# 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

$\boxed{1}$ Browser → DNS:

`twitter.com?`

# DNS in a nutshell

$\boxed{1}$ Browser → DNS:

`twitter.com?`

$\boxed{2}$ DNS → browser:

`twitter.com A 199.16.156.38`

# DNS in a nutshell

$\boxed{1}$ Browser → DNS:

`twitter.com?`

$\boxed{2}$ DNS → browser:

`twitter.com A 199.16.156.38`

---

$\boxed{0}$ Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

# DNS in a nutshell

$\boxed{1}$ Browser → DNS:

`twitter.com?`

$\boxed{2}$ DNS → browser:

`twitter.com A 199.16.156.38`

---

$\boxed{0}$ Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

$\boxed{1}$ Browser → `ns2.twitter.com`:

`twitter.com?`

# DNS in a nutshell

$\boxed{1}$ Browser → DNS:

`twitter.com?`

$\boxed{2}$ DNS → browser:

`twitter.com A 199.16.156.38`

---

$\boxed{0}$ Admin → `ns2.twitter.com`:

`twitter.com A 199.16.156.38`

$\boxed{1}$ Browser → `ns2.twitter.com`:

`twitter.com?`

$\boxed{2}$ `ns2.twitter.com` → browser:

`twitter.com A 199.16.156.38`

---

$\boxed{-3}$ com admin $\to$ `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{-3}$ com admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{-2}$ Browser → `f.ns.com`:

`twitter.com?`

$\boxed{-3}$ com admin $\rightarrow$ `f.ns.com`:

`twitter.com` NS `ns2...`

`ns2... A 204.13.250.34`

$\boxed{-2}$ Browser $\rightarrow$ `f.ns.com`:

`twitter.com?`

$\boxed{-1}$ `f.ns.com` $\rightarrow$ browser:

`twitter.com` NS `ns2...`

`ns2... A 204.13.250.34`

$\boxed{-3}$ com admin → `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{-2}$ Browser → `f.ns.com`:

`twitter.com?`

$\boxed{-1}$ `f.ns.com` → browser:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{0}$ Twitter admin → `ns2`:

`twitter.com A 199.16.156.38`

$\boxed{-3}$ com admin $\to$ `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{-2}$ Browser $\to$ `f.ns.com`:

`twitter.com?`

$\boxed{-1}$ `f.ns.com` $\to$ browser:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{0}$ Twitter admin $\to$ `ns2`:

`twitter.com A 199.16.156.38`

$\boxed{1}$ Browser $\to$ 204.13.250.34:

`twitter.com?`

$\boxed{-3}$ com admin $\to$ `f.ns.com`:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{-2}$ Browser $\to$ `f.ns.com`:

`twitter.com?`

$\boxed{-1}$ `f.ns.com` $\to$ browser:

`twitter.com NS ns2...`

`ns2... A 204.13.250.34`

$\boxed{0}$ Twitter admin $\to$ `ns2`:

`twitter.com A 199.16.156.38`

$\boxed{1}$ Browser $\to$ 204.13.250.34:

`twitter.com?`

$\boxed{2}$ 204.13.250.34 $\to$ 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.

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`?

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.

# 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

Real Internet protocol example:

User asks browser for

`http://theguardian.com`.

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.

# 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.

## 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.”

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.

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.**

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.

# DNSCurve, CurveCP, etc.

Most Internet connections today have no cryptographic protection.

# 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.

# 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.

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.

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?

How does server obtain client's public key?

Client sends it with first packet.

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

How does client obtain
server's public key?

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.

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.