The DNS security mess

D. J. Bernstein

University of Illinois at Chicago

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.

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.

The hard way
for attackers to do this:

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

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.

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 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.

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 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.”

"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!

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!

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?

November 2017: 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-cloud-e1.
    googledomains.com.
```

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
```

np90u3h.org. 86400 IN RRS
IG NSEC3 7 2 86400 201712
13231839 20171122221839 1
862 org. GfxhBt4c+7E7OUyE
cnf5CwwthLUR0O70GiRGYK5f0
I6nmIG/yELCJGSa 9lcVp5JcS
S1YwPad9aQRrVedZXAV6qFnPi
qowaJMWJ207DHbFD02Lus7 M4
jyqeZh7cMXLN0HxQ1q0zW/j4g
086z1nCr5pWsUlme76hB3lz9E
XBba nvk=

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

0PP0BENBFCGBMB6RGT2JDC21E
 A RRSIG

bgca0g0ug0p6o7425emkt9ue4
qng3p2f.org. 86400 IN RRS
IG NSEC3 7 2 86400 201712
08152932 20171117142932 1
862 org. RZIhCS7+uAxG39iO
lzEH+88fDlJ8x3uYPtHt/K3EE
xgvymEadj77bza2 yuj5nJ0t0
d+LkHm8KHOAFLoRmt24WiZrkP
FERTcC3OW6+mhH/rF1sqGm Hj
4EsixBFa6rYLdqR/NylQxCtA5
rfjnFAcHCAfFEfor0MR9qtARU
aWxu RYU=

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 657-byte IP packet.

See more DNSSEC data:

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

Sends 74-byte IP packet, receives two IP fragments totalling 2653 bytes.

Interlude: the attacker's view

What happens if we aim
this data at someone else?

Interlude: the attacker's view

What happens if we aim
this data at someone else?

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
```

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
```

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?

