Internet security

D. J. Bernstein
University of Illinois at Chicago

Thanks to:
NSF ITR–0716498

Internet insecurity

D. J. Bernstein
University of Illinois at Chicago

No thanks to:
NSF ITR–0716498

Why doesn’t the Internet use cryptography?

Why doesn't the Internet use cryptography?

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

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 ≈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 ⇒ 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?

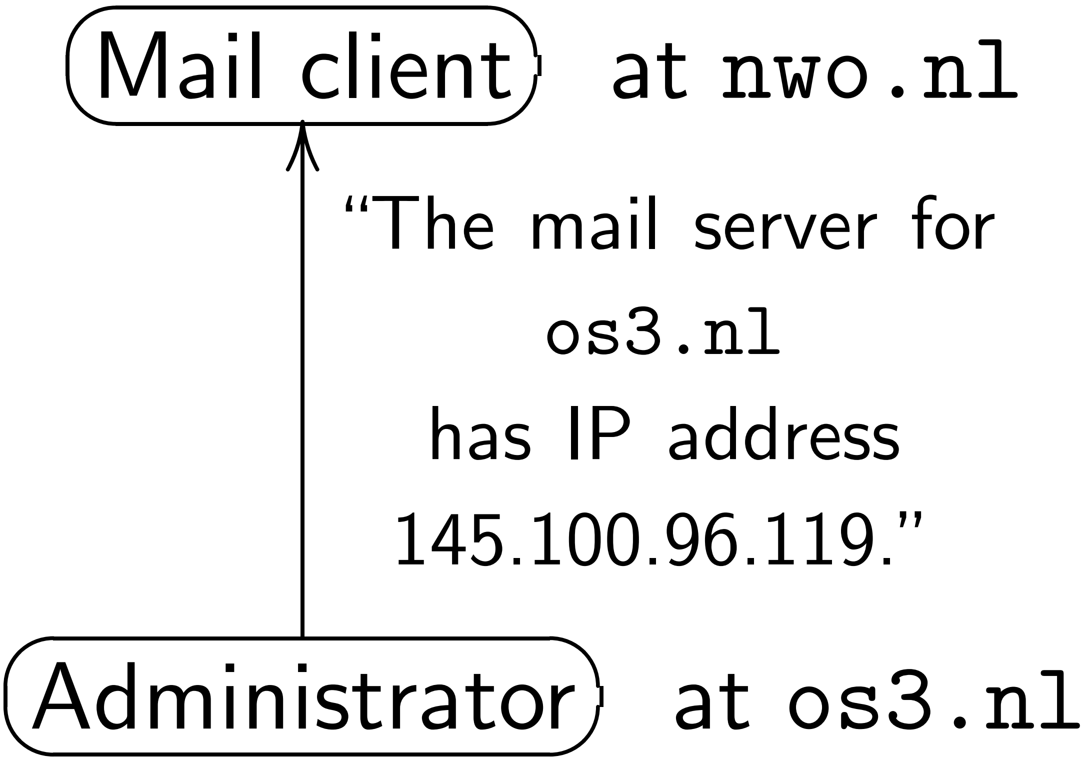
Can crypto be fast enough
to solidly protect all of
Google's communications?

Can crypto be fast enough
to protect every Internet packet?

Can universal crypto be *usable*?

The Domain Name System

nwo.nl has mail to deliver to someone@os3.nl.



Mail client at nwo.nl

"The mail server for os3.nl has IP address 145.100.96.119."

