# Cryptography in DNS

D. J. Bernstein
University of Illinois at Chicago

Exercise: How big is the `dig +dnssec -t any se @a.ns.se` response packet?
How big was the query packet?

# Some general questions

Why doesn't the Internet use cryptography?

# Some general questions

Why doesn't the Internet use cryptography?

"The Internet does use cryptography! I just made an SSL connection to my bank."

## Some general questions

Why doesn't the Internet use cryptography?

"The Internet does use cryptography! I just made an SSL connection to my bank."

Indeed, many connections use SSL, Skype, etc.
But *most* connections don't.

# Why is there so much unprotected Internet communication?

Why is there so much unprotected Internet communication?

"Because nobody cares.
Cryptography is pointless.
Attackers are exploiting
buffer overflows; they aren't
intercepting or forging packets."

Why is there so much unprotected Internet communication?

“Because nobody cares.
Cryptography is pointless.
Attackers are exploiting buffer overflows; they aren’t intercepting or forging packets.”

In fact, attackers are forging packets *and* exploiting buffer overflows *and* doing much more. Users want *all* of these problems fixed.

# Why are typical Internet packets unencrypted and unauthenticated?

Why are typical Internet packets unencrypted and unauthenticated?

"It's too easy to write Internet software that exchanges data without any cryptographic protection. Most Internet clients and servers don't know how to make cryptographic connections."

Why are typical Internet packets unencrypted and unauthenticated?

"It's too easy to write Internet software that exchanges data without any cryptographic protection. Most Internet clients and servers don't know how to make cryptographic connections."

True for most protocols.
But let's focus on HTTP.
Most HTTP servers and browsers (Apache, Internet Explorer, Firefox, etc.) support SSL.

# Why is SSL used for only a tiny fraction of all HTTP connections?

Why is SSL used for only a tiny fraction of all HTTP connections?

“Have you ever tried to set up SSL? Do you want to go through all these extra Apache configuration steps? Do you want to pay for a certificate? Do you want to annoy your web-site visitors with self-signed certificates?”

Why is SSL used for only a tiny fraction of all HTTP connections?

“Have you ever tried to set up SSL? Do you want to go through all these extra Apache configuration steps? Do you want to pay for a certificate? Do you want to annoy your web-site visitors with self-signed certificates?”

Indeed, usability is a major issue. Only $\approx$1% of the Apache servers on the Internet have SSL enabled.

But let's focus on Google.

Google has already
paid for a certificate.
Google uses SSL for
`https://mail.google.com`.

But let's focus on Google.

Google has already
paid for a certificate.
Google uses SSL for
`https://mail.google.com`.

If you connect to
`https://www.google.com`,
Google redirects your browser to
`http://www.google.com`.

Why does Google actively *turn off* cryptographic protection?

Why does Google actively
*turn off* cryptographic protection?

"Enabling SSL
for more than a small fraction
of Google connections would
overload the Google servers.
Google doesn't want to pay for
a bunch of extra computers.
Too slow $\Rightarrow$ unusable."

Why does Google actively
*turn off* cryptographic protection?

"Enabling SSL
for more than a small fraction
of Google connections would
overload the Google servers.
Google doesn't want to pay for
a bunch of extra computers.
Too slow $\Rightarrow$ unusable."

Many companies sell
SSL-acceleration hardware,
but that costs money too.

# Why are cryptographic computations so expensive?

Why are cryptographic
computations so expensive?

Can crypto be faster,
without being easy to break?

Why are cryptographic computations so expensive?

Can crypto be faster, without being easy to break?

Can crypto be fast enough to solidly protect all of Google's communications?

Why are cryptographic computations so expensive?

Can crypto be faster, without being easy to break?

Can crypto be fast enough to solidly protect all of Google's communications?

Can crypto be fast enough to protect every Internet packet?

Why are cryptographic computations so expensive?

Can crypto be faster, without being easy to break?

Can crypto be fast enough to solidly protect all of Google's communications?

Can crypto be fast enough to protect every Internet packet?

Can universal crypto be *usable*?

