Understanding DNSCurve

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimer: I haven't
released DNSCurve software yet.

But you can try prototypes:
`@mdempsky`'s DNSCurve cache,
`@hhavt`'s CurveDNS server.

See also related projects: NaCl,
DNSCrypt, CurveCP, MinimaLT.
Varying release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

Understanding DNSCurve

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimer: I haven't
released DNSCurve software yet.

But you can try prototypes:
`@mdempsky`'s DNSCurve cache,
`@hhavt`'s CurveDNS server.

See also related projects: NaCl,
DNSCrypt, CurveCP, MinimaLT.
Varying release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

Understanding DNSCurve

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimer: I haven't
released DNSCurve software yet.

But you can try prototypes:
`@mdempsky`'s DNSCurve cache,
`@hhavt`'s CurveDNS server.

See also related projects: NaCl,
DNSCrypt, CurveCP, MinimaLT.
Varying release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

Understanding DNSCurve

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimer: I haven't
released DNSCurve software yet.

But you can try prototypes:
`@mdempsky`'s DNSCurve cache,
`@hhavt`'s CurveDNS server.

See also related projects: NaCl,
DNSCrypt, CurveCP, MinimaLT.
Varying release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

Understanding DNSCurve

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimer: I haven't
released DNSCurve software yet.

But you can try prototypes:
`@mdempsky`'s DNSCurve cache,
`@hhavt`'s CurveDNS server.

See also related projects: NaCl,
DNSCrypt, CurveCP, MinimaLT.
Varying release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

2 `ns2.twitter.com` → browser:
`twitter.com A 199.16.156.38`

---

anding DNSCurve

ernstein

ty of Illinois at Chicago &
che Universiteit Eindhoven

---

ner: I haven't
DNSCurve software yet.

can try prototypes:
sky's DNSCurve cache,
s CurveDNS server.

related projects: NaCl,
pt, CurveCP, MinimaLT.
release levels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

2 `ns2.twitter.com` → browser:
`twitter.com A 199.16.156.38`

−3 com
twitter
ns2...

NSCurve

is at Chicago &
siteit Eindhoven

---

n't
e software yet.

rototypes:
SCurve cache,
NS server.

rojects: NaCl,
CP, MinimaLT.
vels.

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

2 `ns2.twitter.com` → browser:
`twitter.com A 199.16.156.38`

−3 com admin →
`twitter.com NS`
`ns2... A 204.13`

ago &
hoven

yet.

che,

aCl,
aLT.

DNS in a nutshell

1 Browser → DNS:

`twitter.com?`

2 DNS → browser:

`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:

`twitter.com?`

2 `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.co`

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

DNS in a nutshell

1 Browser → DNS:

`twitter.com?`

2 DNS → browser:

`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:

`twitter.com?`

2 `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

DNS in a nutshell

1 Browser → DNS:

`twitter.com?`

2 DNS → browser:

`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:

`twitter.com?`

2 `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

−2 Browser → `f.ns.com`:

`twitter.com?`

DNS in a nutshell

1 Browser → DNS:

`twitter.com?`

2 DNS → browser:

`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:

`twitter.com?`

2 `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

−2 Browser → `f.ns.com`:

`twitter.com?`

−1 `f.ns.com` → browser:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

2 `ns2.twitter.com` → browser:
`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:
`twitter.com NS ns2...`
`ns2... A 204.13.250.34`

−2 Browser → `f.ns.com`:
`twitter.com?`

−1 `f.ns.com` → browser:
`twitter.com NS ns2...`
`ns2... A 204.13.250.34`

0 Twitter admin → `ns2`:
`twitter.com A 199.16.156.38`

DNS in a nutshell

1 Browser → DNS:
`twitter.com?`

2 DNS → browser:
`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:
`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:
`twitter.com?`

2 `ns2.twitter.com` → browser:
`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:
`twitter.com NS ns2...`
`ns2... A 204.13.250.34`

