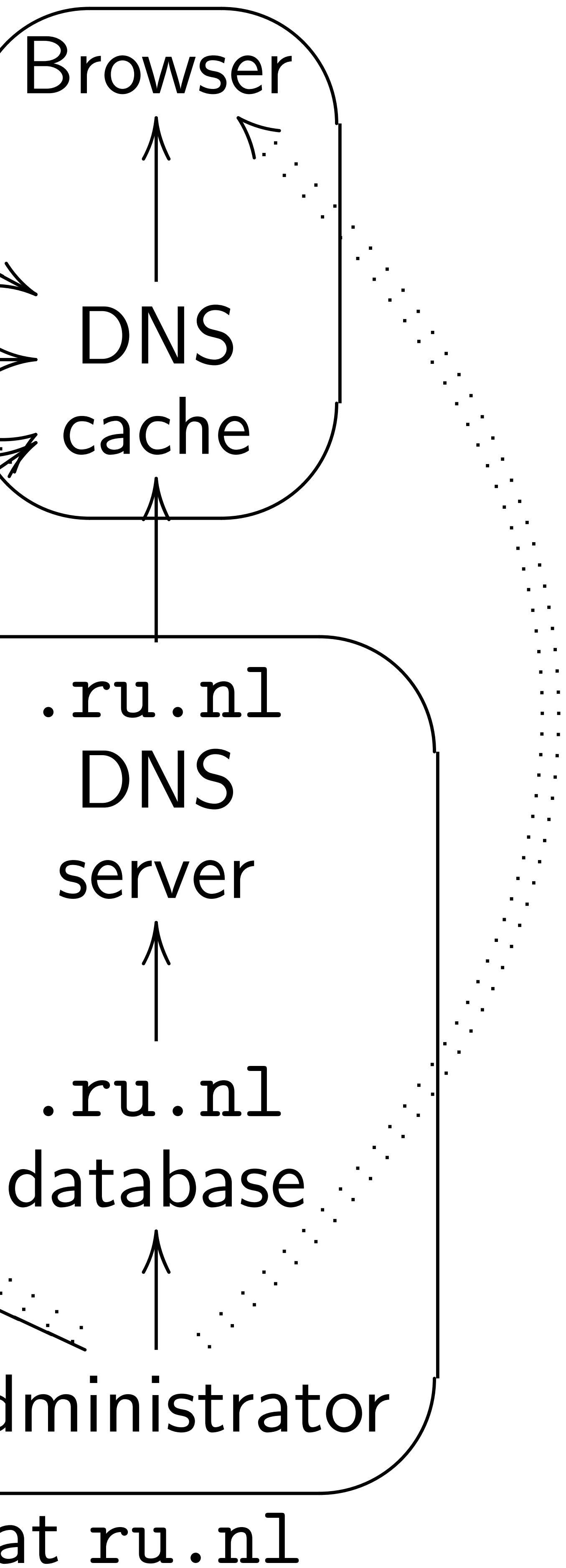
DNSSEC

DNSSEC
every DN

Wheneve
a DNS r
precomp
signature

Often co
for the t

Example
can proc
Tool rea
probably



DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, Knot DNS, PowerDNS, MaraDNS, pdnsd, Nominum ANS, Nominum Vantio, Posadis, Unbound, Cisco Network Registrar, dnrd, gdnsd, YADIFA, yaku-ns, DNS Blast.

Much wider variety of DNS database-management tools, plus hundreds of homegrown tools written by DNS registrars etc.

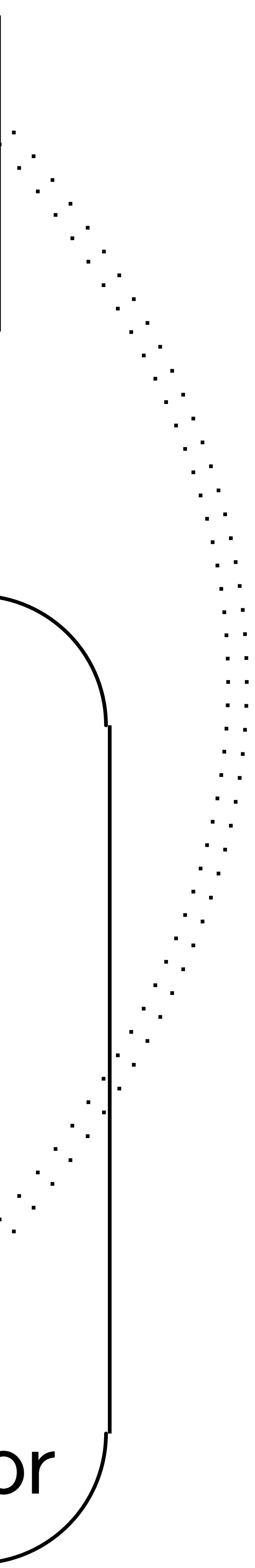
DNSSEC changes

DNSSEC demands
every DNS-manag

Whenever a tool a
a DNS record, also
precompute and st
signature for the n

Often considerable
for the tool progra

Example: Signing
can produce 20GB
Tool reading datab
probably has to be



DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, Knot DNS, PowerDNS, MaraDNS, pdnsd, Nominum ANS, Nominum Vantio, Posadis, Unbound, Cisco Network Registrar, dnrd, gdnsd, YADIFA, yaku-ns, DNS Blast.

Much wider variety of DNS database-management tools, plus hundreds of homegrown tools written by DNS registrars etc.

DNSSEC changes everything

DNSSEC demands new code every DNS-management too

Whenever a tool adds or cha a DNS record, also has to precompute and store a DN signature for the new record

Often considerable effort for the tool programmers.

Example: Signing 3GB data can produce 20GB database Tool reading database into probably has to be reenginee

DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, Knot DNS, PowerDNS, MaraDNS, pdnsd, Nominum ANS, Nominum Vantio, Posadis, Unbound, Cisco Network Registrar, dnrd, gdnsd, YADIFA, yaku-ns, DNS Blast.

Much wider variety of DNS database-management tools, plus hundreds of homegrown tools written by DNS registrars etc.

DNSSEC changes everything

DNSSEC demands new code in every DNS-management tool.

Whenever a tool adds or changes a DNS record, also has to precompute and store a DNSSEC signature for the new record.

Often considerable effort for the tool programmers.

Example: Signing 3GB database can produce 20GB database. Tool reading database into RAM probably has to be reengineered.

rver software listed in
ia: BIND, Microsoft
bdns, Dnsmasq, Simple
us, NSD, Knot DNS,
NS, MaraDNS, pdnsd,
m ANS, Nominum Vantio,
Unbound, Cisco Network
r, dnrd, gdnsd, YADIFA,
DNS Blast.

ider variety of DNS
e-management tools, plus
s of homegrown tools
by DNS registrars etc.

