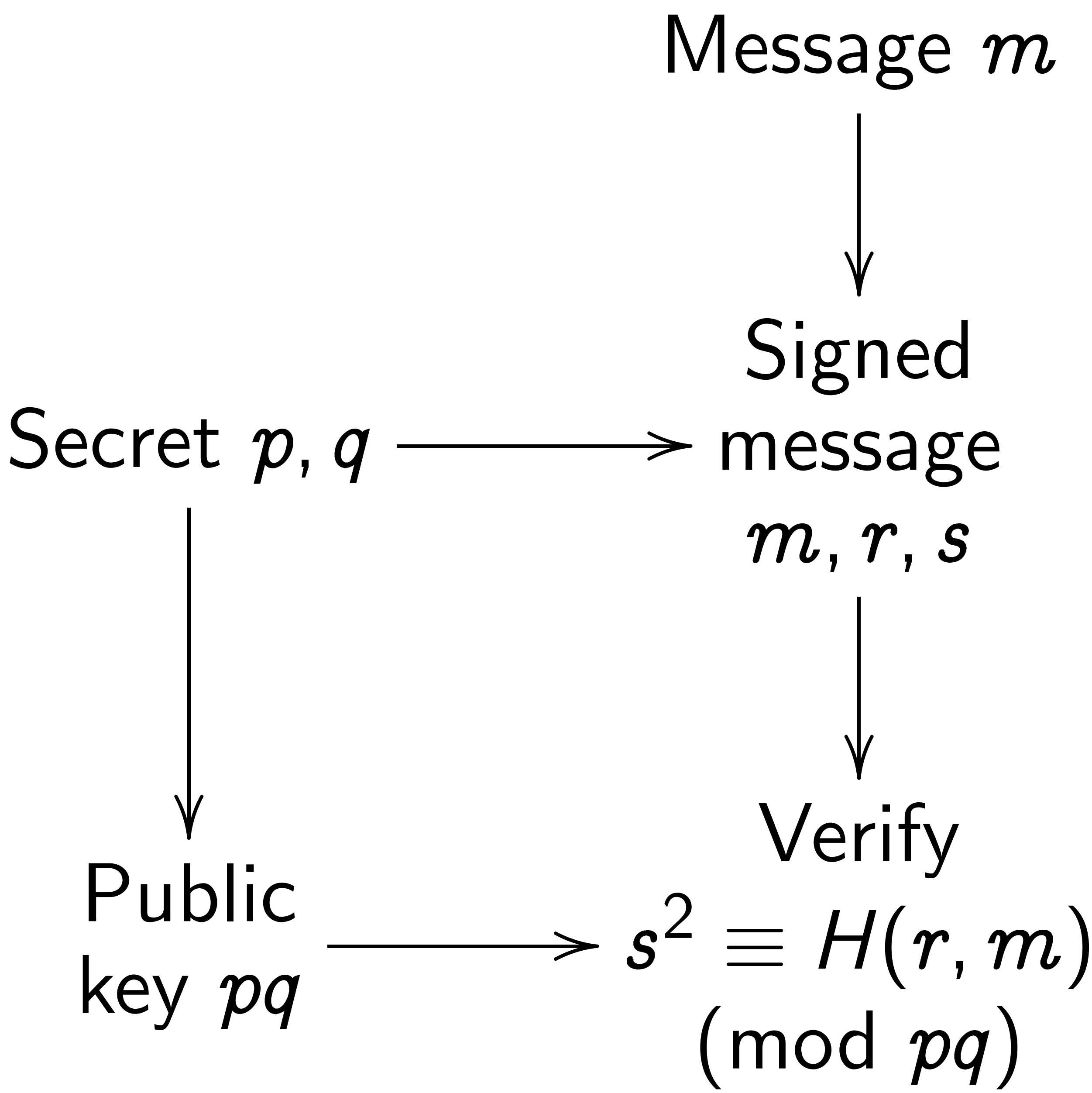
# The DNS security mess

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation
Math Sciences Research Institute
University of California at Berkeley

# Rabin's public-key signature system



# The Internet

Web-browsing procedure:

1. Figure out web page's URL.
2. Figure out server's IP address.
3. Figure out server's public key.
4. Retrieve page.

Similar procedure for mail et al.

Need to protect each step
against forgery.
(And against denial of service.)