−2 Browser → `f.ns.com`:
`twitter.com?`

−1 `f.ns.com` → browser:
`twitter.com NS ns2...`
`ns2... A 204.13.250.34`

0 Twitter admin → `ns2`:
`twitter.com A 199.16.156.38`

1 Browser → 204.13.250.34:
`twitter.com?`

DNS in a nutshell

1 Browser → DNS:

`twitter.com?`

2 DNS → browser:

`twitter.com A 199.16.156.38`

---

0 Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

1 Browser → `ns2.twitter.com`:

`twitter.com?`

2 `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

−3 `com` admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

−2 Browser → `f.ns.com`:

`twitter.com?`

−1 `f.ns.com` → browser:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

0 Twitter admin → `ns2`:

`twitter.com A 199.16.156.38`

1 Browser → 204.13.250.34:

`twitter.com?`

2 204.13.250.34 → browser:

`twitter.com A 199.16.156.38`

a nutshell

ser → DNS:

r.com?

→ browser:

r.com A 199.16.156.38

---

n → ns2.twitter.com:

r.com A 199.16.156.38

ser → ns2.twitter.com:

r.com?

twitter.com → browser:

r.com A 199.16.156.38

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often ev

- Mayb
  where
  Has t

S:

r:

99.16.156.38

---

twitter.com:

99.16.156.38

.twitter.com:

com → browser:

99.16.156.38

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often even more s

- Maybe browser
  where .com ser
  Has to ask root

6.38

com:

6.38

owser:

6.38

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often even more steps:

- Maybe browser doesn't k
  where .com server is.
  Has to ask root server.

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often even more steps:

- Maybe browser doesn't know where .com server is.
  Has to ask root server.

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often even more steps:

- Maybe browser doesn't know where .com server is.
  Has to ask root server.
- twitter.com server name is actually ns2.p34.dynect.net.
  Is browser allowed to accept ns2.p34.dynect.net address from the .com server?
  Does it have to ask .net?

−3 com admin → f.ns.com:

twitter.com NS ns2...

ns2... A 204.13.250.34

−2 Browser → f.ns.com:

twitter.com?

−1 f.ns.com → browser:

twitter.com NS ns2...

ns2... A 204.13.250.34

0 Twitter admin → ns2:

twitter.com A 199.16.156.38

1 Browser → 204.13.250.34:

twitter.com?

2 204.13.250.34 → browser:

twitter.com A 199.16.156.38

Often even more steps:

- Maybe browser doesn't know where .com server is.
  Has to ask root server.
- twitter.com server name is actually ns2.p34.dynect.net.
  Is browser allowed to accept ns2.p34.dynect.net address from the .com server?
  Does it have to ask .net?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

admin → f.ns.com:

r.com NS ns2...

A 204.13.250.34

wser → f.ns.com:

r.com?

s.com → browser:

r.com NS ns2...

A 204.13.250.34

er admin → ns2:

r.com A 199.16.156.38

ser → 204.13.250.34:

r.com?

3.250.34 → browser:

r.com A 199.16.156.38

Often even more steps:

- Maybe browser doesn't know where .com server is.
  Has to ask root server.
- twitter.com server name is actually ns2.p34.dynect.net.
  Is browser allowed to accept ns2.p34.dynect.net address from the .com server?
  Does it have to ask .net?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

DNS in

The use
twitter

The use
pull twe
push tw

f.ns.com:

ns2...

.250.34

.ns.com:

browser:

ns2...

.250.34

→ ns2:

99.16.156.38

.13.250.34:

→ browser:

99.16.156.38

Often even more steps:

- Maybe browser doesn't know where .com server is.
  Has to ask root server.
- twitter.com server name is actually ns2.p34.dynect.net.
  Is browser allowed to accept ns2.p34.dynect.net address from the .com server?
  Does it have to ask .net?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

DNS in the real w

The user doesn't w
twitter.com's IP

The user wants to
pull tweets from T
push tweets to Tw

m:

6.38

4:

:

6.38

Often even more steps:

- Maybe browser doesn't know where `.com` server is.
  Has to ask root server.
- `twitter.com` server name is actually `ns2.p34.dynect.net`.
  Is browser allowed to accept `ns2.p34.dynect.net` address from the `.com` server?
  Does it have to ask `.net`?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

DNS in the real world

The user doesn't want `twitter.com`'s IP address.