DNSSEC changes everything

DNSSEC demands new code in every DNS-management tool.

Whenever a tool adds or changes a DNS record, also has to precompute and store a DNSSEC signature for the new record.

Often considerable effort for the tool programmers.

Example: Signing 3GB database can produce 20GB database. Tool reading database into RAM probably has to be reengineered.

Nijmege
to send

The `.nl`
and data
and web
need to
to accep
and to s

DNS cac
to fetch
and veri

Tons of

are listed in
Microsoft
masq, Simple
Knot DNS,
DNS, pdnsd,
ominum Vantio,
, Cisco Network
dnsd, YADIFA,
st.

y of DNS
ment tools, plus
grown tools
egistrars etc.

DNSSEC changes everything

DNSSEC demands new code in every DNS-management tool.

Whenever a tool adds or changes a DNS record, also has to precompute and store a DNSSEC signature for the new record.

Often considerable effort for the tool programmers.

Example: Signing 3GB database can produce 20GB database. Tool reading database into RAM probably has to be reengineered.

Nijmegen administ
to send public key

The `.nl` server
and database softw
and web interface
need to be update
to accept these pu
and to sign everyt

DNS cache needs
to fetch keys, fetc
and verify signatur

Tons of pain for in

n
t
nple
,
sd,
Vantio,
etwork
IFA,

, plus
ls
tc.

DNSSEC changes everything

DNSSEC demands new code in every DNS-management tool.

Whenever a tool adds or changes a DNS record, also has to precompute and store a DNSSEC signature for the new record.

Often considerable effort for the tool programmers.

Example: Signing 3GB database can produce 20GB database. Tool reading database into RAM probably has to be reengineered.

Nijmegen administrator also to send public key to `.nl`.

The `.nl` server
and database software
and web interface
need to be updated
to accept these public keys
and to sign everything.

DNS cache needs new softw
to fetch keys, fetch signatur
and verify signatures.

Tons of pain for implemento

DNSSEC changes everything

DNSSEC demands new code in every DNS-management tool.

Whenever a tool adds or changes a DNS record, also has to precompute and store a DNSSEC signature for the new record.

Often considerable effort for the tool programmers.

Example: Signing 3GB database can produce 20GB database. Tool reading database into RAM probably has to be reengineered.

Nijmegen administrator also has to send public key to `.nl`.

The `.nl` server
and database software
and web interface
need to be updated
to accept these public keys
and to sign everything.

DNS cache needs new software to fetch keys, fetch signatures, and verify signatures.

Tons of pain for implementors.

C changes everything

C demands new code in
NS-management tool.

er a tool adds or changes
record, also has to
oute and store a DNSSEC
e for the new record.

onsiderable effort
ool programmers.

e: Signing 3GB database
duce 20GB database.
ding database into RAM
has to be reengineered.

Nijmegen administrator also has to send public key to `.nl`.

The `.nl` server *and* database software *and* web interface need to be updated to accept these public keys and to sign everything.

DNS cache needs new software to fetch keys, fetch signatures, and verify signatures.

Tons of pain for implementors.

Original
would ha
to sign i
millions

Concept
much to

So the D
added co
allowing
a small r
and to s
but has
covering

everything

s new code in
ement tool.

adds or changes
o has to
tore a DNSSEC
new record.

e effort
mmers.

3GB database
B database.
base into RAM
e reengineered.

Nijmegen administrator also has to send public key to `.nl`.

The `.nl` server *and* database software *and* web interface need to be updated to accept these public keys and to sign everything.

DNS cache needs new software to fetch keys, fetch signatures, and verify signatures.

Tons of pain for implementors.

Original DNSSEC
would have require
to sign its whole d
millions of records

Conceptually simp
much too slow, m

So the DNSSEC p
added complicated
allowing `.org` to s
a small number of
and to sign "migh
but has not signed
covering the other

g

e in

ol.

anges

SSEC

.

base

.

RAM

ered.

Nijmegen administrator also has to send public key to `.nl`.

The `.nl` server
and database software
and web interface
need to be updated
to accept these public keys
and to sign everything.

DNS cache needs new software
to fetch keys, fetch signatures,
and verify signatures.

Tons of pain for implementors.

Original DNSSEC protocols
would have required `.org`
to sign its whole database:
millions of records.

Conceptually simple but
much too slow, much too bi

So the DNSSEC protocol
added complicated options
allowing `.org` to sign
a small number of records,
and to sign “might have dat
but has not signed any of it’
covering the other records.

Nijmegen administrator also has to send public key to `.nl`.

The `.nl` server
and database software
and web interface
need to be updated
to accept these public keys
and to sign everything.

DNS cache needs new software
to fetch keys, fetch signatures,
and verify signatures.

Tons of pain for implementors.

Original DNSSEC protocols
would have required `.org`
to sign its whole database:
millions of records.

Conceptually simple but
much too slow, much too big.

So the DNSSEC protocol
added complicated options
allowing `.org` to sign
a small number of records,
and to sign "might have data
but has not signed any of it"
covering the other records.

n administrator also has
public key to `.nl`.

. server
abase software
o interface
be updated
ot these public keys
ign everything.

che needs new software
keys, fetch signatures,
fy signatures.

pain for implementors.

Original DNSSEC protocols would have required `.org` to sign its whole database: millions of records.

Conceptually simple but much too slow, much too big.

So the DNSSEC protocol added complicated options allowing `.org` to sign a small number of records, and to sign "might have data but has not signed any of it" covering the other records.

What ab

e.g. Mo
return ra
to sprea

Often th
adjust lis
in light o
client lo

trator also has
to .nl.

ware

ed
ublic keys
hing.

new software
h signatures,
res.

mplementors.

Original DNSSEC protocols would have required `.org` to sign its whole database: millions of records.

Conceptually simple but much too slow, much too big.

So the DNSSEC protocol added complicated options allowing `.org` to sign a small number of records, and to sign "might have data but has not signed any of it" covering the other records.

What about *dynam*

e.g. Most big sites
return random IP
to spread load acr

Often they automa
adjust list of addre
in light of dead se
client location, etc

has

are

es,

rs.

Original DNSSEC protocols would have required `.org` to sign its whole database: millions of records.

Conceptually simple but much too slow, much too big.

So the DNSSEC protocol added complicated options allowing `.org` to sign a small number of records, and to sign "might have data but has not signed any of it" covering the other records.