Assuming URL is protected:

Why not put IP address into URL?
Protects IP address for free.

Answer: IP addresses often change.
Want old links to keep working.

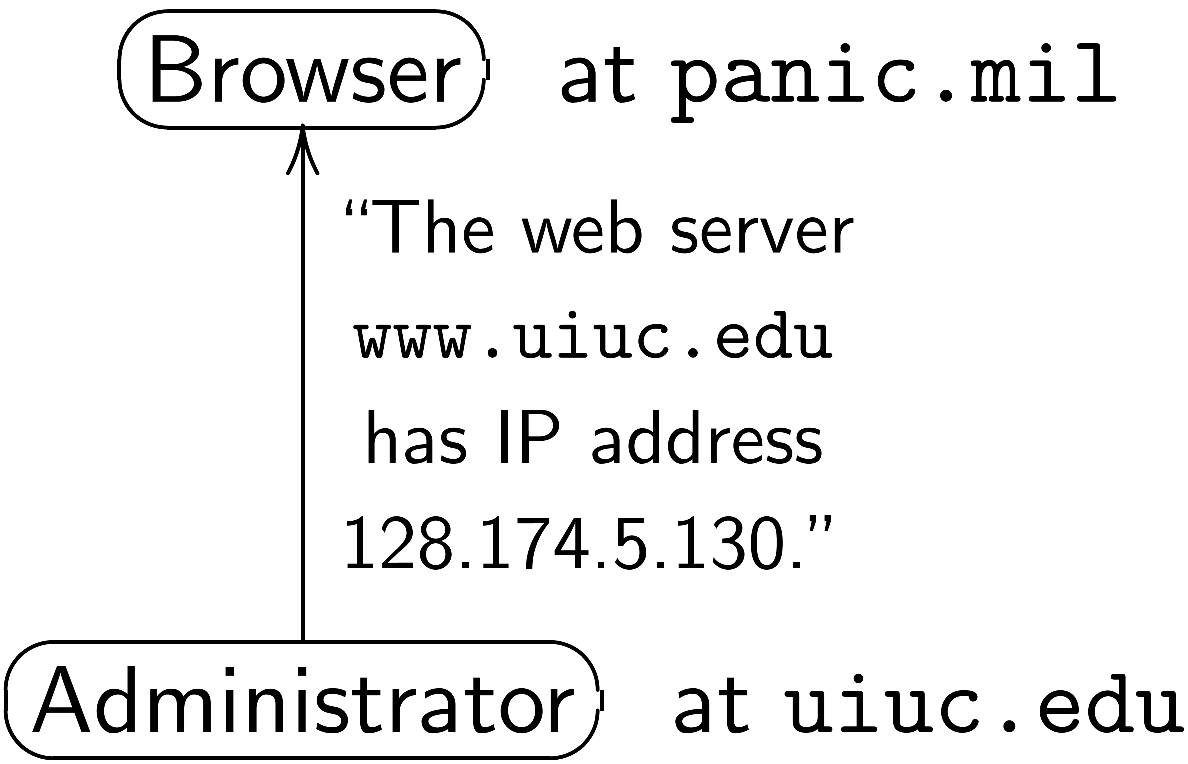
Why not put public key into URL?
Protects public key for free.

Will come back to this.

This talk focuses on step 2:
given web-page URL,
find server's IP address.

e.g. if URL is
`http://www.uiuc.edu/Library/`
then need to find IP address
of `www.uiuc.edu`.

# The Domain Name System



Many 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.

But what about protocol holes?

Attacker can forge DNS packets.

Blind attacker must guess cookie;
32 bits in best current software.
Could make cookie larger by
extending or abusing protocol.

Sniffing attacker succeeds easily,
no matter how big cookie is.

Solution: public-key signatures.