# What cryptography can do

Cryptography can
stop sniffing attackers
by scrambling legitimate packets.

Cryptography is often described
as protecting confidentiality:
attackers can't understand
the scrambled packets.

Can also protect integrity:
attackers can't figure out
a properly scrambled forgery.

Traditional cryptography requires each legitimate client-server pair to share a secret key.

Public-key cryptography has much lower requirements. (1976 Diffie–Hellman; many subsequent refinements)

Each party has one public key. Two parties can communicate securely if each party knows the other party's public key.

1993: IETF begins "DNSSEC" project to add public-key signatures to DNS.

## Paul Vixie, 1995.06:

This sounds simple but it has deep reaching consequences in both the protocol and the implementation—which is why it's taken more than a year to choose a security model and design a solution. We expect it to be another year before DNSSEC is in wide use on the leading edge, and at least a year after that before its use is commonplace on the Internet.

## BIND 8.2 blurb, 1999.03:

[Top feature:] Preliminary DNSSEC.

## BIND 9 blurb, 2000.09:

[Top feature:] DNSSEC.

# Paul Vixie, 2002.11:

We are still doing basic research on what kind of data model will work for DNS security. After three or four times of saying "NOW we've got it, THIS TIME for sure" there's finally some humility in the picture . . . "Wonder if THIS'll work?" . . .

It's impossible to know how many more flag days we'll have before it's safe to burn ROMs . . . It sure isn't plain old SIG+KEY, and it sure isn't DS as currently specified. When will it be? We don't know. . . .

2535 is already dead and buried. There is no installed base. We're starting from scratch.

Paul Vixie, 2004.04.20, announcing BIND 9.3 beta:

BIND 9.3 will ship with DNSSEC

# Paul Vixie, 2004.04.20, announcing BIND 9.3 beta:

BIND 9.3 will ship with DNSSEC support turned off by default in the configuration file.

# Paul Vixie, 2004.04.20, announcing BIND 9.3 beta:

> BIND 9.3 will ship with DNSSEC support turned off by default in the configuration file.
>
> . . .
>
> ISC will also begin offering direct support to users of BIND through the sale of annual support contracts.

# Paul Vixie, 2005.11.01:

> Had we done a requirements doc ten years ago . . . they might not have noticed that it would intersect their national privacy laws or business requirements, we might still have run into the NSEC3 juggernaut and be just as far off the rails now as we actually are now.

After fifteen years and millions of dollars of U.S. government grants (e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation), how successful is DNSSEC?

The Internet has about 78000000 `*.com` names.

After fifteen years and millions of dollars of U.S. government grants (e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation), how successful is DNSSEC?

The Internet has about 78000000 `*.com` names.

Surveys by DNSSEC developers, last updated 2009.02.28, have found 251 `*.com` names with DNSSEC signatures. 116 on 2008.08.20; $251 > 116$.

# Why is nobody using DNSSEC?

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

DNSSEC tries to minimize server-side costs by *precomputing* signatures of DNS records.

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 through
choice of crypto primitive.

DNSSEC RFCs
say DSA is "10 to 40 times as
slow for verification" as RSA;
recommend RSA "as the
preferred algorithm" for DNSSEC;
suggest RSA key size
of only 1024 bits
for "leaf nodes in the DNS."

I say:
1024-bit RSA is irresponsible.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets.

