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.
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.
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
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

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

-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 NS ns2...
ns2... A 204.13.250.34

1 Browser → ns2.twitter.com:
twitter.com A 199.16.156.38

2 ns2.twitter.com → browser:
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
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 NS ns2...
ns2... A 204.13.250.34

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

2 ns2.twitter.com → browser:
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
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`
3. `com admin` → `f.ns.com`:
   `twitter.com NS ns2... ns2... A 204.13.250.34`
4. Browser → `f.ns.com`:
   `twitter.com`?
5. `f.ns.com` → browser:
   `twitter.com NS ns2... ns2... A 204.13.250.34`
6. `Twitter admin` → `ns2`:
   `twitter.com A 199.16.156.38`
7. 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

Admin → ns2.twitter.com:
twitter.com NS ns2...
ns2... A 204.13.250.34

Browser → f.ns.com:
twitter.com?

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

Twitter admin → ns2:
twitter.com A 199.16.156.38

Browser → 204.13.250.34:
twitter.com?

204.13.250.34 → browser:
twitter.com A 199.16.156.38
DNS in a nutshell

User → DNS:
twitter.com?

DNS → browser:
twitter.com A 199.16.156.38

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

Browser → ns2.twitter.com:
twitter.com A 199.16.156.38

Twitter admin → ns2:
twitter.com A 199.16.156.38

Browser → 204.13.250.34:
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.
DNS in a nutshell

1 Browser ! DNS:
   twitter.com?

2 DNS ! browser:
   twitter.com A 199.16.156.38

0 Admin ! ... A 199.16.156.38

1 Browser ! ns2.twitter.com:
   twitter.com?

2 ns2.twitter.com ! 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
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

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.
## 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.
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.
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.
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.
Often even more steps:
The browser doesn’t know where .com server is.
Has to ask root server.

Twitter.com server name is actually ns2.p34.dynect.net.
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.

Real Internet protocol example:
User asks browser for
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.

Real Internet protocol example:

User asks browser
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.
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:
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:

Many levels of redirection:
root DNS →
.com DNS →
.theguardian.com DNS →
http://theguardian.com →
http://www.theguardian.com →
http://www.theguardian.com/uk.

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

A 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:

Many levels of redirection:

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.
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:


Many levels of redirection:
- root DNS ➔
- .com DNS ➔
- .theguardian.com DNS ➔
- http://theguardian.com ➔
- http://www.theguardian.com ➔
- 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:
Many levels of redirection:
root DNS ➔
.com DNS ➔
.theguardian.com DNS ➔
http://theguardian.com ➔
http://www.theguardian.com ➔
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:

Many levels of redirection:
root DNS 7!
.com DNS 7!
.theguardian.com DNS 7!
http://theguardian.com 7!
http://www.theguardian.com 7!
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:

Many levels of redirection:
root DNS ➔
.com DNS ➔
.theguardian.com DNS ➔
http://theguardian.com ➔
http://www.theguardian.com ➔
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.

Many levels of redirection:
root DNS $\rightarrow$
.com DNS $\rightarrow$
.theguardian.com DNS $\rightarrow$
http://theguardian.com $\rightarrow$
http://www.theguardian.com $\rightarrow$
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.
Real Internet protocol example:

Many levels of redirection:
root DNS \[ \rightarrow \]
theguardian.com DNS \[ \rightarrow \]
www.theguardian.com DNS \[ \rightarrow \]
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.

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.”
Real Internet protocol example:
Many levels of redirection:
root DNS 7!
.com DNS 7!
.theguardian.com DNS 7!
http://theguardian.com 7!
http://www.theguardian.com 7!
http://www.theguardian.com/uk 7!

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.

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.
Verisign Trust Managers
hold 3 out of 16 smart cards
required to sign the data.

You can trust online servers.
Our DNS data is signed online
by a Secure DNS service.
Security for IP addresses
is irrelevant unless it helps
protect user communication.

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

What does DNS security mean?
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.
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 mail server is online.
The web server is online.
The shell server is online.

The web server is online, and most web pages are dynamic.

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

Occasionally user data is broadcast+static+single-source, so offline creation and signing might help protect integrity.
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. 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.
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.
Example of bogus “security”:

Don’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.”

Is this protect users? No!

web server is online,
and most web pages are dynamic.

mail server is online.

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.

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.
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.
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.
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.

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.
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.

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.

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.

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.
Occasionally user data is broadcast-static-single-source, so offline creation and signing might help protect integrity. But this is a rare corner case. Online 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.

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.
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.
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.

How does server obtain client’s public key?
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.

How does server obtain client’s public key?
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.

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.