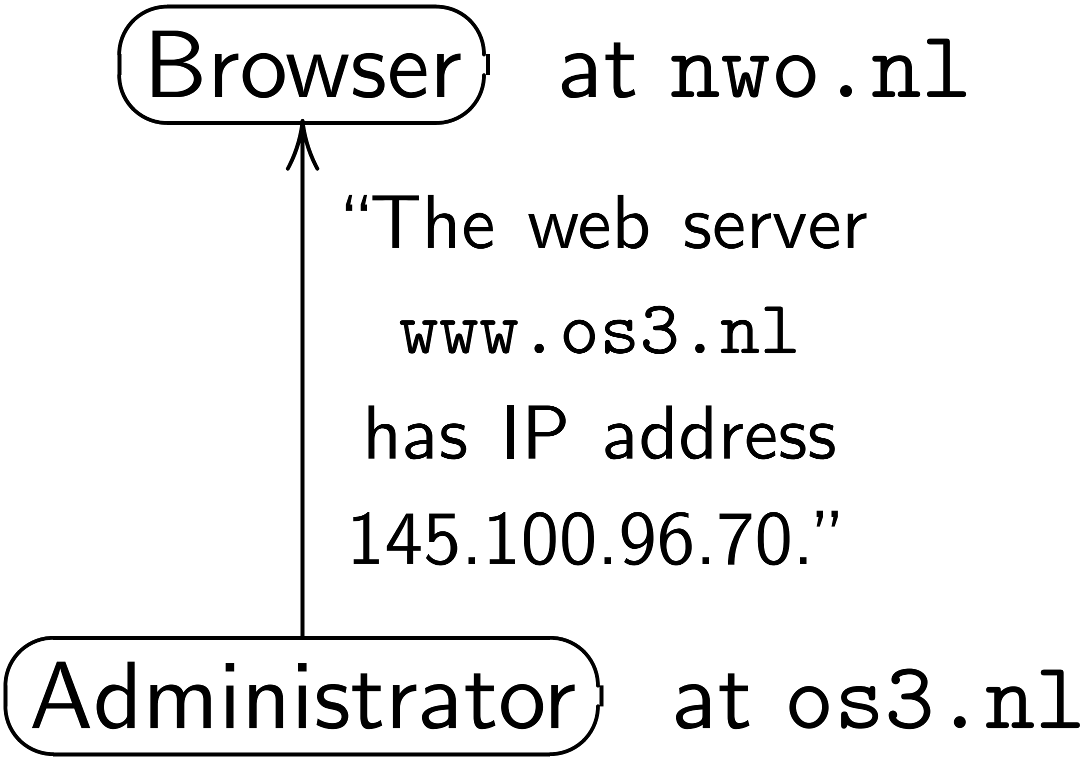
Administrator at os3.nl

Now nwo.nl delivers mail to IP address 145.100.96.119.

Same for web browsing.

`nwo.nl` wants to see

`http://www.os3.nl`.



Browser at `nwo.nl`

“The web server
`www.os3.nl`
has IP address
145.100.96.70.”

Administrator at `os3.nl`

Now `nwo.nl`
retrieves web page from
IP address 145.100.96.70.

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

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.