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

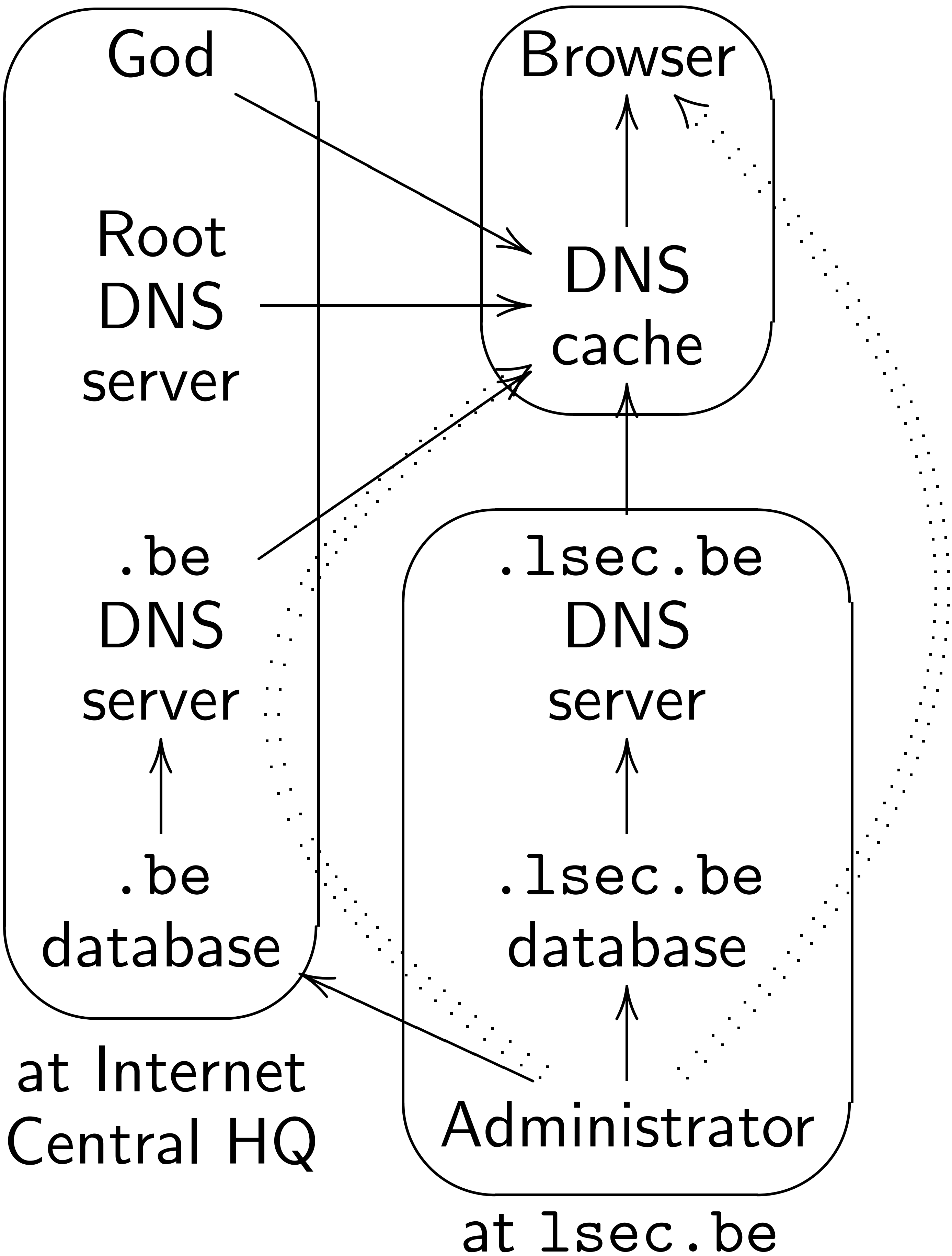
I say:
1024-bit RSA is irresponsible.

2003: Shamir–Tromer et al. concluded that 1024-bit RSA was already breakable by large companies and botnets.

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

But *most users don't know this*. Why aren't they using DNSSEC?

Recall the DNS architecture:



DNS server software listed in Wikipedia: BIND, Microsoft DNS, djbdns, Dnsmasq, Simple DNS Plus, NSD, PowerDNS, MaraDNS, ANS, Posadis, Secure64 DNS.

DNS database-management tools listed by 2008 Salomon: BPP, DNS Boss, DNStool, `gencidrzone`, `h2n`, `makezones`, NSC, `nsupdate`, SENDS, `updatehosts`, Utah Tools, `webdns`, `zsu`. Plus hundreds of homegrown tools written by DNS registrars etc.