Can that really be true?
$>$2000 DNSSEC servers
around the Internet, each
providing $>30\times$ 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
```

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

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.

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.

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.

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.

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.

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.

2017: No SecSpider downloads???

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.

2017: No SecSpider downloads??? Exercise: Collect+publish data.

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

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.”

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

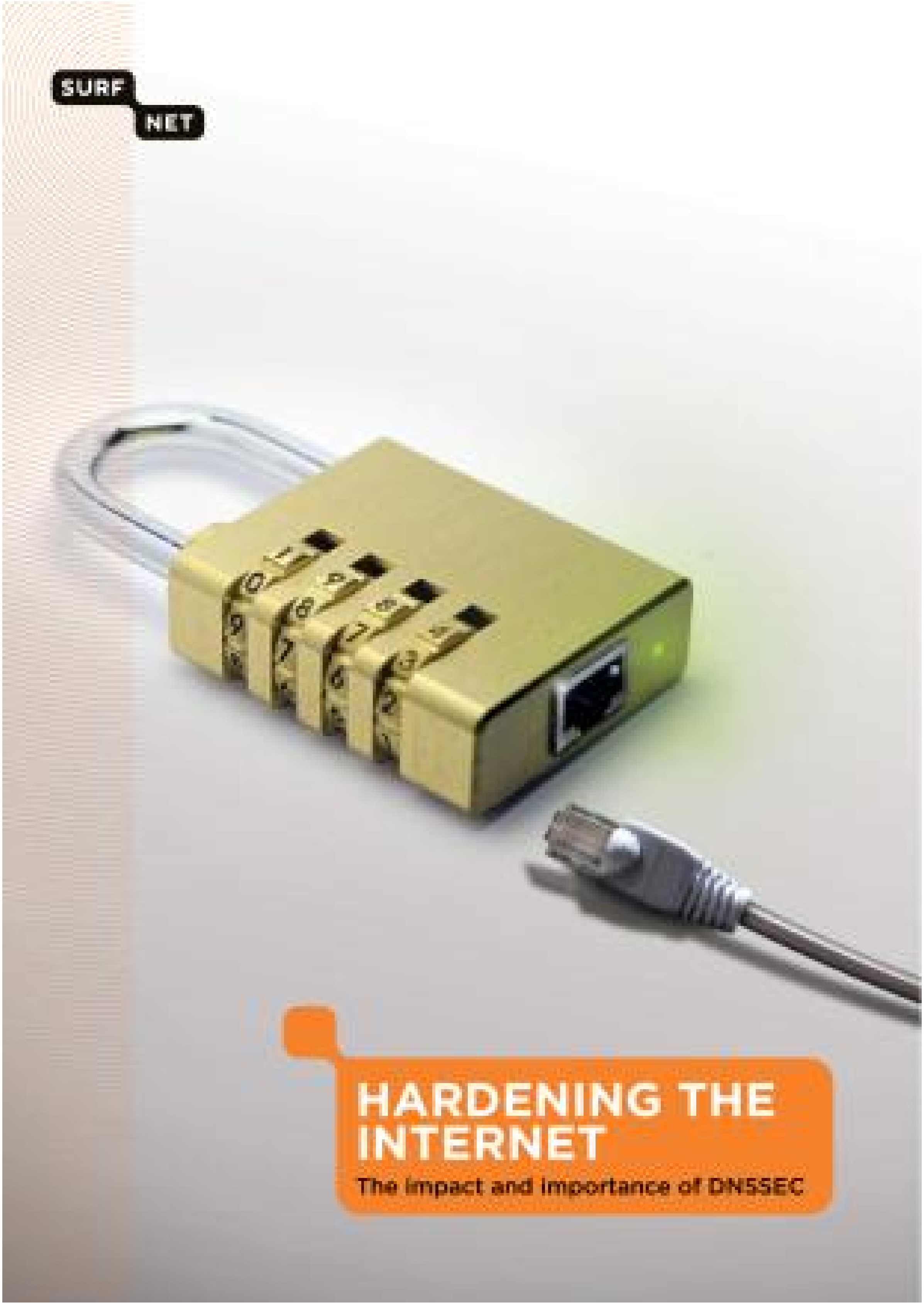