The user wants to pull tweets from Twitter, push tweets to Twitter.

Often even more steps:

- Maybe browser doesn't know where `.com` server is.
  Has to ask root server.
- `twitter.com` server name is actually `ns2.p34.dynect.net`.
  Is browser allowed to accept `ns2.p34.dynect.net` address from the `.com` server?
  Does it have to ask `.net`?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

DNS in the real world

The user doesn't want `twitter.com`'s IP address.

The user wants to pull tweets from Twitter, push tweets to Twitter.

Often even more steps:

- Maybe browser doesn't know where `.com` server is.
  Has to ask root server.
- `twitter.com` server name is actually `ns2.p34.dynect.net`.
  Is browser allowed to accept `ns2.p34.dynect.net` address from the `.com` server?
  Does it have to ask `.net`?
- Browser actually pulls from a laptop-wide DNS cache.
  Or a site-wide DNS cache.

DNS in the real world

The user doesn't want `twitter.com`'s IP address.

The user wants to pull tweets from Twitter, push tweets to Twitter.

The big picture:
DNS is just one small part of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

ven more steps:

e browser doesn't know
e `.com` server is.
o ask root server.

`ter.com` server name is
lly `ns2.p34.dynect.net`.
owser allowed to accept
`p34.dynect.net` address
the `.com` server?
it have to ask `.net`?

ser actually pulls from
top-wide DNS cache.
site-wide DNS cache.

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
DNS is just one small part
of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Inte
User ask

steps:

doesn't know
ver is.
t server.

server name is
34.dynect.net.
wed to accept
ct.net address
server?
ask .net?

y pulls from
DNS cache.
DNS cache.

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
DNS is just one small part
of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Internet proto
User asks browser

now

e is
t.net.
ept
ddress

?

m
.
e.

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
DNS is just one small part
of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Internet protocol exam
User asks browser for

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
DNS is just one small part
of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Internet protocol example:
User asks browser for

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
DNS is just one small part
of any real Internet protocol.

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

DNS in the real world

The user doesn't want
`twitter.com`'s IP address.

The user wants to
pull tweets from Twitter,
push tweets to Twitter.

The big picture:
**DNS is just one small part
of any real Internet protocol.**

Typical examples:
HTTP starts with DNS.
SMTP starts with DNS.
SSH starts with DNS.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

the real world

r doesn't want
r.com's IP address.

r wants to
ets from Twitter,
eets to Twitter.

picture:
just one small part
real Internet protocol.

examples:
tarts with DNS.
tarts with DNS.
rts with DNS.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What do

Crypto g
integrity
for **the**

Security
is irrelev
protect

orld

want
' address.

Twitter,
witter.

small part
net protocol.

DNS.
DNS.
NS.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What does DNS s

Crypto goals: con
integrity, and avai
for **the user's co**

Security for *IP ad*
is irrelevant unless
protect user comm

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What does DNS security me

Crypto goals: confidentiality
integrity, and availability
for **the user's communicat**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication

t
col.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Real Internet protocol example:
User asks browser for
`http://theguardian.com`.

Many levels of redirection:
root DNS $\mapsto$
`.com` DNS $\mapsto$
`.theguardian.com` DNS $\mapsto$
`http://theguardian.com` $\mapsto$
`http://www.theguardian.com` $\mapsto$
`http://www.theguardian.com/uk`.

And then the hard work begins:
browser receives page,
displays page for user.

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

ernet protocol example:
ks browser for
/theguardian.com.

vels of redirection:
S ↦
NS ↦
ardian.com DNS ↦
/theguardian.com ↦
/www.theguardian.com ↦
/www.theguardian.com/uk.

n the hard work begins:
receives page,
page for user.

