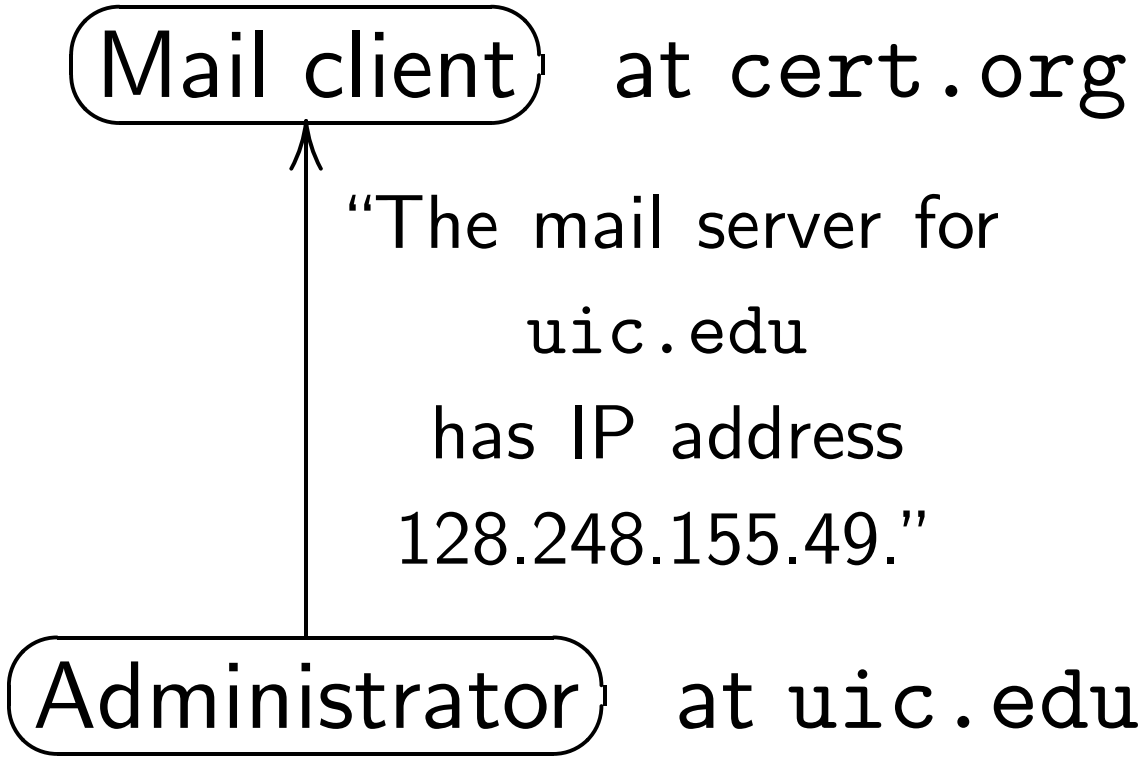
DNSCurve:
Usable security for DNS

D. J. Bernstein
Research Professor
Center for RITES: Research and
Instruction in Technologies
for Electronic Security
Department of Computer Science
University of Illinois at Chicago

