The DNS security mess

D. J. Bernstein
University of Illinois at Chicago

A public-key signature system

$$\begin{tikzcd}
 & \text{Message } m \arrow[d] \\
\begin{array}{c}\text{Signer's}\\ \text{secret}\\ \text{key } n\end{array} \arrow[r] \arrow[d] & \begin{array}{c}\text{Signed}\\ \text{message}\\ m, r, s\end{array} \arrow[d] \\
\begin{array}{c}\text{Signer's}\\ \text{public}\\ \text{key } nB\end{array} \arrow[r] & \begin{array}{c}\text{Verify}\\ r = \text{SHA-256}\\ (sB - rnB, m)\end{array}
\end{tikzcd}$$

Signer can compute signature.

Anyone can verify signature.

Seems hard for attacker
to forge signature.

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.akamai.com/`
`html/support`

then need to find IP address of `www.akamai.com`.

The Domain Name System

Browser at `panic.gov`

↑

“The web server
`www.akamai.com`
has IP address
64.212.198.121.”

Administrator at `akamai.com`

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.
`http://cr.yp.to/djbdns.html`

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.

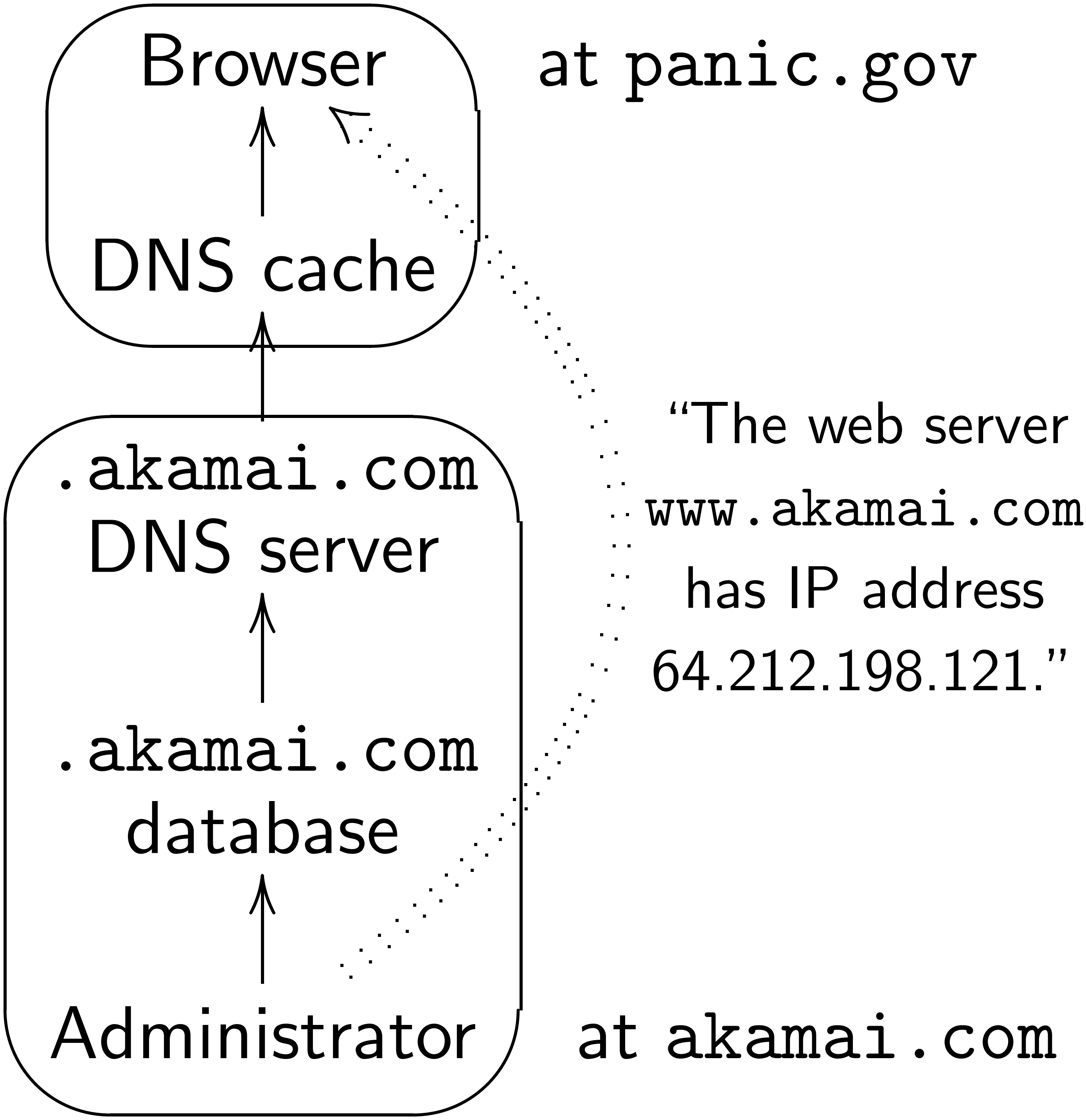
Paul Vixie, 20 April 2004, announcing BIND 9.3 beta:

> 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.

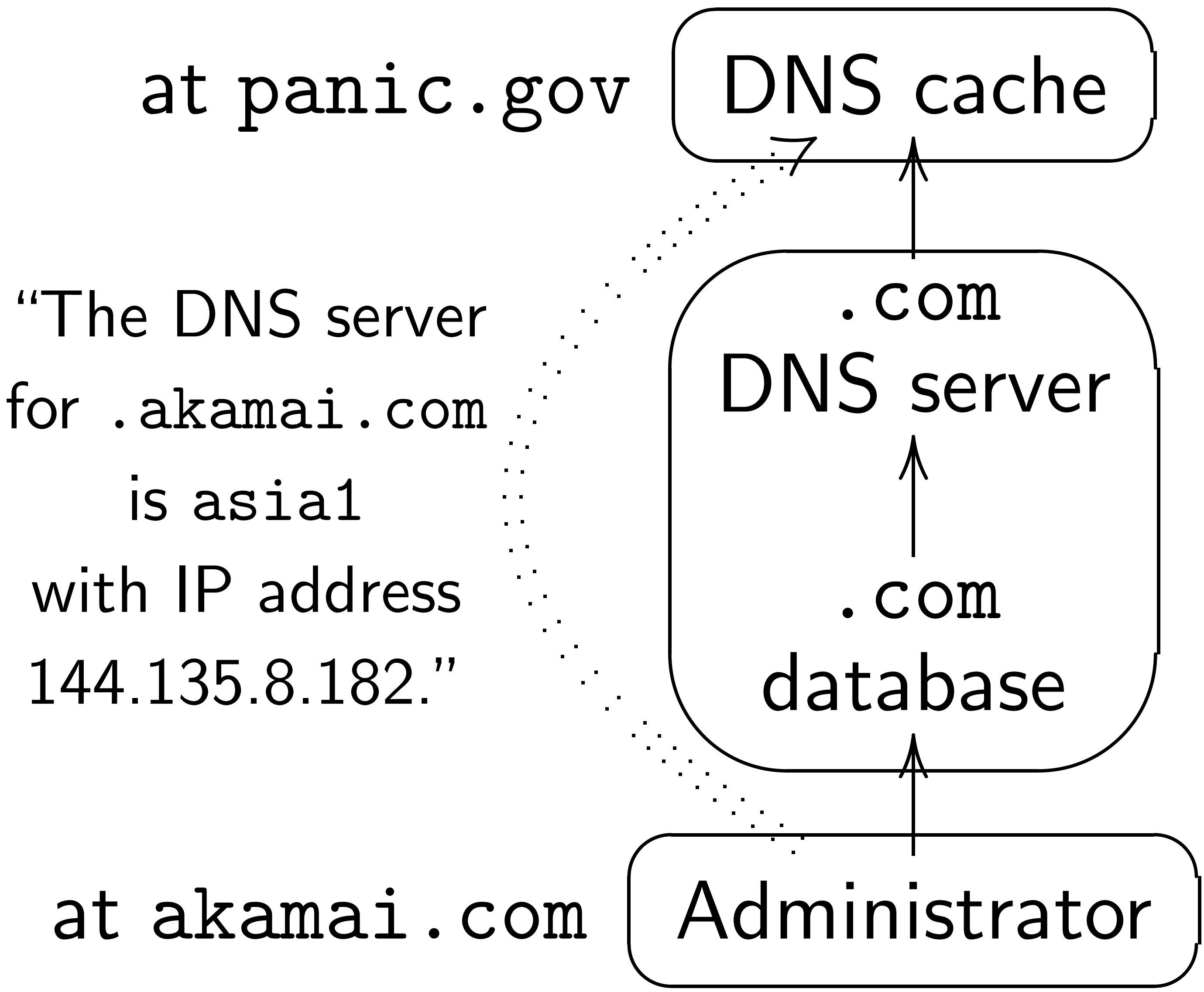
Paul Vixie, 1 November 2005:

> 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.