What about *dynamic* DNS

e.g. Most big sites return random IP addresses to spread load across server

Often they automatically adjust list of addresses in light of dead servers, client location, etc.

Original DNSSEC protocols would have required `.org` to sign its whole database: millions of records.

Conceptually simple but much too slow, much too big.

So the DNSSEC protocol added complicated options allowing `.org` to sign a small number of records, and to sign "might have data but has not signed any of it" covering the other records.

What about *dynamic* DNS data?

e.g. Most big sites return random IP addresses to spread load across servers.

Often they automatically adjust list of addresses in light of dead servers, client location, etc.

Original DNSSEC protocols would have required `.org` to sign its whole database: millions of records.

Conceptually simple but much too slow, much too big.

So the DNSSEC protocol added complicated options allowing `.org` to sign a small number of records, and to sign "might have data but has not signed any of it" covering the other records.

What about *dynamic* DNS data?

e.g. Most big sites return random IP addresses to spread load across servers.

Often they automatically adjust list of addresses in light of dead servers, client location, etc.

DNSSEC purists say "Answers should always be static".

DNSSEC protocols
ave required .org
ts whole database:
of records.

ually simple but
o slow, much too big.

DNSSEC protocol
omplicated options
.org to sign
number of records,
ign "might have data
not signed any of it"
the other records.

What about *dynamic* DNS data?

e.g. Most big sites
return random IP addresses
to spread load across servers.

Often they automatically
adjust list of addresses
in light of dead servers,
client location, etc.

DNSSEC purists say "Answers
should always be static".

Even in
each res
dynamic
from sev
MX ansv

DNSSEC
a signatu
not for e

⇒ One
includes
Massive

That's w
so much

protocols
ed .org
latabase:
.

le but
uch too big.

rotocol
l options
sign
records,
t have data
l any of it"
records.

What about *dynamic* DNS data?

e.g. Most big sites
return random IP addresses
to spread load across servers.

Often they automatically
adjust list of addresses
in light of dead servers,
client location, etc.

DNSSEC purists say "Answers
should always be static".

Even in "static" D
each response pac
dynamically assem
from several answe
MX answer, NS ar

DNSSEC precomp
a signature for eac
not for each packe

⇒ One DNSSEC
includes several sig
Massive bloat on t

That's why DNSS
so much amplifica

What about *dynamic* DNS data?

e.g. Most big sites
return random IP addresses
to spread load across servers.

Often they automatically
adjust list of addresses
in light of dead servers,
client location, etc.

DNSSEC purists say "**Answers should always be static**".

Even in "static" DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

⇒ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That's why DNSSEC allows
so much amplification.

What about *dynamic* DNS data?

e.g. Most big sites
return random IP addresses
to spread load across servers.

Often they automatically
adjust list of addresses
in light of dead servers,
client location, etc.

DNSSEC purists say "Answers
should always be static".

Even in "static" DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

$\Rightarrow$ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That's why DNSSEC allows
so much amplification.

bout *dynamic* DNS data?

st big sites
andom IP addresses
d load across servers.

hey automatically
st of addresses
of dead servers,
cation, etc.

C purists say "Answers
lways be static".

Even in "static" DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

⇒ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That's why DNSSEC allows
so much amplification.

What ab
Are the

Can an a
obsolete

e.g. You
Attacker
replays c

mic DNS data?

s
addresses
oss servers.

atically
esses
rvers,
c.

ay "Answers
static".

Even in "static" DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

⇒ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That's why DNSSEC allows
so much amplification.

What about *old* D
Are the signatures

Can an attacker re
obsolete signed da

e.g. You move IP
Attacker grabs old
replays old signatu

data?

s.

ers

Even in “static” DNS, each response packet is dynamically assembled from several answers: MX answer, NS answer, etc.

DNSSEC precomputes a signature for each answer, not for each packet.

⇒ One DNSSEC packet includes several signatures. Massive bloat on the wire.

That’s why DNSSEC allows so much amplification.

What about *old* DNS data? Are the signatures still valid

Can an attacker replay obsolete signed data?

e.g. You move IP addresses. Attacker grabs old address, replays old signature.

Even in “static” DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

⇒ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That’s why DNSSEC allows
so much amplification.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

Even in “static” DNS,
each response packet is
dynamically assembled
from several answers:
MX answer, NS answer, etc.

DNSSEC precomputes
a signature for each answer,
not for each packet.

$\Rightarrow$ One DNSSEC packet
includes several signatures.
Massive bloat on the wire.

That’s why DNSSEC allows
so much amplification.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

"static" DNS,
ponse packet is
ally assembled
veral answers:
wer, NS answer, etc.

C precomputes
ure for each answer,
each packet.

DNSSEC packet
several signatures.
bloat on the wire.

why DNSSEC allows
amplification.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

A few D

2010.09.

NS,
ket is
bled
ers:
nswer, etc.

utes
h answer,
t.

packet
gnatures.
he wire.

EC allows
ion.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

A few DNSSEC su

2010.09.02: `.us` k

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

A few DNSSEC suicide exan

2010.09.02: `.us` killed itself

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay obsolete signed data?

e.g. You move IP addresses. Attacker grabs old address, replays old signature.

If clocks are synchronized then signatures can include expiration times. But frequent re-signing is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt:
"`dnssec-accept-expired yes`"

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay obsolete signed data?

e.g. You move IP addresses. Attacker grabs old address, replays old signature.

If clocks are synchronized then signatures can include expiration times. But frequent re-signing is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay obsolete signed data?

e.g. You move IP addresses. Attacker grabs old address, replays old signature.

If clocks are synchronized then signatures can include expiration times. But frequent re-signing is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

e.g. You move IP addresses.
Attacker grabs old address,
replays old signature.

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt:
"`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org`
killed itself.

2015.12.11: `af.mil` killed itself.

What about *old* DNS data?
Are the signatures still valid?

Can an attacker replay obsolete signed data?

e.g. You move IP addresses. Attacker grabs old address, replays old signature.

If clocks are synchronized then signatures can include expiration times. But frequent re-signing is an administrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

bout *old* DNS data?
signatures still valid?

attacker replay
signed data?

u move IP addresses.
r grabs old address,
old signature.

are synchronized
natures can
expiration times.
uent re-signing
ministrative disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt:
"`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org`
killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com`
`/pub/dnssec-outages.html`.

What ab

DNS data?
still valid?

eplay
ta?

addresses.
l address,
ure.

ronized
n
times.
gning
ve disaster.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonex*

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonexistent* dat

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonexistent* data?

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

Crazy! Obvious approach: "We sign each record that exists, and don't sign anything else."

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt: "`dnssec-accept-expired yes`"

2012.10.28: `.nl` killed itself.