The Domain Name System

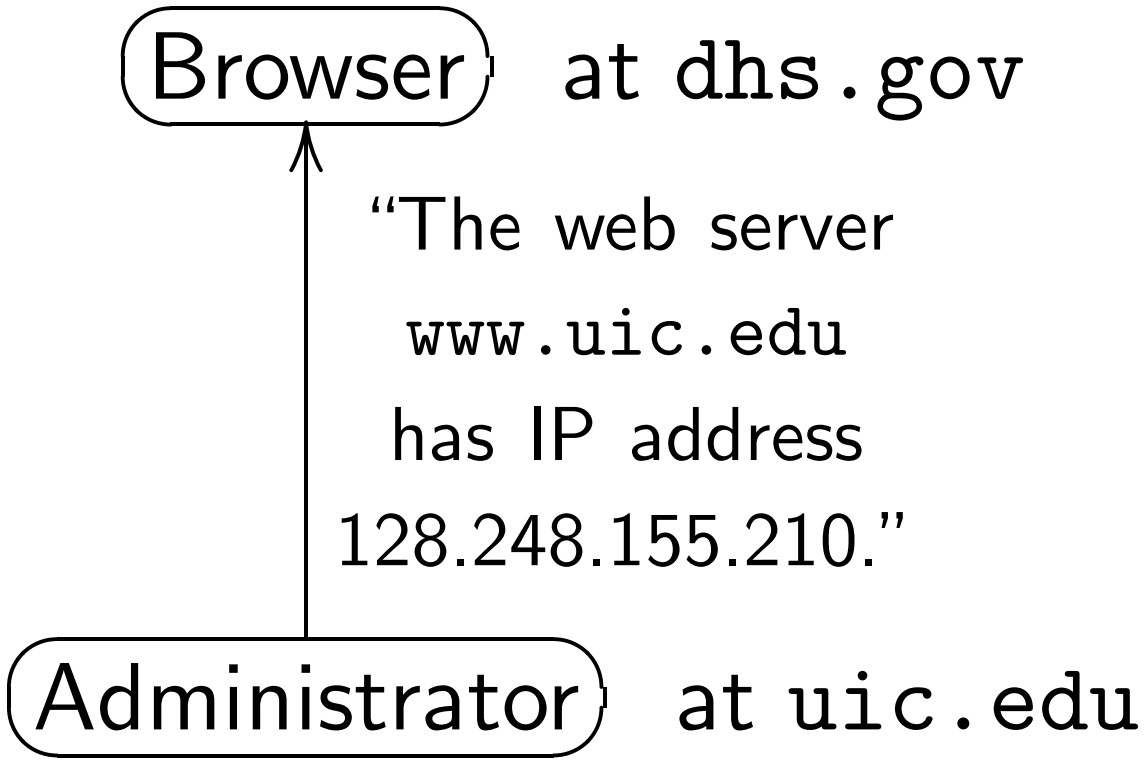
`cert.org` has mail
to deliver to `someone@uic.edu`.



Now `cert.org`
delivers mail to
IP address 128.248.155.49.

Same for web browsing.

`dhs.gov` wants to see
`http://www.uic.edu`.



Now `dhs.gov`
retrieves web page from
IP address 128.248.155.210.