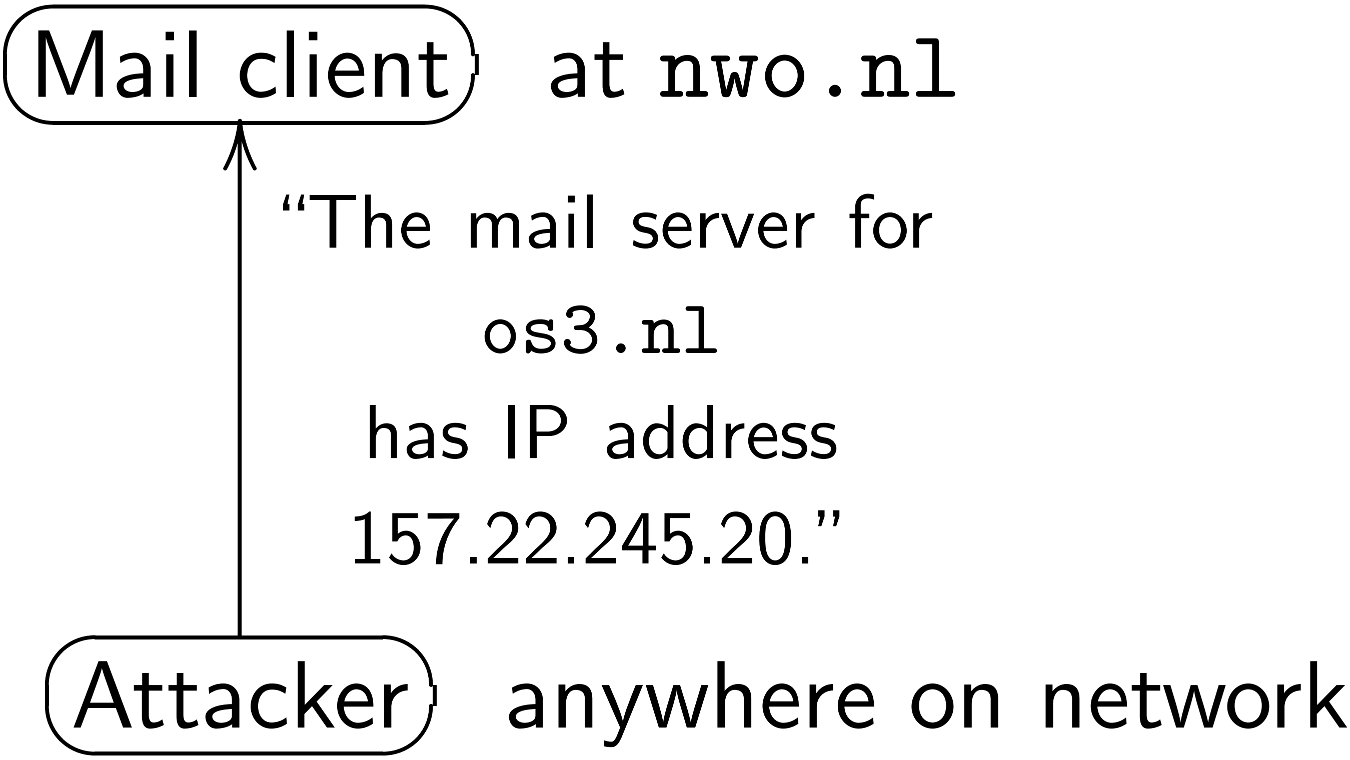
DNS software security holes:
BIND libresolv buffer overflow,
Microsoft cache promiscuity,
BIND 8 TSIG buffer overflow,
BIND 9 dig promiscuity, etc.

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

`nwo.nl` has mail
to deliver to `someone@os3.nl`.


Mail client at `nwo.nl`

"The mail server for
`os3.nl`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `nwo.nl`
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.os3.nl`?”

“Is the client always listening for the address of `www.os3.nl`?”

— 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.

“Is the client always listening for the address of `www.os3.nl`?”

— 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 `os3.nl`?”

“Doesn’t the attacker have to win a race against the legitimate DNS packets from the administrator at `os3.nl`?”

— 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.

“Doesn’t the attacker have to win a race against the legitimate DNS packets from the administrator at `os3.nl`?”

— 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 an 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.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

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.

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 serious 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 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 70000000 `*.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 70000000 `*.com` names.

Surveys by DNSSEC developers, last updated 2008.10.09, have found 133 `*.com` names with DNSSEC signatures. 116 on 2008.08.20; $133 > 116$.

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?

Effects of precomputation:

1. Pain for implementors. Hundreds of DNS tools need to be modified to precompute and store signatures.

2. Pain for administrators, far beyond a simple upgrade.

3. Reduced privacy. DNSSEC publishes all "secured" DNS records. (NSEC3: almost all.)

4. Reduced reliability, "DNSSEC suicide," reduced denial-of-service protection, etc.

Rethinking signatures

What we've learned:
DNSSEC precomputes
public-key signatures for speed,
creating severe usability problems
while sacrificing security.

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

Rethinking signatures

What we've learned:
DNSSEC precomputes
public-key signatures for speed,
creating severe usability problems
while sacrificing security.

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.

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 Core 2 Quad costing under \$200.

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
Core 2 Quad costing under \$200.

Handling 4 trillion partners/day:
under \$1000000.

Administration, implementation:
Much simpler than DNSSEC.
See `http://dnscurve.org`.

Packets are encrypted
just before transmission,
decrypted upon receipt.
Minimal impact on software.

Administration, implementation:
Much simpler than DNSSEC.
See `http://dnscurve.org`.

Packets are encrypted
just before transmission,
decrypted upon receipt.
Minimal impact on software.

Final question for today:
Can this protection
be pushed beyond DNS
to protect every Internet packet?