2015.01.25: `opendnssec.org` killed itself.

2015.12.11: `af.mil` killed itself.

Many more: see `ianix.com /pub/dnssec-outages.html`.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

Crazy! Obvious approach: "We sign each record that exists, and don't sign anything else."

User asks for nonexistent name. Receives *unsigned* answer saying the name doesn't exist. Has no choice but to trust it.

NSSEC suicide examples:

.02: `.us` killed itself.

.28, ISC's Evan Hunt:
c-accept-expired yes"

.28: `.nl` killed itself.

.25: `opendnssec.org`
self.

.11: `af.mil` killed itself.

ore: see `ianix.com`
`nssec-outages.html`.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

Crazy! Obvious approach:
"We sign each record that exists, and don't sign anything else."

User asks for nonexistent name.
Receives *unsigned* answer saying the name doesn't exist.
Has no choice but to trust it.

User ask
Receives
a packet
saying th
Has no c

Clearly a
Sometim
This is n

uicide examples:

killed itself.

Evan Hunt:
-expired yes"

killed itself.

dnssec.org

il killed itself.

anix.com
tages.html.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

Crazy! Obvious approach: "We sign each record that exists, and don't sign anything else."

User asks for nonexistent name. Receives *unsigned* answer saying the name doesn't exist. Has no choice but to trust it.

User asks for `www.`
Receives unsigned
a packet forged by
saying the name d
Has no choice but

Clearly a violation
Sometimes a viola
This is not a good

mples:

.

t:

yes”

.

rg

tself.

n

ml.

What about *nonexistent* data?

Does Nijmegen administrator
precompute signatures on
“`aaaaa.ru.nl does not exist`”,
“`aaaab.ru.nl does not exist`”,
etc.?

Crazy! Obvious approach:
“We sign each record that exists,
and don’t sign anything else.”

User asks for nonexistent name.
Receives *unsigned* answer
saying the name doesn’t exist.
Has no choice but to trust it.

User asks for `www.google.c`
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn’t exis
Has no choice but to trust i

Clearly a violation of availab
Sometimes a violation of int
This is not a good approach

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

Crazy! Obvious approach: "We sign each record that exists, and don't sign anything else."

User asks for nonexistent name. Receives *unsigned* answer saying the name doesn't exist. Has no choice but to trust it.

User asks for `www.google.com`. Receives unsigned answer, a packet forged by attacker, saying the name doesn't exist. Has no choice but to trust it.

Clearly a violation of availability. Sometimes a violation of integrity. This is not a good approach.

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on “`aaaaa.ru.nl does not exist`”, “`aaaab.ru.nl does not exist`”, etc.?

Crazy! Obvious approach: “We sign each record that exists, and don’t sign anything else.”

User asks for nonexistent name. Receives *unsigned* answer saying the name doesn’t exist. Has no choice but to trust it.

User asks for `www.google.com`. Receives unsigned answer, a packet forged by attacker, saying the name doesn’t exist. Has no choice but to trust it.

Clearly a violation of availability. Sometimes a violation of integrity. This is not a good approach.

Alternative: DNSSEC’s “NSEC”. e.g. `nonex.clegg.com` query returns “`There are no names between nick.clegg.com and start.clegg.com`” + signature.

bout *nonexistent* data?

jmegen administrator
bute signatures on
.ru.nl does not exist",
.ru.nl does not exist",

Obvious approach:
n each record that exists,
't sign anything else."

ks for nonexistent name.
s *unsigned* answer
he name doesn't exist.
choice but to trust it.

User asks for `www.google.com`.
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: DNSSEC's "NSEC".
e.g. `nonex.clegg.com` query
returns "`There are no names between nick.clegg.com and start.clegg.com`" + signature.

Try `foo`
After sev
complete
`_jabber`
`server.`
`andrew`,
`googlef`
`home`, `in`
`localho`

xistent data?

ministrator
tures on
`es not exist`”,
`es not exist`”,

pproach:
ord that exists,
thing else.”

xistent name.
answer
oesn't exist.
to trust it.

User asks for `www.google.com`.
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: DNSSEC's “NSEC”.
e.g. `nonex.clegg.com` query
returns “`There are no names`
`between nick.clegg.com and`
`start.clegg.com`” + signature.

Try `foo.clegg.c`
After several queri
complete `clegg.c`
`_jabber._tcp`, `_`
`server._tcp`, `ala`
`andrew`, `brian`, `c`
`googleffffffe`
`home`, `imogene`, `j`
`localhost`, `mail`,

ta?

or

xist”,

xist”,

xists,

.”

me.

st.

t.

User asks for `www.google.com`.
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn’t exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: DNSSEC’s “NSEC”.
e.g. `nonex.clegg.com` query
returns “`There are no names`
`between nick.clegg.com and`
`start.clegg.com`” + signature.

Try `foo.clegg.com` etc.
After several queries have
complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-`
`server._tcp`, `alan`, `alvis`
`andrew`, `brian`, `calendar`,
`googleffffffffe91126e7`,
`home`, `imogene`, `jennifer`,
`localhost`, `mail`, `wiki`, `ww`

User asks for `www.google.com`.
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: DNSSEC's "NSEC".
e.g. `nonex.clegg.com` query
returns "`There are no names between` `nick.clegg.com` `and` `start.clegg.com`" + signature.

Try `foo.clegg.com` etc.
After several queries have
complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`,
`andrew`, `brian`, `calendar`, `dlv`,
`googleffffffffe91126e7`,
`home`, `imogene`, `jennifer`,
`localhost`, `mail`, `wiki`, `www`.

User asks for `www.google.com`.
Receives unsigned answer,
a packet forged by attacker,
saying the name doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: DNSSEC's "NSEC".
e.g. `nonex.clegg.com` query
returns "`There are no names between nick.clegg.com and start.clegg.com`" + signature.

Try `foo.clegg.com` etc.
After several queries have
complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`,
`andrew`, `brian`, `calendar`, `dlv`,
`googleffffffffe91126e7`,
`home`, `imogene`, `jennifer`,
`localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator
disabled DNS "zone transfers"
— but then leaked the same data
by installing DNSSEC.
(This was a real example.)

ks for `www.google.com`.
s unsigned answer,
t forged by attacker,
he name doesn't exist.
choice but to trust it.

a violation of availability.
nes a violation of integrity.
not a good approach.

ive: DNSSEC's "NSEC".
nex.clegg.com query
"There are no names
n nick.clegg.com and
clegg.com" + signature.