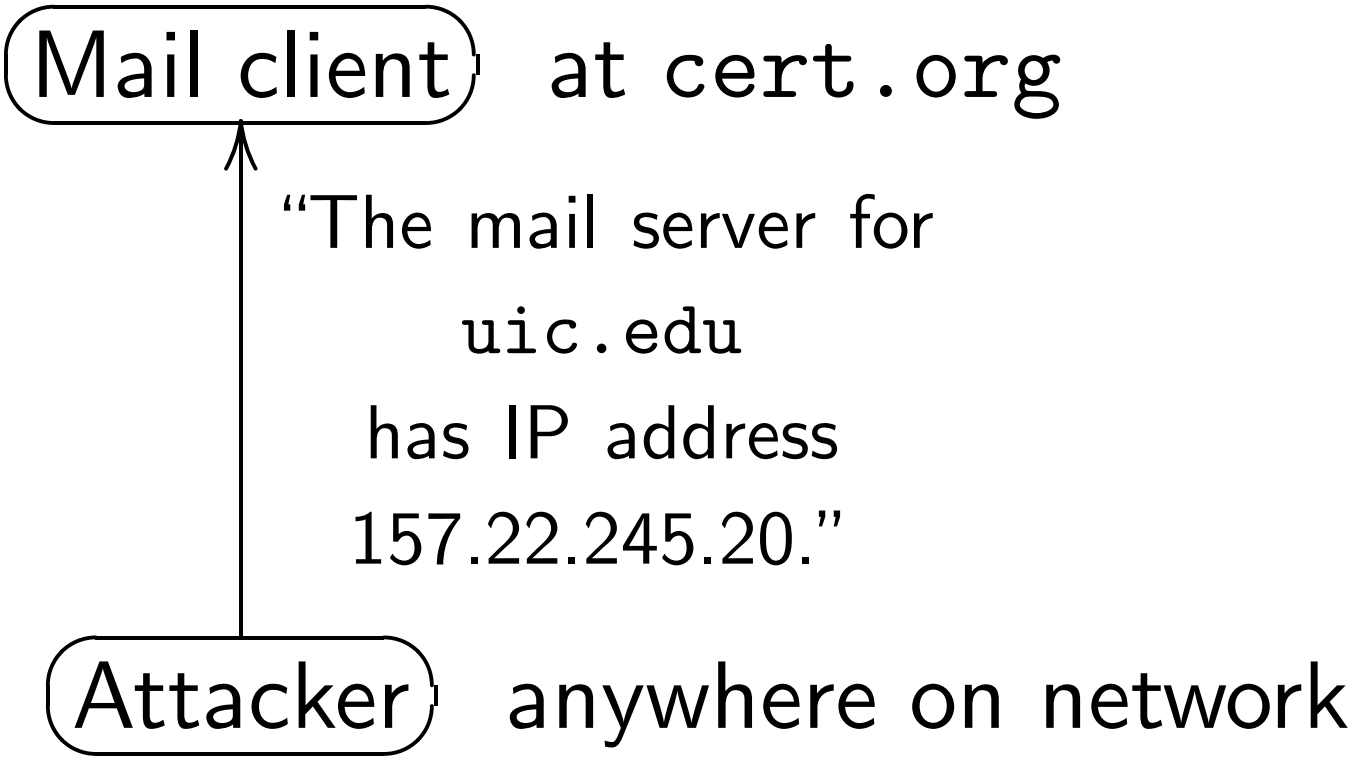
DNS software security holes:
BIND libresolv buffer overflow,
Microsoft cache promiscuity,
BIND 8 TSIG buffer overflow,
BIND 9 dig promiscuity, etc.

Fix: Use better DNS software.
`http://cr.yp.to/djbdns.html`
carries a $1000 reward for
first verifiable public report
of a software security hole.

But what about *protocol* holes?

Forging DNS packets

`cert.org` has mail
to deliver to `someone@uic.edu`.



Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

“Can attackers do that?”

“Can attackers do that?”

— Yes.

“Can attackers do that?”

— Yes.

“Really?”

“Can attackers do that?”

— Yes.

“Really?” — Yes.

“Can attackers do that?”
— Yes.

“Really?” — Yes.

“Don’t the clients check
who’s sending information?”

“Can attackers do that?”
— Yes.

“Really?” — Yes.

“Don’t the clients check
who’s sending information?”
— Yes, but the attacker
forges the sender address;
as easy as forging address
on a physically mailed letter.

"Is the client always listening for the address of `www.uic.edu`?"

“Is the client always
listening for the address of
`www.uic.edu`?”

— No, but many ways for
attackers to work around this:

1. Attack repeatedly.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:

...

"Is the client always
listening for the address of
`www.uic.edu`?"

— No, but many ways for
attackers to work around this:

1. Attack repeatedly.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `uic.edu`?"

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `uic.edu`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.
4. Easy, succeeds instantly:

...

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `uic.edu`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.
4. Easy, succeeds instantly: Sniff the network.

What about cookies?

Client's DNS query packet contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence:
1, 2, 3, 4, 5, etc.

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Many “random” IDs
are actually quite easy to predict.
Client asks for information
from attacker’s servers;
attacker inspects IDs,
predicts subsequent IDs.

See, e.g., emergency BIND 9
upgrade (2007.07.24) responding
to attack by Amit Klein.

But modern cryptographic
random-number generators
are extremely difficult to predict.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised
in `djbdns` since 1999,
and in PowerDNS since 2006.

Same feature added 2008.07
in emergency upgrade to BIND,
Microsoft DNS, Nominum CNS,
most Cisco products, etc.

Many ways for attackers
to beat this randomization,
even if it's cryptographic:

1. Attack repeatedly.
"An attacker who makes a few
billion random guesses is likely
to succeed at least once; tens of
millions of guesses are adequate
with a colliding attack;" etc.