## 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.

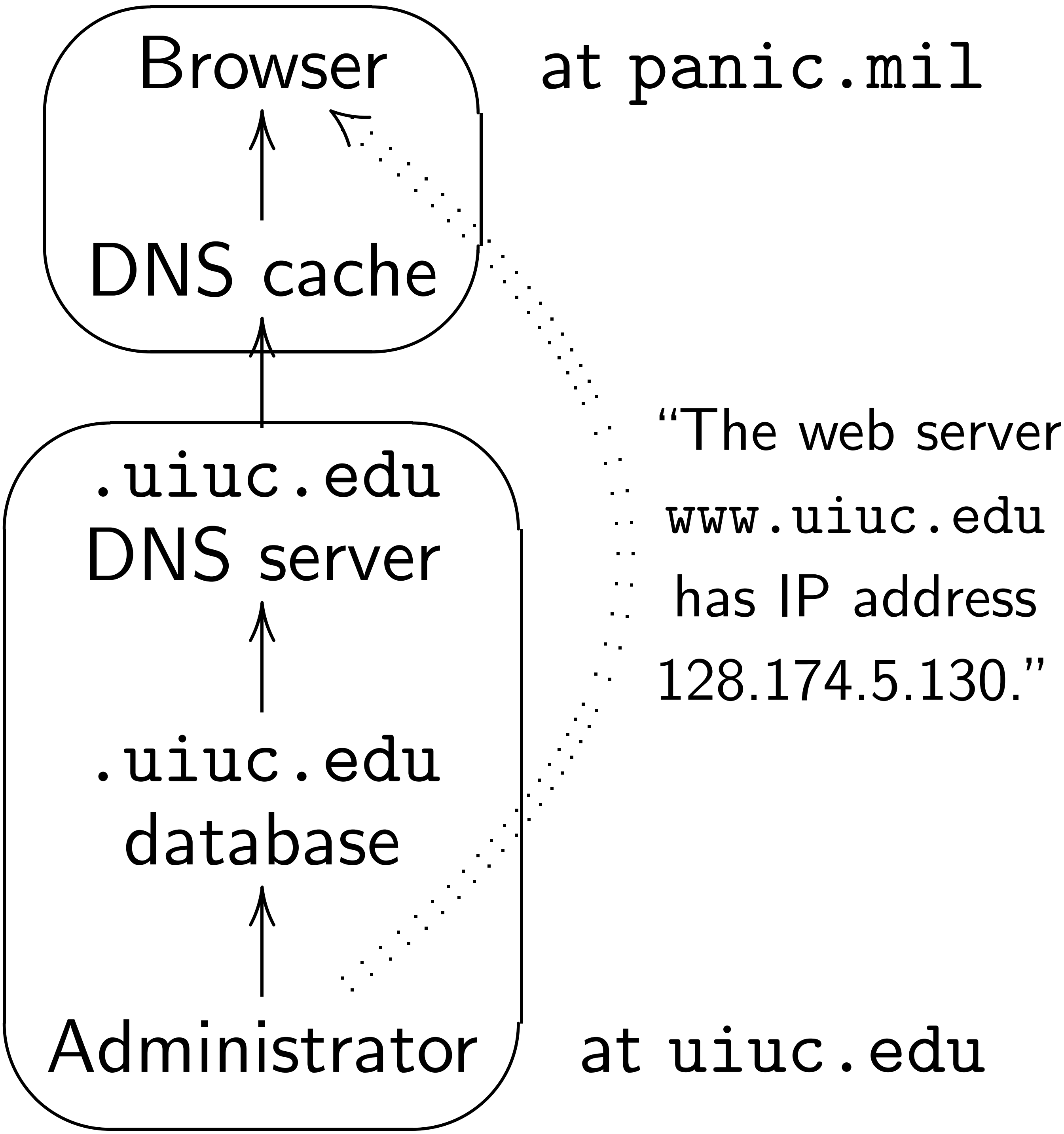
Paul Vixie, 20 April 2004,
announcing BIND 9.3 beta:

BIND 9.3 will ship with DNSSEC

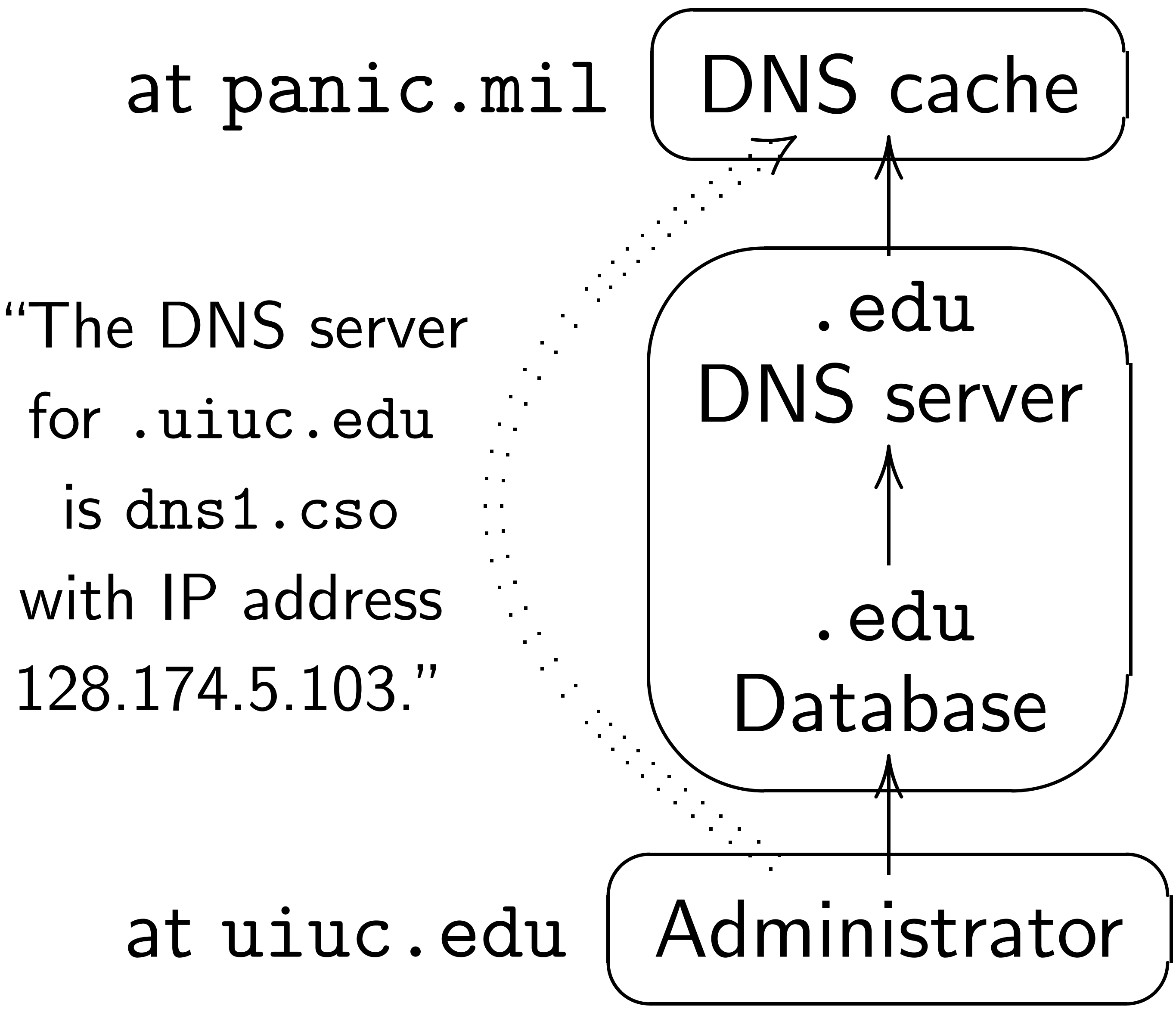
# Paul Vixie, 20 April 2004, announcing BIND 9.3 beta:

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

# DNS in more detail



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



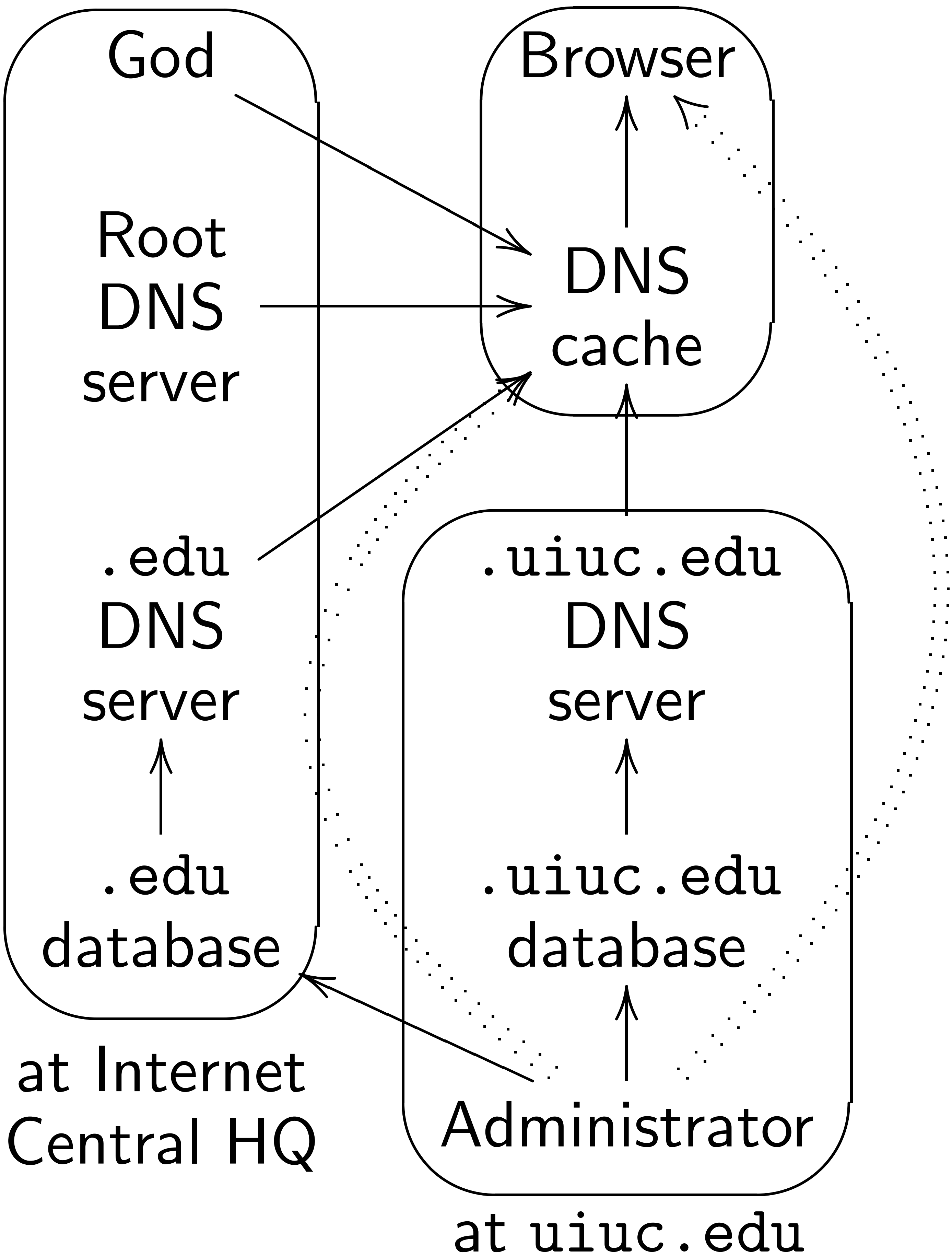
# Packets to/from DNS cache

God sayeth unto the DNS cache:
"DNS Root `K.Heaven` 193.0.14.129"

"Web `www.uiuc.edu?`"
193.0.14.129 ← → DNS cache
"DNS `.edu a3` 192.5.6.32"

"Web `www.uiuc.edu?`"
192.5.6.32 ← → DNS cache
"DNS `.uiuc.edu dns1.cso` 128.174.5.103"

"Web `www.uiuc.edu?`"
128.174.5.103 ← → DNS cache
"Web `www.uiuc.edu` 128.174.5.130"


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

# Making DNS secure

Many popular ways to authenticate
cache $\to$ browser: e.g., IPSEC, or
put cache on same box as browser.
Same for other local communication.

Limited risk for God $\to$ cache:
data set is small, stable, widespread.
Keep safe local copy of result.
Can also keep copies of
data from root server.

Many popular ways to authenticate Urbana admin $\rightarrow$ `.edu`: e.g., SSL-encrypted passwords.
Be careful: In January 2001, someone fooled Internet HQ into accepting fake Microsoft data.

Want to use public-key signatures for `.edu` server $\rightarrow$ cache and `.uiuc.edu` server $\rightarrow$ cache.

# Who should check signatures?

Caches have responsibility
for verifying signatures.

Could check in browser instead,
but caches are easier than browsers
to upgrade and redeploy.

(Also, without cache support,
can't stop denial of service.)