Try `foo.clegg.com` etc.
After several queries have complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`, `andrew`, `brian`, `calendar`, `dlv`, `googleffffffffe91126e7`, `home`, `imogene`, `jennifer`, `localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator disabled DNS "zone transfers"
— but then leaked the same data by installing DNSSEC.
(This was a real example.)

Summar
all n nar
(with sig
that the
using n

.google.com.
answer,
attacker,
oesn't exist.
to trust it.

of availability.
tion of integrity.
approach.

SEC's "NSEC".
g.com query
e no names
egg.com and
n" + signature.

Try foo.clegg.com etc.
After several queries have
complete clegg.com list:
_jabber._tcp, _xmpp-server._tcp, alan, alvis,
andrew, brian, calendar, dlv,
googleffffffffe91126e7,
home, imogene, jennifer,
localhost, mail, wiki, www.

The clegg.com administrator
disabled DNS "zone transfers"
— but then leaked the same data
by installing DNSSEC.
(This was a real example.)

Summary: Attack
all n names in an
(with signatures g
that there are no
using n DNS quer

com.

st.

t.

bility.

egrity.

.

SEC".

ry

es

and

ature.

Try `foo.clegg.com` etc.
After several queries have complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`, `andrew`, `brian`, `calendar`, `dlv`, `googleffffffffe91126e7`, `home`, `imogene`, `jennifer`, `localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator disabled DNS "zone transfers" — but then leaked the same data by installing DNSSEC.
(This was a real example.)

Summary: Attacker learns all *n* names in an NSEC zon
(with signatures guaranteein
that there are no more)
using *n* DNS queries.

Try `foo.clegg.com` etc.
After several queries have complete `clegg.com` list:
`_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`, `andrew`, `brian`, `calendar`, `dlv`, `googleffffffffe91126e7`, `home`, `imogene`, `jennifer`, `localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator disabled DNS "zone transfers" — but then leaked the same data by installing DNSSEC.
(This was a real example.)

Summary: Attacker learns all n names in an NSEC zone (with signatures guaranteeing that there are no more) using n DNS queries.

Try `foo.clegg.com` etc.
After several queries have complete `clegg.com` list: `_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`, `andrew`, `brian`, `calendar`, `dlv`, `googleffffffffe91126e7`, `home`, `imogene`, `jennifer`, `localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator disabled DNS "zone transfers" — but then leaked the same data by installing DNSSEC.
(This was a real example.)

Summary: Attacker learns all *n* names in an NSEC zone (with signatures guaranteeing that there are no more) using *n* DNS queries.

This is not a good approach.

Try `foo.clegg.com` etc.

After several queries have complete `clegg.com` list: `_jabber._tcp`, `_xmpp-server._tcp`, `alan`, `alvis`, `andrew`, `brian`, `calendar`, `dlv`, `googleffffffffe91126e7`, `home`, `imogene`, `jennifer`, `localhost`, `mail`, `wiki`, `www`.

The `clegg.com` administrator disabled DNS "zone transfers" — but then leaked the same data by installing DNSSEC.

(This was a real example.)

Summary: Attacker learns all n names in an NSEC zone (with signatures guaranteeing that there are no more) using n DNS queries.

This is not a good approach.

DNSSEC purists disagree: "It is part of the design philosophy of the DNS that the data in it is public." But this notion is so extreme that it became a public-relations problem.

.clegg.com etc.
veral queries have
e clegg.com list:
r._tcp, _xmpp-
._tcp, alan, alvis,
brian, calendar, dlv,
ffffffe91126e7,
mogene, jennifer,
ost, mail, wiki, www.

egg.com administrator
DNS “zone transfers”
hen leaked the same data
ling DNSSEC.
as a real example.)

Summary: Attacker learns all n names in an NSEC zone (with signatures guaranteeing that there are no more) using n DNS queries.

This is not a good approach.

DNSSEC purists disagree:
“It is part of the design philosophy of the DNS that the data in it is public.”
But this notion is so extreme that it became a public-relations problem.

New DN

1. “NSE
Use a “c
such as
Reveal h
instead o
“There
hashes

om etc.
es have
com list:
xmpp-
an, alvis,
alendar, dlv,
e91126e7,
ennifer,
, wiki, www.

dministrator
ne transfers"
d the same data
SEC.
xample.)

Summary: Attacker learns all n names in an NSEC zone (with signatures guaranteeing that there are no more) using n DNS queries.

This is not a good approach.

DNSSEC purists disagree:
"It is part of the design philosophy of the DNS that the data in it is public."
But this notion is so extreme that it became a public-relations problem.

New DNSSEC app

1. "NSEC3" techn
Use a "one-way ha
such as (iterated s
Reveal *hashes* of n
instead of revealin
"There are no na
hashes between

,
`dlv`,
`w`.
or
rs”
e data

Summary: Attacker learns all *n* names in an NSEC zone (with signatures guaranteeing that there are no more) using *n* DNS queries.

This is not a good approach.

DNSSEC purists disagree: “It is part of the design philosophy of the DNS that the data in it is public.” But this notion is so extreme that it became a public-relations problem.

New DNSSEC approach:

1. “NSEC3” technology:
Use a “one-way hash functic
such as (iterated salted) SH
Reveal *hashes* of names
instead of revealing names.
“`There are no names with`
`hashes between` ... `and` ..

Summary: Attacker learns all n names in an NSEC zone (with signatures guaranteeing that there are no more) using n DNS queries.

This is not a good approach.

DNSSEC purists disagree: "It is part of the design philosophy of the DNS that the data in it is public." But this notion is so extreme that it became a public-relations problem.

New DNSSEC approach:

1. "NSEC3" technology: Use a "one-way hash function" such as (iterated salted) SHA-1. Reveal *hashes* of names instead of revealing names. "`There are no names with hashes between` ... `and` ..."

Summary: Attacker learns all *n* names in an NSEC zone (with signatures guaranteeing that there are no more) using *n* DNS queries.

This is not a good approach.

DNSSEC purists disagree: "It is part of the design philosophy of the DNS that the data in it is public." But this notion is so extreme that it became a public-relations problem.

New DNSSEC approach:

1. "NSEC3" technology: Use a "one-way hash function" such as (iterated salted) SHA-1. Reveal *hashes* of names instead of revealing names. "`There are no names with hashes between ... and ...`"

2. Marketing: Pretend that NSEC3 is less damaging than NSEC.

ISC: "NSEC3 does not allow enumeration of the zone."

y: Attacker learns
mes in an NSEC zone
gnatures guaranteeing
re are no more)
DNS queries.

not a good approach.

C purists disagree:
rt of the design
hy of the DNS
data in it is public."
notion is so extreme
ecame a
elations problem.