2. Allocate most UDP ports
to other tasks, non-reusably.

3. Easy, succeeds instantly:

...

Many ways for attackers
to beat this randomization,
even if it's cryptographic:

1. Attack repeatedly.
"An attacker who makes a few
billion random guesses is likely
to succeed at least once; tens of
millions of guesses are adequate
with a colliding attack;" etc.

2. Allocate most UDP ports
to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

What about thorough crypto?

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, June 1995:

> 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, March 1999:

> [Top feature:] Preliminary DNSSEC.

BIND 9 blurb, September 2000:

> [Top feature:] DNSSEC.

Paul Vixie, November 2002:

> 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, 20 April 2004, announcing BIND 9.3 beta:

> BIND 9.3 will ship with DNSSEC

Paul Vixie, 20 April 2004, announcing BIND 9.3 beta:

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

Paul Vixie, 20 April 2004, 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, 1 November 2005:

> 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 government grants (e.g., DISA to BIND company; NSF to UCLA; DHS to Secure64 Software Corporation), how successful is DNSSEC?

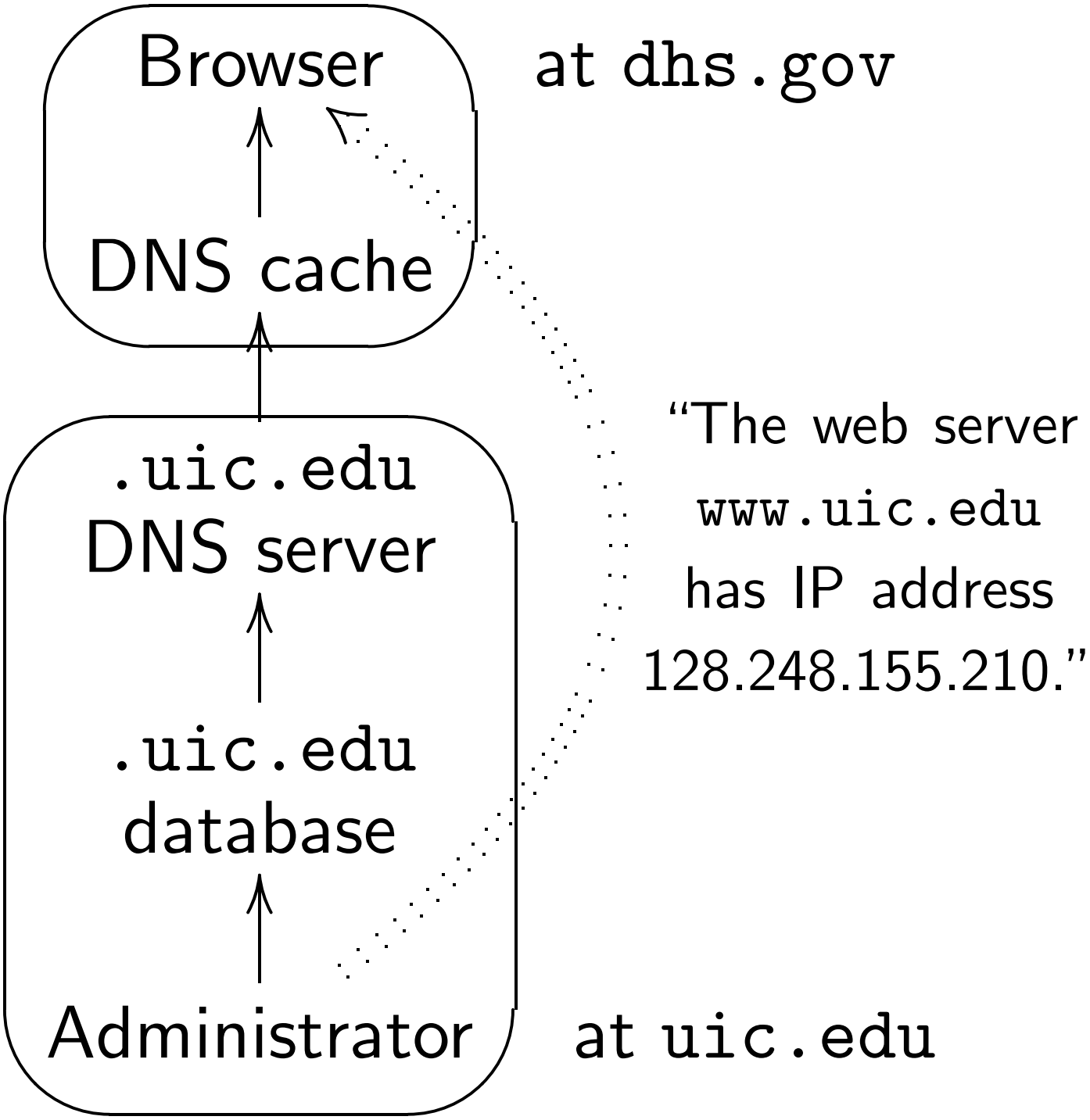
The Internet has about 70000000 `*.com` names.