DNS in more detail



DNS cache learns location of `.akamai.com` DNS server from `.com` DNS server:



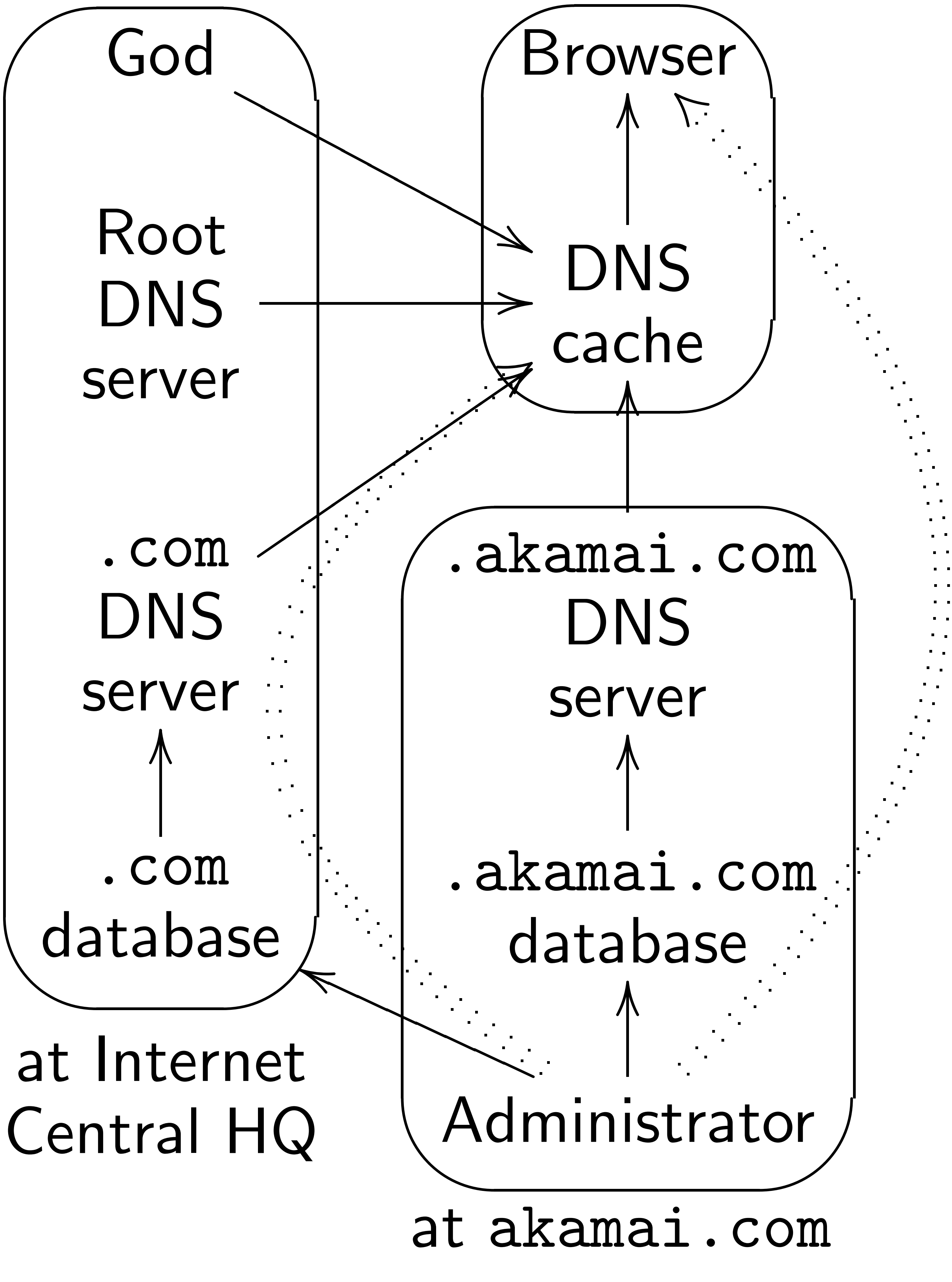
Packets to/from DNS cache

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

"Web `www.akamai.com`?"
193.0.14.129 ⟵ DNS cache
193.0.14.129 ⟶ DNS cache
"DNS `.com` `k` 192.52.178.30"