DNSSEC requires 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 2GB database can produce 10GB database (2005 NIST study).
Tool reading database into RAM probably has to be reengineered.

Because of engineering costs and redeployment costs, very few database-management tools have added DNSSEC support.

Administrator has to manually mix existing management tools with separate signature generation for every change to DNS data.

Because of engineering costs and redeployment costs, very few database-management tools have added DNSSEC support.

Administrator has to manually mix existing management tools with separate signature generation for every change to DNS data.

2008 slideshow "DNSSEC in six minutes" (79 pages): "Any time you modify a zone … you must re-run `dnssec-signzone`."

Administrator also has to send
public key to `.be`.

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

Big zones such as `.com`
refuse to sign complete database.
Full DNSSEC signing would be
much too slow and much too big.

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

Often many more packets than original DNS.
Higher latency for user.
More frequent failures.

Also, much easier for attacker to deny service.
Official DNSSEC response, RFC 4033: "DNSSEC provides no protection against denial of service attacks."

Replay attack on DNSSEC:

Attacker inspects DNSSEC signatures from `lsec.be`.

`lsec.be` changes location, acquires new IP addresses, changes DNS records.

Replay attack on DNSSEC:

Attacker inspects DNSSEC signatures from `lsec.be`.

`lsec.be` changes location, acquires new IP addresses, changes DNS records.

Attacker buys the old addresses, forges DNS responses with the *old* DNS records and the *old* signatures. Successfully steals mail!

DNSSEC has a partial defense.
Signature has an expiration date,
normally signing date + 30 days.

Not very good security:
replay attack continues to work
for up to 30 days!

DNSSEC has a partial defense. Signature has an expiration date, normally signing date + 30 days.

Not very good security: replay attack continues to work for up to 30 days!

Also a major administrative hassle: administrator must generate new signatures before old signatures expire.

If administrator forgets, domain is destroyed. "DNSSEC suicide."

Imagine an “HTTPSEC”
that works like DNSSEC.

Imagine an “HTTPSEC”
that works like DNSSEC.

Installing HTTPSEC software
and setting up a public key
is just the beginning.

After every change to web pages,
have to run `httpsec-signpages`
to precompute new signatures.

Imagine an “HTTPSEC” that works like DNSSEC.

Installing HTTPSEC software and setting up a public key is just the beginning.

After every change to web pages, have to run `httpsec-signpages` to precompute new signatures.

Replay attacks work for 30 days.

Have to run `httpsec-signpages` before 30-day expiration or your web pages are destroyed.

But wait, there's more!

NXDOMAIN attack on DNSSEC: Attacker forges DNS response from `google.com` saying that `citronella.google.com` doesn't exist.

But wait, there's more!

NXDOMAIN attack on DNSSEC: Attacker forges DNS response from `google.com` saying that `citronella.google.com` doesn't exist.

Cache can't accept this without a signature: otherwise attacker can knock names off the Internet.

But wait, there's more!

NXDOMAIN attack on DNSSEC: Attacker forges DNS response from `google.com` saying that `citronella.google.com` doesn't exist.

Cache can't accept this without a signature: otherwise attacker can knock names off the Internet.

When is the signature precomputed? Does Google precompute signatures for *all* possible names? Too many!

DNSSEC solution: Sign multi-NXDOMAIN such as "there are no names between `chrome.google.com` and `code.google.com`."

DNSSEC server issues this signed data in response to any name between `chrome` and `code`. Implementing "between" is tricky but theoretically possible.

DNSSEC solution: Sign multi-NXDOMAIN such as "there are no names between `chrome.google.com` and `code.google.com`."

DNSSEC server issues this signed data in response to any name between `chrome` and `code`. Implementing "between" is tricky but theoretically possible.

Consequence: If you deploy DNSSEC then you are exposing *all* of your DNS names!

Newest DNSSEC variant:
"NSEC3" (2008 Laurie),
exposing *hashes* of DNS names.
Hash is 150 SHA-1 iterations.

Hash-enumeration attack:
Attacker guesses many names,
computes their hashes,
compares to the hashes
exposed by DNSSEC+NSEC3.