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

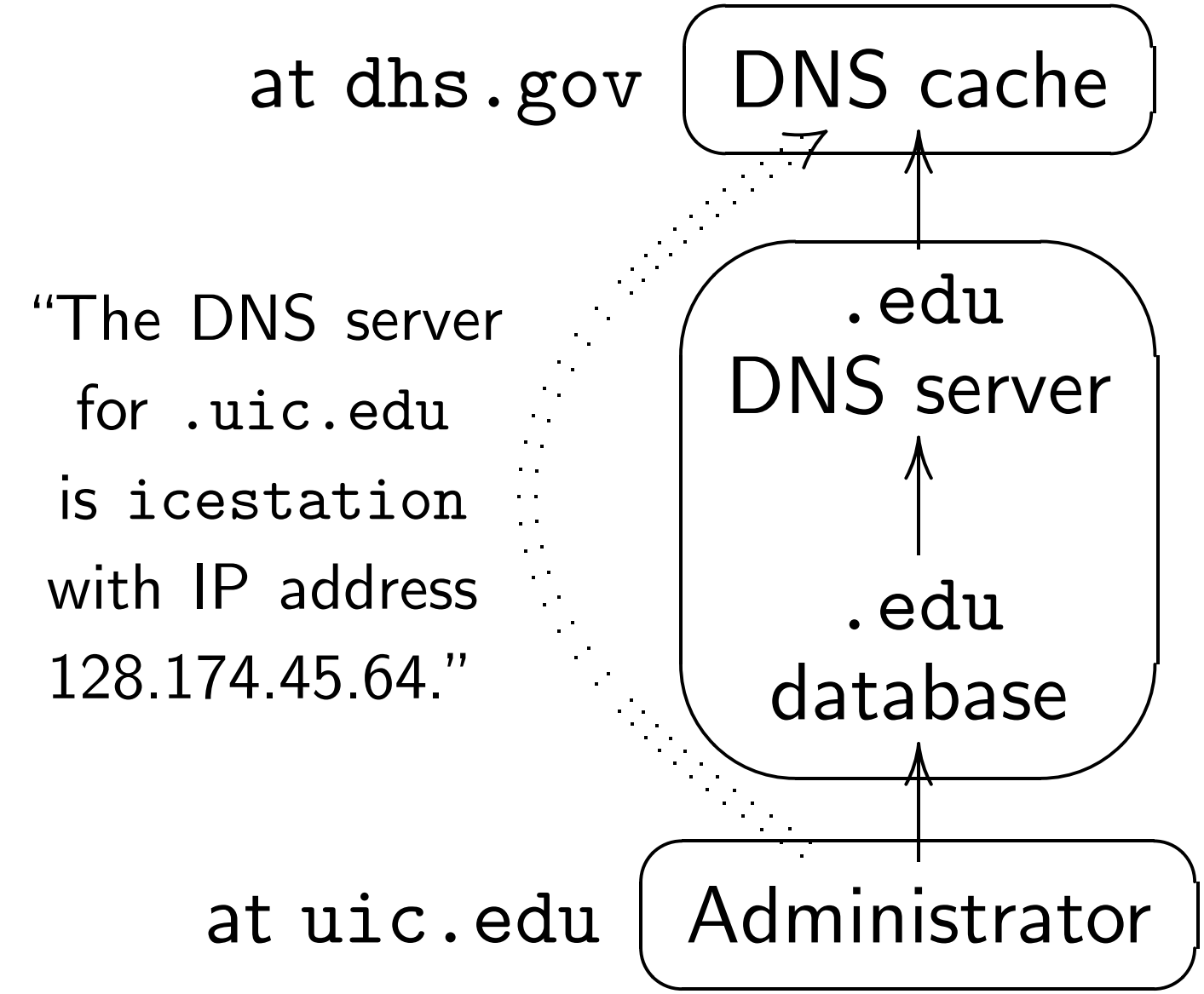
The Internet has about 70000000 `*.com` names.