New DNSSEC approach:

1. "NSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
`"There are no names with hashes between ... and ..."`

2. Marketing:
Pretend that NSEC3 is
less damaging than NSEC.

ISC: "NSEC3 does not allow
enumeration of the zone."

Reality:
by abusi
compute
for many
quickly
(and kno

er learns
NSEC zone
uaranteeing
more)
ies.

l approach.

lisagree:
lesign
DNS
is public."
so extreme

oblem.

New DNSSEC approach:

1. "NSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NSEC3 is
less damaging than NSEC.

ISC: "NSEC3 does not allow enumeration of the zone."

Reality: Attacker
by abusing DNSSE
computes the sam
for many different
quickly discovers a
(and knows # mis

New DNSSEC approach:

1. "NSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with`
`hashes between ... and ...`"

2. Marketing:
Pretend that NSEC3 is
less damaging than NSEC.

ISC: "NSEC3 does not allow
enumeration of the zone."

Reality: Attacker grabs the
by abusing DNSSEC's NSEC
computes the same hash fur
for many different name gue
quickly discovers almost all
(and knows # missing name

New DNSSEC approach:

1. "NSEC3" technology:
Use a "one-way hash function" such as (iterated salted) SHA-1.
Reveal *hashes* of names instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NSEC3 is less damaging than NSEC.

ISC: "NSEC3 does not allow enumeration of the zone."

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

New DNSSEC approach:

1. “NSEC3” technology:
Use a “one-way hash function” such as (iterated salted) SHA-1.
Reveal *hashes* of names instead of revealing names.
“`There are no names with hashes between ... and ...`”

2. Marketing:
Pretend that NSEC3 is less damaging than NSEC.

ISC: “NSEC3 does not allow enumeration of the zone.”

Reality: Attacker grabs the hashes by abusing DNSSEC’s NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: “You could have sent all the same guesses as queries to the server.”

New DNSSEC approach:

1. “NSEC3” technology:
Use a “one-way hash function” such as (iterated salted) SHA-1.
Reveal *hashes* of names instead of revealing names.
“`There are no names with hashes between ... and ...`”

2. Marketing:
Pretend that NSEC3 is less damaging than NSEC.

ISC: “NSEC3 does not allow enumeration of the zone.”

Reality: Attacker grabs the hashes by abusing DNSSEC’s NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: “You could have sent all the same guesses as queries to the server.”

4Mbps flood of queries is under 500 million noisy guesses/day.
NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

ISSEC approach:

EC3" technology:
one-way hash function"
(iterated salted) SHA-1.
ashes of names
of revealing names.
`are no names with`
`between ... and ...`"

eting:
that NSEC3 is
aging than NSEC.

SEC3 does not allow
tion of the zone."

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

This is c

Imagine
that wor

roach:

nology:
ash function"
salted) SHA-1.
names
g names.
ames with
. . . and . . ."

C3 is
n NSEC.

s not allow
e zone."

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

This is crazy!

Imagine an "HTT
that works like DN

on”
A-1.

.

.”

/

Reality: Attacker grabs the hashes by abusing DNSSEC’s NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: “You could have sent all the same guesses as queries to the server.”

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

<u>This is crazy!</u>

Imagine an “HTTPSEC” that works like DNSSEC.

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

This is crazy!

Imagine an "HTTPSEC" that works like DNSSEC.

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

This is crazy!

Imagine an "HTTPSEC" that works like DNSSEC.

Store a signature next to every web page.
Recompute and store signature for every minor wiki edit, and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Reality: Attacker grabs the hashes by abusing DNSSEC's NSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

DNSSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 500 million noisy guesses/day. NSEC3 allows typical attackers 1000000 million to 1000000000 million silent guesses/day.

<u>This is crazy!</u>

Imagine an "HTTPSEC" that works like DNSSEC.

Store a signature next to every web page.
Recompute and store signature for every minor wiki edit, and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Attacker grabs the hashes
ng DNSSEC's NSEC3;
es the same hash function
y different name guesses;
discovers almost all names
ows # missing names).

C purists: "You could
nt all the same guesses
es to the server."

lood of queries is under
ion noisy guesses/day.
allows typical attackers
million to 1000000000
silent guesses/day.

This is crazy!

Imagine an "HTTPSEC"
that works like DNSSEC.

Store a signature next to
every web page.
Recompute and store signature
for every minor wiki edit,
and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DN

There *ar*
signed w
caches c
Never m
Do thes
accomp

grabs the hashes
EC's NSEC3;
e hash function
name guesses;
lmost all names
sing names).

"You could
ame guesses
erver."

ueries is under
guesses/day.
ical attackers
1000000000
ses/day.

This is crazy!

Imagine an "HTTPSEC"
that works like DNSSEC.

Store a signature next to
every web page.
Recompute and store signature
for every minor wiki edit,
and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DNS security

There *are* some IP
signed with DNSS
caches checking si
Never mind all the
Do these signatu
accomplish anyth

hashes
C3;
nction
esses;
names
es).

d
ses

nder
ay.
kers
000

This is crazy!

Imagine an "HTTPSEC"
that works like DNSSEC.

Store a signature next to
every web page.
Recompute and store signature
for every minor wiki edit,
and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DNS security matter?

There *are* some IP addresse
signed with DNSSEC, and s
caches checking signatures.
Never mind all the problems
Do these signatures
accomplish anything?

This is crazy!

Imagine an "HTTPSEC"
that works like DNSSEC.

Store a signature next to
every web page.
Recompute and store signature
for every minor wiki edit,
and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DNS security matter?

There *are* some IP addresses
signed with DNSSEC, and some
caches checking signatures.
Never mind all the problems.
Do these signatures
accomplish anything?

This is crazy!

Imagine an "HTTPSEC" that works like DNSSEC.

Store a signature next to every web page.
Recompute and store signature for every minor wiki edit, and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures.
Never mind all the problems.
Do these signatures accomplish anything?

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . .

This is crazy!

Imagine an “HTTPSEC”
that works like DNSSEC.

Store a signature next to
every web page.
Recompute and store signature
for every minor wiki edit,
and again every 30 days.
Any failure: HTTPSEC suicide.
Dynamic content? Give up.

Replay attacks work for 30 days.
Filename guessing is much faster.
Nothing is encrypted.
Denial of service is trivial.

Does DNS security matter?

There *are* some IP addresses
signed with DNSSEC, and some
caches checking signatures.
Never mind all the problems.
**Do these signatures
accomplish anything?**

Occasionally these caches
are on client machines,
so attacker can’t simply
forge packets from cache . . .
so attacker intercepts and forges
all the subsequent packets:
web pages, email, etc.

crazy!

an "HTTPSEC"
rks like DNSSEC.

signature next to
eb page.
ute and store signature
y minor wiki edit,
in every 30 days.
ure: HTTPSEC suicide.
c content? Give up.

attacks work for 30 days.
e guessing is much faster.
is encrypted.
of service is trivial.