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

Example

"You ca
Our DN
by a Ha
in a fort
protecte
Signing
3 out of
by VeriS

ocol example:
for
dian.com.

irection:

om DNS ↦
dian.com ↦
guardian.com ↦
guardian.com/uk.

l work begins:
age,
user.

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

Example of bogus

"You can't trust o
Our DNS data is s
by a Hardware Sec
in a fortress in Ma
protected by mach
Signing procedure
3 out of 16 smart
by VeriSign Trust

ple:

→

↦

.com ↦

.com/uk.

gins:

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

Example of bogus "security"

"You can't trust online serve
Our DNS data is signed offli
by a Hardware Security Mod
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

What does DNS security mean?

Crypto goals: confidentiality,
integrity, and availability
for **the user's communication.**

Security for *IP addresses*
is irrelevant unless it helps
protect user communication.

Consider DNSSEC marketing:
`isc.org` is "signed" by DNSSEC.

Reality: What DNSSEC signs
is an IP-address redirection:
`isc.org A 149.20.64.69`.
This is meaningless for users.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

oes DNS security mean?

goals: confidentiality,
, and availability
user's communication.

for *IP addresses*
ant unless it helps
user communication.

r DNSSEC marketing:
g is "signed" by DNSSEC.

What DNSSEC signs
-address redirection:
g `A 149.20.64.69`.
meaningless for users.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasion
broadcas
so offline
might he

ecurity mean?

fidentiality,
lability
mmunication.

dresses
s it helps
nunication.

C marketing:
d" by DNSSEC.

NSSEC signs
edirection:
0.64.69.
ss for users.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasionally **user**
broadcast+static+
so offline creation
might help protect

ean?

,

ion.

g:
SSEC.

ns

s.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasionally **user data** is
broadcast+static+single-sou
so offline creation and signi
might help protect integrity.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasionally **user data** is
broadcast+static+single-source,
so offline creation and signing
might help protect integrity.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasionally **user data** is
broadcast+static+single-source,
so offline creation and signing
might help protect integrity.

But this is a rare corner case.
Offline creation and signing:
impossible for most user data.

Example of bogus "security":

"You can't trust online servers.
Our DNS data is signed offline
by a Hardware Security Module
in a fortress in Maryland
protected by machine guns.
Signing procedure requires
3 out of 16 smart cards held
by VeriSign Trust Managers."

Does this protect users? No!
The **web server** is online,
and most web pages are dynamic.
The **mail server** is online.
The **shell server** is online.

Occasionally **user data** is
broadcast+static+single-source,
so offline creation and signing
might help protect integrity.

But this is a rare corner case.
Offline creation and signing:
impossible for most user data.

By insisting on signatures,
DNSSEC creates problems for
lookups of dynamic DNS data;
lookups of nonexistent names;
speed; robustness; availability;
freshness; confidentiality.
Analogy: imagine HTTPSEC.

e of bogus "security":

n't trust online servers.
S data is signed offline
rdware Security Module
ress in Maryland
d by machine guns.
procedure requires
16 smart cards held
ign Trust Managers."

is protect users? No!
b server is online,
st web pages are dynamic.
il server is online.
ll server is online.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case.
Offline creation and signing: impossible for most user data.

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality.
Analogy: imagine HTTPSEC.

DNSCur

“security”:

nline servers.
signed offline
curity Module
aryland
hine guns.
requires
cards held
Managers.”

users? No!
s online,
ges are dynamic.
s online.
s online.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case. **Offline creation and signing: impossible for most user data.**

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality. Analogy: imagine HTTPSEC.

DNSCurve, Curve

:

ers.
ine
dule

„

!

namic.

Occasionally **user data** is
broadcast+static+single-source,
so offline creation and signing
might help protect integrity.

But this is a rare corner case.
Offline creation and signing:
impossible for most user data.