RFC 4033 doesn't say
"DNSSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

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:

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.

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

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."

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`.

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.

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.

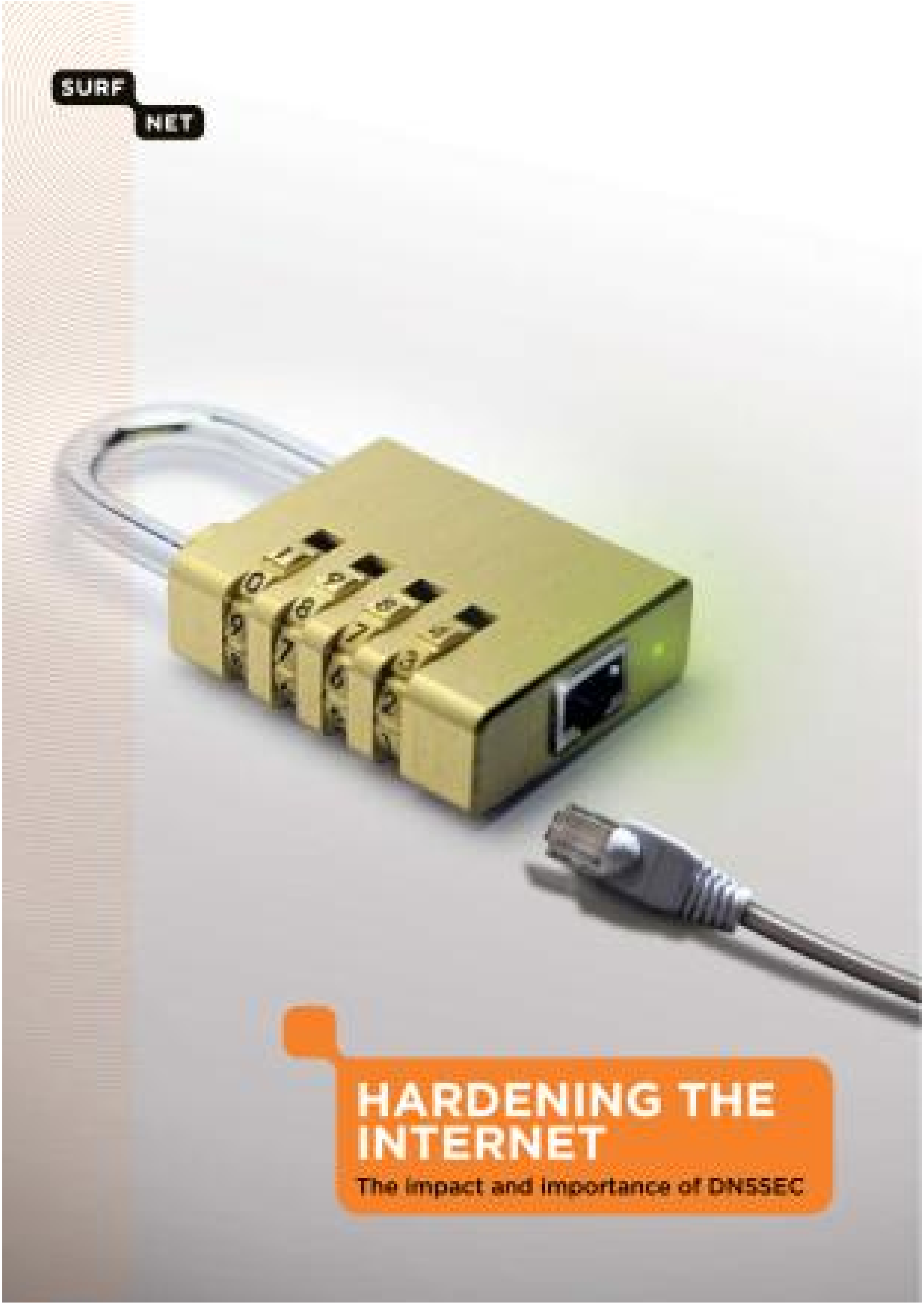
Fact: DNSSEC "verification" won't notice the change. The signatures say *nothing* about the NS+A records. *The forgery will be accepted.*

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."

"DNSSEC: Built, not plugged in."



What went wrong?

Rushed development process?

What went wrong?

Rushed development process?

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