Surveys by DNSSEC developers, last updated 2008.08.20, have found 116 `*.com` names with DNSSEC signatures.

DNS in more detail



DNS cache learns location of `.uic.edu` DNS server from `.edu` DNS server:



Packets to/from DNS cache

God sayeth unto the DNS cache:

"DNS Root `K.Heaven` 193.0.14.129"

"Web `www.uic.edu`?"

193.0.14.129 ⟵ ⟶ DNS cache

"DNS `.edu a3` 192.5.6.32"

"Web `www.uic.edu`?"

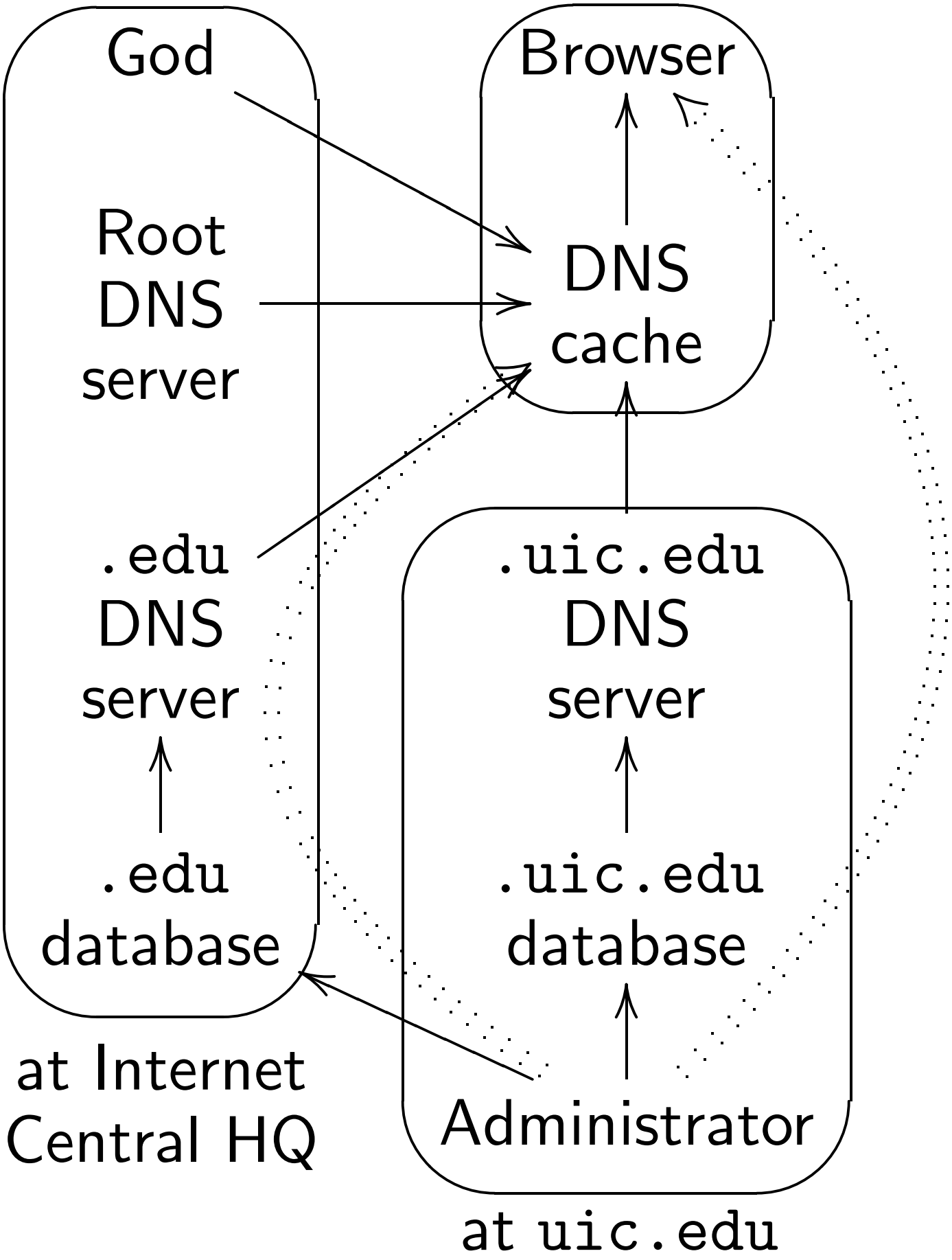
192.5.6.32 ⟵ ⟶ DNS cache

"DNS `.uic.edu icestation` 128.174.45.64"

"Web `www.uic.edu`?"

128.174.45.64 ⟵ ⟶ DNS cache

"Web `www.uic.edu` 128.248.155.210"


God
Root
DNS
server
.edu
DNS
server
.edu
database
at Internet
Central HQ
Browser
DNS
cache
.uic.edu
DNS
server
.uic.edu
database
Administrator
at uic.edu

The impact of DNSSEC

DNSSEC changes everything.

Administrator doesn't just
put data into database.
Must sign each update.
Sign again every month or
`uic.edu` drops off Internet;
"DNSSEC suicide."

Administrator has to change
the `uic.edu` server
and the database software
(maybe a homegrown Perl script?)
to support signed records.

Administrator has to send
public key to `edu`.

The `.edu` 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.”

Most DNS administrators
have disabled “zone transfers.”

DNSSEC subverts this.
DNSSEC publishes all DNS data.

“NSEC3” DNSSEC variant
tries to limit exposure
but is almost always
very easy to break.
Attacker can extract
$>$ 99% of all DNS data.

Official DNSSEC response:
“DNSSEC is not designed
to provide confidentiality.”

Standard DNSSEC signatures:
1024-bit RSA (with SHA-1).

Popular but shortsighted.
Breakable today by botnets
and by large companies.