## How does the cache obtain keys?

Urbana administrator signs `www.uiuc.edu` information under `.uiuc.edu` public key. Cache needs safe copy of that key.

Old DNSSEC approach: `.uiuc.edu` server sends its key, signed by `.edu` key, to the cache.

Current DNSSEC approach:
`.edu` server sends
second Urbana key to cache,
signed by `.edu` key;
`.uiuc.edu` server sends
first Urbana key to cache,
signed by second key.

New software for DNS servers,
`.edu` database to store keys,
and `.uiuc.edu` database.

No reason to change software!

`.edu` server has to sign
"`.uiuc.edu dns1.cso` 128.174.5.103"
anyway. Embed Urbana key $k$
into `dns1.cso` field as $k.m_1$
where $m_1$ is a magic number.

Cache sees $m_1$, extracts $k$,
rejects data not signed by $k$.

Another solution:
Put public keys into URLs.
Use `www.`$k$.$m_2$`.uiuc.edu`
instead of `www.uiuc.edu`.

Cache sees $m_2$, extracts $k$,
rejects data not signed by $k$.

Doesn't need HQ cooperation.
In fact, secure against HQ!
(But HQ can still deny service.)

# How does cache obtain signatures?

How are signatures encoded
in DNS responses?

DNSSEC: Servers are responsible
for volunteering signatures
in a new signature format.
(Sometimes cache has to
go track down signatures;
makes denial of service easier.)

New software for DNS servers.

No reason to change software!
Put signed data into existing servers.

Cache wants `ab.cd.uiuc.edu`
data from `.uiuc.edu`
with signature under key $k$.
Instead requests data for
$r.m_3$`.ab.cd.`$k.m_3$`.uiuc.edu`
where $r$ is a cookie.
Rejects unsigned results.

(Cookie stops blind attacks.)

Simplified example in BIND format:

.uiuc.edu server has

```
*.123.www.8675309.123.uiuc.edu.
TXT "A 128.174.5.130 ..."
```

where ... is a signature of

```
www A 128.174.5.103
```

under Urbana's key 8675309.

.edu server has

```
*.uiuc.3141592.123.edu.
TXT "uiuc NS
8675309.789 128.174.5.103 ...".
```

Cache wants data for
`www.uiuc.edu` or
`www.8675309.456.uiuc.edu`.

Asks `.edu` server about

`237.123.www.uiuc`

`.3141592.123.edu`.

Checks signature under key 3141592.

Asks `.uiuc.edu` server about

`291.123.www.`

`.8675309.123.uiuc.edu`.

Checks signature under key 8675309.

## Packet space limitations

DNS packets over UDP are limited to 512 bytes.
DNS packets over TCP are much more costly. (And allow much easier denial of service.)

DNSSEC uses RSA keys too small for comfort, and changes servers to use larger packets; still has to fall back to TCP frequently.

Could use signature systems
with slower verification,
but that could overload caches
(and help denial of service).

Better: Compress Rabin keys
to 1/3 size; Coppersmith.
Then replace keys with hashes.
Compress signatures
to 1/2 size; Bleichenbacher.

Helps dramatically to use
only one signature per packet.

# Precomputation hassles

Many DNS servers receive several thousand queries per second. Can't keep up without precomputing some signatures.

To avoid changing server (and to prevent denial of service), need to precompute all signatures.

Can't use client's fresh cookie in precomputation, so need secure global clocks for freshness.

Can't precompute signatures for all possible responses: .uiuc.edu controls quizno357.uiuc.edu etc.

DNSSEC approach: Sign wildcards such as "there are no names between quaalude.uiuc.edu and quizzical.uiuc.edu." Saves time for snoops.

Better: Skip it. Users don't care. Handle SRV silliness separately.

The `.com` database is $\approx$ 2GB.
With signatures, several times larger; won't fit into memory.
(VM allows easy denial of service.)

DNSSEC approach: "opt-in."
Useless signatures such as "This is a signature for any data you might receive for `x.com` through `y.com`."

Better: Buy enough memory!

# What's next?

Next release of `dnscache` checks 1536-bit signatures, using mechanisms $m_1, m_2, m_3$.

`dnssec2` tool creates public key and precomputes signatures.

`floatasm` lower-level tools: new programming language for straight-line floating-point code.

Planned: `dnsforge` tool.