Small 10-computer cluster:
$\approx 2^{44}$ guesses/year.

Large company or botnet:
$\approx 2^{64}$ guesses/year.

Without DNSSEC,
attacker has to send query
for each guessed name.

Flooding a 4Mbps connection:
$\approx 2^{37}$ guesses/year.

Compared to normal DNS,
DNSSEC+NSEC3
makes guessing *silent* and
makes it *millions of times faster*
for a well-equipped attacker.

DNSSEC+NSEC3 is advertised
as being better than DNSSEC;
but it still loses privacy
compared to normal DNS.

Precomputation impact summary:

DNSSEC is pain for implementors. Hundreds of DNS programs— all caches, all servers, and all management tools— need to be modified to precompute and store signatures.

DNSSEC is pain for administrators, far beyond a simple upgrade.

DNSSEC hurts privacy.
DNSSEC hurts reliability.
DNSSEC aids denial of service.

# Rethinking signatures

Conventional wisdom:
DNSSEC's precomputation,
sacrificing security while
creating severe usability problems,
is necessary for speed.

Can we achieve adequate speed
*without* precomputation?
Let's change the design.

# Rethinking signatures

Conventional wisdom:
DNSSEC's precomputation,
sacrificing security while
creating severe usability problems,
is necessary for speed.

Can we achieve adequate speed
*without* precomputation?
Let's change the design.

**1. Add encryption.**
Want to protect against sabotage
*and* against espionage.
So use public-key signatures
*and* public-key encryption.

**2. Merge signing with encryption.**

"Public-key signcryption" protects against forgery and eavesdropping in one step.

"Public-key authenticated encryption" is even faster.

No need to
partition the algorithms into
an encryption component and
an authentication component.
Combined algorithms are faster.

## 3. Merge public-key operations across multiple messages.

It's silly for a sender
to authcrypt two messages
to the same recipient.

"Hybrid cryptography"
is much faster.

Example: Sender
generates a random AES key,
authcrypts the AES key,
uses the AES key to encrypt and
authenticate both messages.

## 4. Choose sensible primitives.

256-bit elliptic-curve cryptography using public-domain software:

489069 Core 2 cycles to handle a new communication partner.

5355 cycles to encrypt and authenticate a 510-byte message.

6786 cycles to verify and decrypt a legitimate 510-byte message.

3465 cycles to reject a forged 510-byte message.

A 2.5GHz Intel Core 2 Quad
Q9300 CPU costs $225.
Complete computer: $400.

This CPU has 4 cores.
Each core carries out
2.5 billion cycles/second.

On this computer,
this software takes just 49
seconds to handle 1000000 new
communication partners,
and just 12 seconds to handle
10000000 incoming packets *and*
10000000 outgoing packets.

VeriSign is spending $>$\$100000000 to upgrade the Internet's `.com` DNS servers.

In a typical day, these servers together handle 35 billion queries from 5 million clients.

VeriSign is spending
>$100000000 to upgrade the
Internet's `.com` DNS servers.

In a typical day, these servers together handle 35 billion queries from 5 million clients.

Total cryptographic cost:
about half a day on a single
$400 computer with this software.

VeriSign is spending $>$\$100000000 to upgrade the Internet's `.com` DNS servers.

In a typical day, these servers together handle 35 billion queries from 5 million clients.

Total cryptographic cost: about half a day on a single \$400 computer with this software.

Verisign says that it wants to be prepared for 4 trillion packets/day. Cryptographic cost of 4 trillion partners/day with this software: $< 3000$ computers.

This software is a new library “NaCl” (Networking and Cryptography library) developed within the EU FP7 “CACE” (Computer Aided Cryptography Engineering) project.

This software is a new library "NaCl" (Networking and Cryptography library) developed within the EU FP7 "CACE" (Computer Aided Cryptography Engineering) project.

News: All parts of NaCl needed for DNS are done!

This software is a new library "NaCl" (Networking and Cryptography library) developed within the EU FP7 "CACE" (Computer Aided Cryptography Engineering) project.