By insisting on signatures,
DNSSEC creates problems for
lookups of dynamic DNS data;
lookups of nonexistent names;
speed; robustness; availability;
freshness; confidentiality.
Analogy: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case. **Offline creation and signing: impossible for most user data.**

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality. Analogy: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case. **Offline creation and signing: impossible for most user data.**

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality. Analogy: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Most Internet connections today have no cryptographic protection.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case. **Offline creation and signing: impossible for most user data.**

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality. Analogy: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Most Internet connections today have no cryptographic protection.

The big plan: replace DNS with DNSCurve, TCP with CurveCP, HTTP with HTTPCurve, etc.

Occasionally **user data** is broadcast+static+single-source, so offline creation and signing might help protect integrity.

But this is a rare corner case. **Offline creation and signing: impossible for most user data.**

By insisting on signatures, DNSSEC creates problems for lookups of dynamic DNS data; lookups of nonexistent names; speed; robustness; availability; freshness; confidentiality. Analogy: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Most Internet connections today have no cryptographic protection.

The big plan: replace DNS with DNSCurve, TCP with CurveCP, HTTP with HTTPCurve, etc.

All client data is authenticated + encrypted to server's public key from client's public key.

All server data is authenticated + encrypted to client's public key from server's public key.

nally **user data** is
st+static+single-source,
e creation and signing
elp protect integrity.

is a rare corner case.
creation and signing:
ble for most user data.

ting on signatures,
C creates problems for
of dynamic DNS data;
of nonexistent names;
obustness; availability;
s; confidentiality.
: imagine HTTPSEC.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto l
to netwo

Each pa
encrypte
Each pa
decrypte
after it i

data is
-single-source,
and signing
t integrity.

corner case.
and signing:
ost user data.

natures,
problems for
c DNS data;
stent names;
availability;
ntiality.
HTTPSEC.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto layer is ver
to network layer.

Each packet is aut
encrypted just bef
Each packet is ver
decrypted immedia
after it is received

urce,
ng
e.
ng:
lata.
or
ta;
es;
y;
C.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto layer is very close
to network layer.

Each packet is authenticated
encrypted just before it is se
Each packet is verified +
decrypted immediately
after it is received.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

DNSCurve, CurveCP, etc.

Most Internet connections today
have no cryptographic protection.

The big plan: replace
DNS with DNSCurve,
TCP with CurveCP,
HTTP with HTTPCurve, etc.

All client data is authenticated +
encrypted to server's public key
from client's public key.

All server data is authenticated +
encrypted to client's public key
from server's public key.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

rve, CurveCP, etc.

ternet connections today
cryptographic protection.

plan: replace
th DNSCurve,
th CurveCP,
with HTTPCurve, etc.

t data is authenticated +
ed to server's public key
ent's public key.

er data is authenticated +
ed to client's public key
ver's public key.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How doe
client's p

CP, etc.

nections today
phic protection.

ace
rve,
P,
PCurve, etc.

uthenticated +
er's public key
c key.

authenticated +
t's public key
ic key.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server o
client's public key

oday
ction.

c.

ted +
key

ted +
key

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?
Client sends it with first packet.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?
Client sends it with first packet.

How does client obtain
server's public key?

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?
Client sends it with first packet.

How does client obtain
server's public key?
Client already had mechanism
to obtain server address.
Server sneaks public key
into that mechanism.

Crypto layer is very close
to network layer.

Each packet is authenticated +
encrypted just before it is sent.
Each packet is verified +
decrypted immediately
after it is received.

Much less invasive than DNSSEC
for DNS protocol, DNS databases,
DNS implementations.
Also easy for HTTP etc.

Separate authenticator on every
packet also improves availability.
No more RST attacks.

How does server obtain
client's public key?
Client sends it with first packet.

How does client obtain
server's public key?
Client already had mechanism
to obtain server address.
Server sneaks public key
into that mechanism.

No extra packets.
Serious crypto for each packet,
but state-of-the-art crypto
(Curve25519, Salsa20, Poly1305)
easily keeps up with the network.