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems. **Do these signatures accomplish anything?**

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Adminis
to prote
. . . but
is stopp

PSEC"
NSSEC.

next to

ore signature
ki edit,
0 days.
PSEC suicide.
Give up.

rk for 30 days.
is much faster.
ted.
s trivial.

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems. **Do these signatures accomplish anything?**

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Administrator can
to protect web pag
. . . but then what
is stopped by DNS

ure

ide.

days.
aster.

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems.
Do these signatures accomplish anything?

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Administrator can use HTTI
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems. **Do these signatures accomplish anything?**

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Administrator can use HTTPS to protect web pages . . . but then what attack is stopped by DNSSEC?

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems. **Do these signatures accomplish anything?**

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Administrator can use HTTPS to protect web pages . . . but then what attack is stopped by DNSSEC?

DNSSEC purists criticize HTTPS: "You can't trust your servers."

DNSSEC signers are offline (preferably in guarded rooms). DNSSEC precomputes signatures. DNSSEC doesn't trust servers.

Does DNS security matter?

There *are* some IP addresses signed with DNSSEC, and some caches checking signatures. Never mind all the problems. **Do these signatures accomplish anything?**

Occasionally these caches are on client machines, so attacker can't simply forge packets from cache . . . so attacker intercepts and forges all the subsequent packets: web pages, email, etc.

Administrator can use HTTPS to protect web pages . . . but then what attack is stopped by DNSSEC?

DNSSEC purists criticize HTTPS: "You can't trust your servers."

DNSSEC signers are offline (preferably in guarded rooms). DNSSEC precomputes signatures. DNSSEC doesn't trust servers.

But DNSSEC is not signing any of the user's data!

NS security matter?

re some IP addresses
vith DNSSEC, and some
checking signatures.
nind all the problems.
se signatures
plish anything?

nally these caches
lient machines,
ker can't simply
ckets from cache . . .
ker intercepts and forges
ubsequent packets:
es, email, etc.

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
"You can't trust your servers."

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn't trust servers.

But DNSSEC is not signing
any of the user's data!

PGP sig
PGP-sig
are prote
misbeha
and agai

y matter?

P addresses
EC, and some
gnatures.
e problems.

ures
ning?

e caches
nines,
simply
n cache . . .
epts and forges
 packets:
 etc.

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
"You can't trust your servers."

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn't trust servers.

But DNSSEC is not signing
any of the user's data!

PGP signs the use
PGP-signed web p
are protected agai
misbehaving serve
and against netwo

s
ome
.
.
orges

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
“You can’t trust your servers.”

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn’t trust servers.

But DNSSEC is not signing
any of the user’s data!

PGP signs the user’s data.
PGP-signed web pages and
are protected against
misbehaving servers,
and against network attacke

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
"You can't trust your servers."

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn't trust servers.

But DNSSEC is not signing
any of the user's data!

PGP signs the user's data.
PGP-signed web pages and email
are protected against
misbehaving servers,
and against network attackers.

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
“You can’t trust your servers.”

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn’t trust servers.

But DNSSEC is not signing
any of the user’s data!

PGP signs the user’s data.
PGP-signed web pages and email
are protected against
misbehaving servers,
and against network attackers.

With PGP, what attack
is stopped by DNSSEC?

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
"You can't trust your servers."

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn't trust servers.

But DNSSEC is not signing
any of the user's data!

PGP signs the user's data.
PGP-signed web pages and email
are protected against
misbehaving servers,
and against network attackers.

With PGP, what attack
is stopped by DNSSEC?

With HTTPS but not PGP, what
attack is stopped by DNSSEC?

Administrator can use HTTPS
to protect web pages
. . . but then what attack
is stopped by DNSSEC?

DNSSEC purists criticize HTTPS:
"You can't trust your servers."

DNSSEC signers are offline
(preferably in guarded rooms).
DNSSEC precomputes signatures.
DNSSEC doesn't trust servers.

But DNSSEC is not signing
any of the user's data!

PGP signs the user's data.
PGP-signed web pages and email
are protected against
misbehaving servers,
and against network attackers.

With PGP, what attack
is stopped by DNSSEC?

With HTTPS but not PGP, what
attack is stopped by DNSSEC?

With neither HTTPS nor PGP,
what attack is stopped by
DNSSEC?

trator can use HTTPS
ct web pages
then what attack
ed by DNSSEC?

C purists criticize HTTPS:
n't trust your servers."

C signers are offline
bly in guarded rooms).
C precomputes signatures.
C doesn't trust servers.

SSEC is not signing
he user's data!

PGP signs the user's data.
PGP-signed web pages and email are protected against misbehaving servers, and against network attackers.

With PGP, what attack is stopped by DNSSEC?

With HTTPS but not PGP, what attack is stopped by DNSSEC?

With neither HTTPS nor PGP, what attack is stopped by DNSSEC?

Getting

State-of-
is fast e
authenti
every pa

Deployed
DNS pa

Deployed
DNS pa

Work in
protects

use HTTPS
ges
attack
SSEC?

riticize HTTPS:
our servers."

are offline
rded rooms).
putes signatures.
trust servers.

ot signing
data!

PGP signs the user's data.
PGP-signed web pages and email
are protected against
misbehaving servers,
and against network attackers.

With PGP, what attack
is stopped by DNSSEC?

With HTTPS but not PGP, what
attack is stopped by DNSSEC?

With neither HTTPS nor PGP,
what attack is stopped by
DNSSEC?

Getting out of the

State-of-the-art E
is fast enough to
authenticate and e
every packet.

Deployed: DNSCu
DNS packets, serv

Deployed: DNSCr
DNS packets, cach

Work in progress:
protects HTTP pa

PS

TTPS:

s."

s).

tures.

ers.

PGP signs the user's data.

PGP-signed web pages and email are protected against misbehaving servers, and against network attackers.

With PGP, what attack is stopped by DNSSEC?

With HTTPS but not PGP, what attack is stopped by DNSSEC?

With neither HTTPS nor PGP, what attack is stopped by DNSSEC?

Getting out of the mess

State-of-the-art ECC is fast enough to authenticate and encrypt every packet.

Deployed: DNSCurve protec DNS packets, server→cache

Deployed: DNSCrypt protec DNS packets, cache→client.

Work in progress: HTTPCu protects HTTP packets.