"Web `www.akamai.com`?"
192.52.178.30 ⟵ DNS cache
192.52.178.30 ⟶ DNS cache
"DNS `.akamai.com` `asia1` 144.135.8.182"

"Web `www.akamai.com`?"
144.135.8.182 ⟵ DNS cache
144.135.8.182 ⟶ DNS cache
"Web `www.akamai.com` 64.212.198.121"


God
Root DNS server
.com DNS server
.com database
at Internet Central HQ
Browser
DNS cache
.akamai.com DNS server
.akamai.com database
Administrator
at akamai.com

Making DNS secure

Many popular ways to authenticate cache $\rightarrow$ browser: e.g., IPSEC, or put cache on same box as browser.

Other local communication: same.

Limited risk for God $\rightarrow$ cache: information on this channel is small, stable, widespread.
Keep safe local copy of result.

Root $\rightarrow$ cache: similar;
can keep safe local copy,
although somewhat unstable.

Many popular ways to authenticate Akamai admin $\rightarrow$ `.com`: e.g., SSL-encrypted passwords.

Be careful: In January 2001, someone fooled Internet HQ into accepting fake Microsoft data; many similar incidents.

Remaining channels, the big DNS security problems: `.com` server $\rightarrow$ cache and `.akamai.com` server $\rightarrow$ cache.

Need to use public-key signatures to protect these channels.

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?

Akamai administrator signs `www.akamai.com` information under `.akamai.com` public key. Cache needs safe copy of that key.

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

Current DNSSEC approach:

`.com` server sends second Akamai key to cache, signed by `.com` key; `.akamai.com` server sends first Akamai key to cache, signed by second key.

New software for DNS servers, `.com` database to store keys, and `.akamai.com` database.

No reason to change software!

`.com` server has to sign
“`.akamai.com asia1` 144.135.8.182”
anyway. Embed Akamai key k
into `asia1` 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`.akamai.com`
instead of `www.akamai.com`.

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 sigs?

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 `xx.yz.akamai.com`
data from `.akamai.com`
with signature under key k.
Instead requests data for
$r.m_3$`.xx.yz.`$k.m_3$`.akamai.com`
where r is a cookie.
Rejects unsigned results.

(Cookie stops blind attacks.)

Simplified example
in BIND format:

`.akamai.com` server has

`*.123.www.8675309.123.akamai.com.`

`TXT "A 64.212.198.121 ..."`

where `...` is a signature of

`www A 64.212.198.121`

under Akamai's key 8675309.

`.com` server has

`*.akamai.3141592.123.com.`

`TXT "akamai NS`

`8675309.789 144.135.8.182 ...".`

Cache wants data for `www.akamai.com` or `www.8675309.456.akamai.com`.

Asks `.com` server about

`237.123.www.akamai`
`.3141592.123.com`.

Checks signature under key 3141592.

Asks `.akamai.com` server about

`291.123.www`
`.8675309.123.akamai.com`.

Checks signature under key 8675309.

Precomputation hassles

Popular DNS server receives > 10000 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: `.akamai.com` controls `quizno357.akamai.com` etc.

DNSSEC approach: Sign wildcards such as "there are no names between `quaalude.akamai.com` and `quizzical.akamai.com`." Big problem: saves time for snoops invading DNS privacy.

Better: Sign only real names. Legitimate users never ask about `quizno357.akamai.com`, so they don't need it signed.

The `.com` database is $\approx$ 2GB.
With signatures,
several times larger;
won't fit into memory.
(Virtual memory 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.
The Internet can trivially afford
a few big `.com` servers.

What's next?

First step:
build state-of-the-art
cryptographic tools.
Need small public keys;
fast signing; small signatures;
extremely fast verification.

Second step:
deploy DNS caches
verifying signatures
using mechanisms m_1, m_2, m_3.

Third step:
deploy DNS signing tool
and start signing data!