Allegedly chosen for
fastest possible verification.
DNSSEC software uses
a few million CPU cycles
to verify a DNS record.

Larger RSA key sizes
create even worse problems
for DNSSEC time and bandwidth.

DNSCurve

DNSCurve is a new project
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
software authors to implement
and very easy for
administrators to deploy.

Administrator has to change
the `uic.edu` 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 `icestation`
to a server name that encodes
the DNSCurve public key.

The `.edu` server
and database software
and web interface
already support
`uic.edu` server names selected
by the `uic.edu` 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.

DNSCurve makes critical use
of state-of-the-art
elliptic-curve cryptography.

Cache has private integer c,
32-byte public key Curve25519(c).
Server has private integer s,
32-byte public key Curve25519(s).

Cache and server compute
shared secret Curve25519(sc),
use that secret to quickly
encrypt and authenticate packets.

Whenever cache sees a
new public key Curve25519(s)
it computes Curve25519(sc).

Core 2 Quad Q6600 can do this
25000 times per second
using latest Curve25519 software.

Per-packet costs: much smaller;
pure secret-key cryptography.

Similar speed for server.
Many new clients per second;
many more packets per second.

Plans

All necessary cryptographic tools are in new "Networking and Cryptography library" (NaCl) co-developed with EU FP7 CACE project.

Top two tasks:

1. DNSCurve forwarder suitable for use with any DNS server.

2. DNSCurve cache, built with minimal modifications to `dnscache` from `djbdns`.