PGP signs the user's data.
PGP-signed web pages and email are protected against misbehaving servers, and against network attackers.

With PGP, what attack is stopped by DNSSEC?

With HTTPS but not PGP, what attack is stopped by DNSSEC?

With neither HTTPS nor PGP, what attack is stopped by DNSSEC?

Getting out of the mess

State-of-the-art ECC is fast enough to authenticate and encrypt every packet.

Deployed: DNSCurve protects DNS packets, server→cache.

Deployed: DNSCrypt protects DNS packets, cache→client.

Work in progress: HTTPCurve protects HTTP packets.

ns the user's data.
ned web pages and email
ected against
ving servers,
inst network attackers.

GP, what attack
ed by DNSSEC?

TTPS but not PGP, what
s stopped by DNSSEC?

ither HTTPS nor PGP,
tack is stopped by
C?

Getting out of the mess

State-of-the-art ECC
is fast enough to
authenticate and encrypt
every packet.

Deployed: DNSCurve protects
DNS packets, server→cache.

Deployed: DNSCrypt protects
DNS packets, cache→client.

Work in progress: HTTPCurve
protects HTTP packets.

Crypto i
handled

Adminis
into nam

Need ne
but no n
server sc
database
web inte

Easy to
easy to

er's data.
pages and email
nst
rs,
rk attackers.

ttack
SSEC?

not PGP, what
by DNSSEC?

TPS nor PGP,
pped by

Getting out of the mess

State-of-the-art ECC
is fast enough to
authenticate and encrypt
every packet.

Deployed: DNSCurve protects
DNS packets, server→cache.

Deployed: DNSCrypt protects
DNS packets, cache→client.

Work in progress: HTTPCurve
protects HTTP packets.

Crypto is at edge o
handled by simple

Administrator puts
into name of serve

Need new DNS ca
but no need to cha
server software,
database-managem
web interfaces, etc

Easy to implement
easy to deploy.

email

rs.

what
EC?

GP,

Getting out of the mess

State-of-the-art ECC
is fast enough to
authenticate and encrypt
every packet.

Deployed: DNSCurve protects
DNS packets, server→cache.

Deployed: DNSCrypt protects
DNS packets, cache→client.

Work in progress: HTTPCurve
protects HTTP packets.

Crypto is at edge of network
handled by simple proxy.

Administrator puts public ke
into name of server.

Need new DNS cache softw
but no need to change
server software,
database-management softw
web interfaces, etc.

Easy to implement,
easy to deploy.

Getting out of the mess

State-of-the-art ECC
is fast enough to
authenticate and encrypt
every packet.

Deployed: DNSCurve protects
DNS packets, server$\to$cache.

Deployed: DNSCrypt protects
DNS packets, cache$\to$client.

Work in progress: HTTPCurve
protects HTTP packets.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

out of the mess

-the-art ECC
nough to
cate and encrypt
cket.

d: DNSCurve protects
ckets, server→cache.

d: DNSCrypt protects
ckets, cache→client.

progress: HTTPCurve
HTTP packets.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No preco

mess

CC

encrypt

urve protects
ver→cache.

ypt protects
he→client.

HTTPCurve
ackets.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No precomputatio

cts
.
ts
.
rve

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No precomputation.

Crypto is at edge of network, handled by simple proxy.

Administrator puts public key into name of server.

Need new DNS cache software but no need to change server software, database-management software, web interfaces, etc.

Easy to implement, easy to deploy.

No precomputation.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No precomputation.

No problems with
dynamic data.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No precomputation.

No problems with
dynamic data.

No problems with
old data: all results
are guaranteed to be fresh.

Crypto is at edge of network,
handled by simple proxy.

Administrator puts public key
into name of server.

Need new DNS cache software
but no need to change
server software,
database-management software,
web interfaces, etc.

Easy to implement,
easy to deploy.

No precomputation.

No problems with
dynamic data.

No problems with
old data: all results
are guaranteed to be fresh.

No problems with
nonexistent data,
database leaks, etc.

Crypto is at edge of network, handled by simple proxy.

Administrator puts public key into name of server.

Need new DNS cache software but no need to change server software, database-management software, web interfaces, etc.

Easy to implement, easy to deploy.

No precomputation.

No problems with dynamic data.

No problems with old data: all results are guaranteed to be fresh.

No problems with nonexistent data, database leaks, etc.

Packets are small. Smaller amplification than existing protocols.

s at edge of network,
by simple proxy.

trator puts public key
ne of server.

w DNS cache software
need to change
oftware,
e-management software,
erfaces, etc.

implement,
deploy.

No precomputation.

No problems with
dynamic data.

No problems with
old data: all results
are guaranteed to be fresh.

No problems with
nonexistent data,
database leaks, etc.

Packets are small.
Smaller amplification
than existing protocols.

DNSCur
and HTT
add real
PGP-sig

Improve
e.g., is t
`firstai`
`diabete`

Improve
e.g., fres

Improve
attacker
doesn't

of network,
proxy.

s public key
er.

che software
ange

ment software,
c.

t,

No precomputation.

No problems with dynamic data.

No problems with old data: all results are guaranteed to be fresh.

No problems with nonexistent data, database leaks, etc.

Packets are small. Smaller amplification than existing protocols.

DNSCurve and DN
and HTTPCurve a
add real security e
PGP-signed web p

Improved confiden
e.g., is the user ac
`firstaid.webmd.`
`diabetes.webmd.`

Improved integrity
e.g., freshness.

Improved availabil
attacker forging a
doesn't break conn

k,

ey

are

vare,

No precomputation.

No problems with
dynamic data.

No problems with
old data: all results
are guaranteed to be fresh.

No problems with
nonexistent data,
database leaks, etc.

Packets are small.
Smaller amplification
than existing protocols.

DNSCurve and DNSCrypt
and HTTPCurve and SMTP
add real security even to
PGP-signed web pages, ema

Improved confidentiality:
e.g., is the user accessing
`firstaid.webmd.com` or
`diabetes.webmd.com`?

Improved integrity:
e.g., freshness.

Improved availability:
attacker forging a packet
doesn't break connections.

No precomputation.

No problems with
dynamic data.

No problems with
old data: all results
are guaranteed to be fresh.

No problems with
nonexistent data,
database leaks, etc.

Packets are small.
Smaller amplification
than existing protocols.

DNSCurve and DNSCrypt
and HTTPCurve and SMTPCurve
add real security even to
PGP-signed web pages, email.

Improved confidentiality:
e.g., is the user accessing
`firstaid.webmd.com` or
`diabetes.webmd.com`?

Improved integrity:
e.g., freshness.

Improved availability:
attacker forging a packet
doesn't break connections.