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.

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 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.

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.

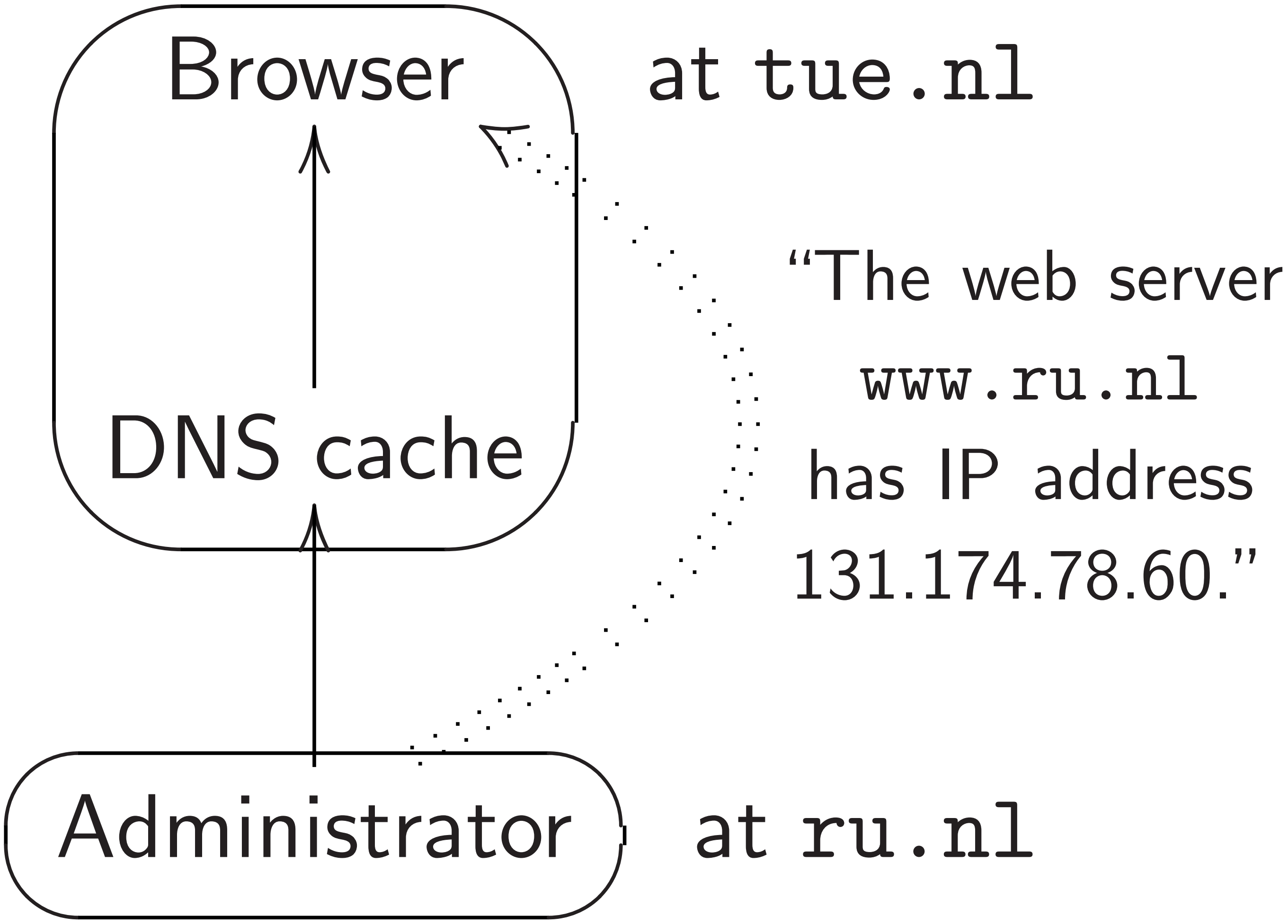
DNSSEC made breakable choices such as 640-bit RSA for no reason other than fear of overload.

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

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

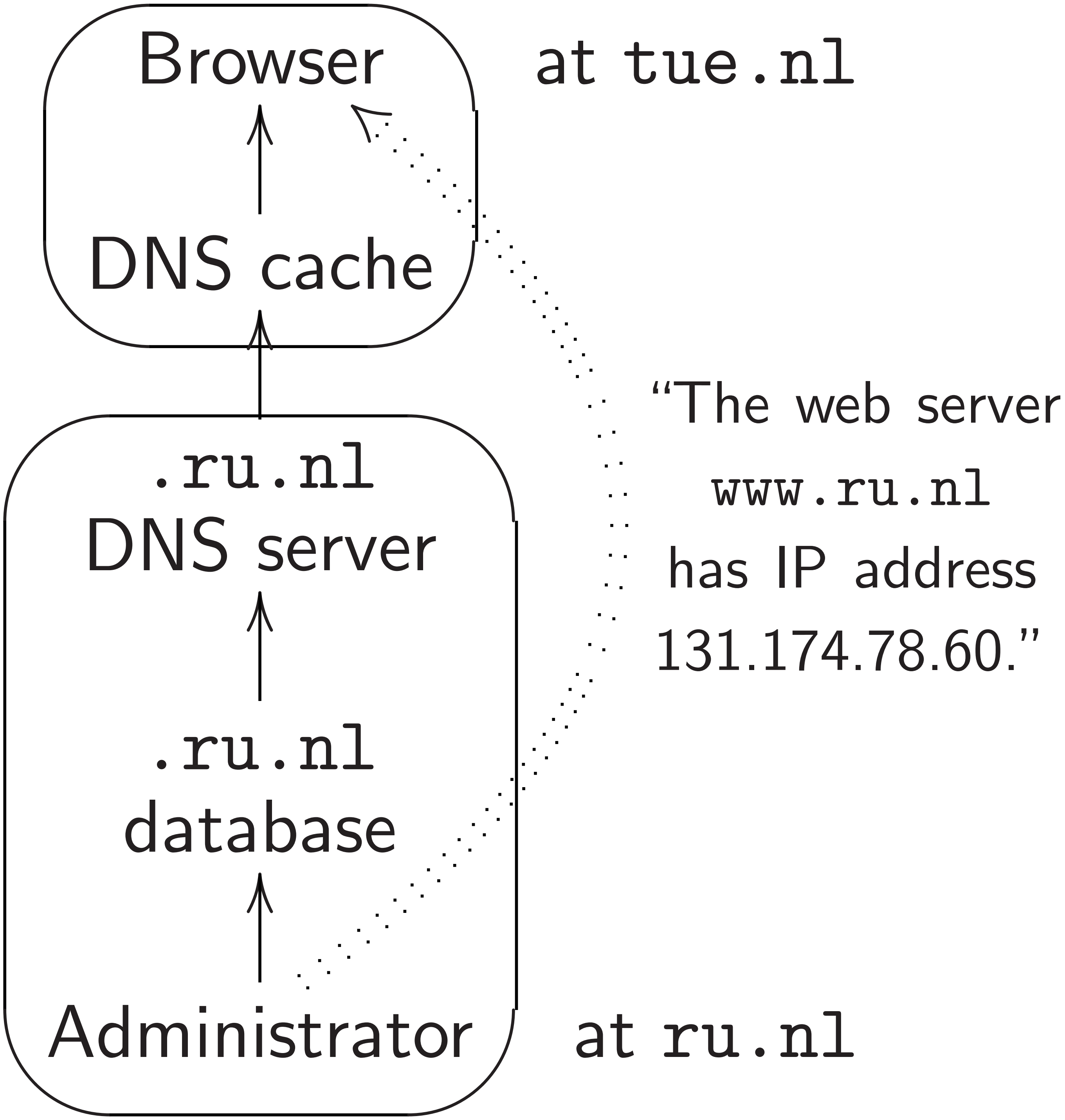
DNS architecture

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

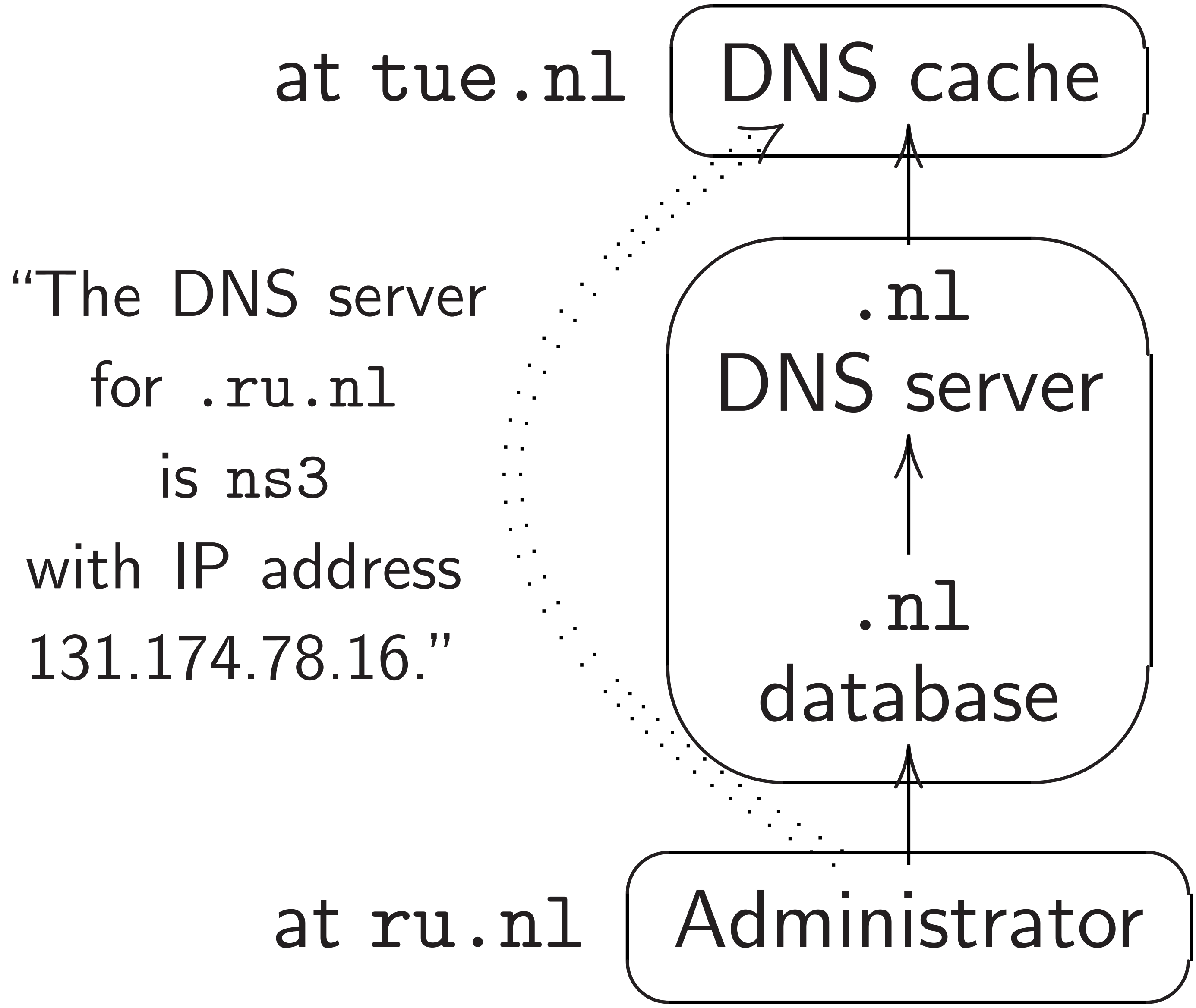


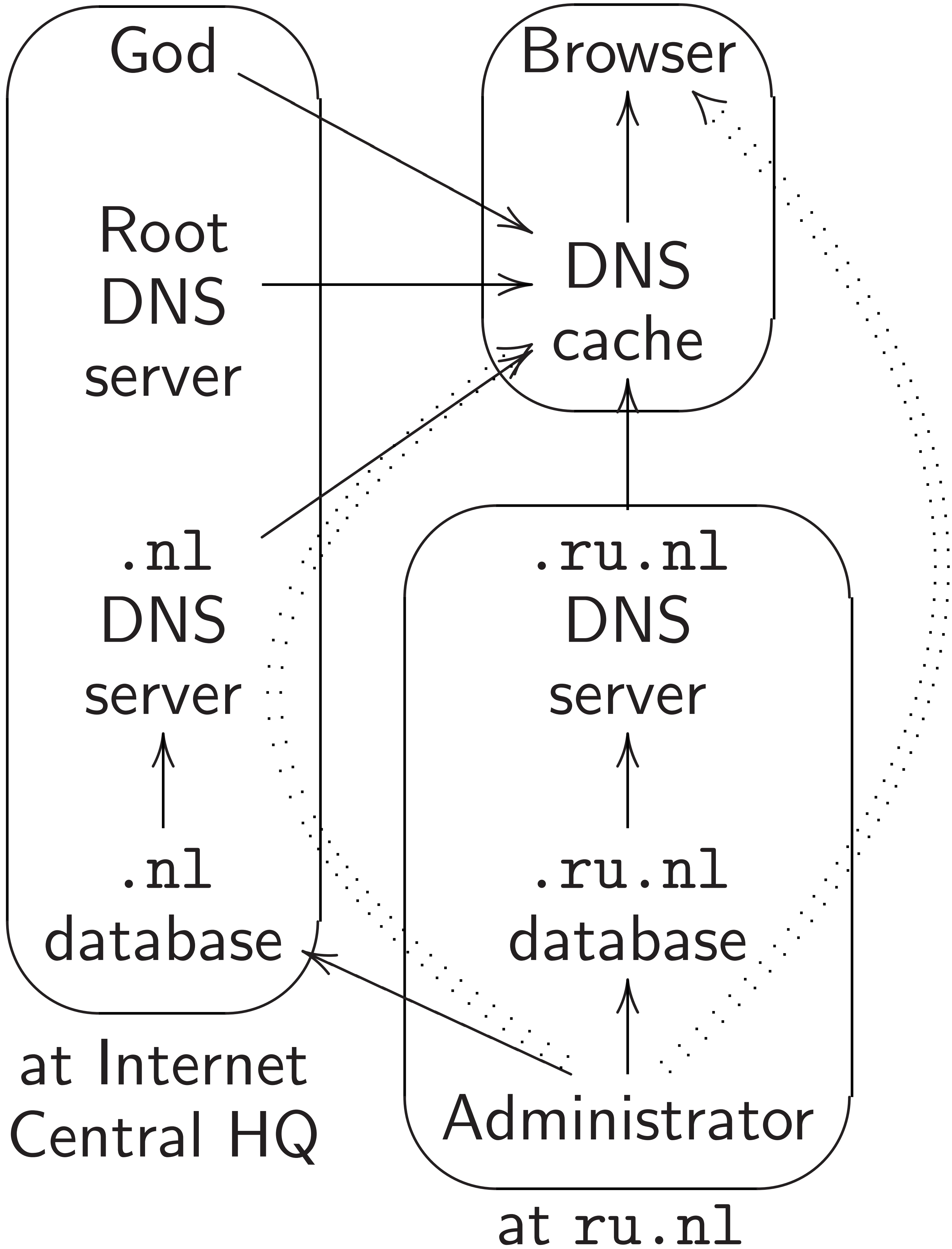
Cache pulls data from administrator if it doesn't already have the data.

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



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




God
Browser
Root DNS server
DNS cache
.nl DNS server
.ru.nl DNS server
.nl database
.ru.nl database
Administrator
at Internet Central HQ
at ru.nl

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 6GB database can produce 40GB 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.

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.

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.

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.

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.

A few DNSSEC suicide examples:

2010.09.02: `.us` killed itself.

2012.02.28, ISC's Evan Hunt:
"`dnssec-accept-expired yes`"

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.

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.

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.

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.

2016.10.24: `dnssec-tools.org` killed itself.

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.

2016.10.24: `dnssec-tools.org` killed itself.

Many more: see `ianix.com/pub/dnssec-outages.html`.

What about *nonexistent* data?

What about *nonexistent* data?

Does Nijmegen administrator precompute signatures on "`aaaaa.ru.nl does not exist`", "`aaaab.ru.nl does not exist`", etc.?

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."

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.

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`.

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.

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.

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` …"

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).

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."

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.

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.

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?

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 . . .

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?

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.

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.

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?

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?

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$\rightarrow$cache.

Deployed: DNSCrypt protects DNS packets, cache$\rightarrow$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.

No precomputation.

No precomputation.

No problems with
dynamic data.

No precomputation.

No problems with dynamic data.

No problems with old data: all results are guaranteed to be fresh.

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.

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.