News: All parts of NaCl needed for DNS are done!

Actually done three months ago. Subsequent time has been QA. This month QA will finish and NaCl will be released.

This software is a new library “NaCl” (Networking and Cryptography library) developed within the EU FP7 “CACE” (Computer Aided Cryptography Engineering) project.

News: All parts of NaCl needed for DNS are done!

Actually done three months ago. Subsequent time has been QA. This month QA will finish and NaCl will be released.

Want to peek?
`http://nacl.cace-project.eu`

# The DNSCurve project

DNSCurve uses NaCl
to add heavy-duty integrity
(RSA-1024 has 80-bit security;
DNSCurve has 128-bit security)
*and* some confidentiality
*and* availability
to the Domain Name System.

Despite all this security,
DNSCurve is very easy for DNS
software authors to implement
and very easy for
administrators to deploy.

Administrator has to change
the `lsec.be` server
to support DNSCurve,
*or* install a DNSCurve forwarder
alongside the server.

Administrator does *not* need to
change database software,
does *not* need to store signatures,
does *not* need new procedures
for updating DNS records, and
does *not* risk DNSSEC suicide.

Administrator changes
server name such as `ns2`
to a server name that encodes
the DNSCurve public key.

The `.be` server
and database software
and web interface
already support
`lsec.be` server names selected
by the `lsec.be` administrator!

Cache has to be upgraded
to support DNSCurve.

Cache naturally sees the
encoded DNSCurve public key.
Cache encrypts and authenticates
DNS packets sent to that server.
Cache verifies and decrypts DNS
packets received from that server.

No extra packets.
Forged packets are
very efficiently discarded.
Denial of service becomes
much more difficult.

Kaminsky, 2008.12: "It's not actually that fast. . . . DNSCurve does a crypto operation per query. With DJB's sample code, a laptop that can do 15,000 DNS queries a second can do maybe 10,000 ECC operations per second. With 1 operation inbound and 1 operation outbound, that's 100% CPU on 1/3 the traffic *before you've parsed a single DNS packet*"

Kaminsky, 2008.12: "It's not actually that fast. . . . DNSCurve does a crypto operation per query. With DJB's sample code, a laptop that can do 15,000 DNS queries a second can do maybe 10,000 ECC operations per second. With 1 operation inbound and 1 operation outbound, that's 100% CPU on 1/3 the traffic *before you've parsed a single DNS packet*"

Wrong. There is only one ECC operation *per communication partner*, not per packet.

Kaminsky, 2008.12:
End-to-end trust with DNSCurve means “abandoning caching . . . $100\times$ increase in load.”

Kaminsky, 2008.12:
End-to-end trust with DNSCurve means "abandoning caching . . . $100\times$ increase in load."

Wrong. Users *can* and *should* run caches on their own computers.

Security advantage: User running a DNSCurve cache is protected from intermediate ISP modifying the administrator's DNS data.

Speed advantage: Per-user caches are actually *more* effective than a centralized ISP cache, *decreasing* DNS load.

There *is* something
that DNSSEC accomplishes
and that DNSCurve doesn't:

DNSSEC can protect against
compromise of DNS servers
if administrator generates
signatures on another machine
that has not been compromised.

There *is* something
that DNSSEC accomplishes
and that DNSCurve doesn't:

DNSSEC can protect against
compromise of DNS servers
if administrator generates
signatures on another machine
that has not been compromised.

Analogy: HTTPSEC can protect
against compromise of HTTP
servers if administrator signs
web pages on another machine.
But does this justify the pain of
DNSSEC+HTTPSEC?

More information on DNSCurve:

See `http://dnscurve.org`.

More information on DNSCurve:
See `http://dnscurve.org`.

Thinking beyond DNS:
Can every Internet packet
be protected in a similar way?

More information on DNSCurve:
See `http://dnscurve.org`.

Thinking beyond DNS:
Can every Internet packet
be protected in a similar way?

Thinking beyond networking:
When people sacrifice security
and usability for the sake of
performance, are they really
improving performance?