# High-speed cryptography and DNSCurve

### D. J. BernsteinUniversity of Illinois at Chicago

Stealing Internet mail: easy!

Given a mail message: Your mail software sends a DNS request, receives a server address, makes an SMTP connection, sends the From/To lines, sends the mail message.

Attackers can easily see all of these packets and change the packets. Forging web pages: easy!

Starting from a URL: Your browser sends a DNS request, receives a server address, makes an HTTP connection, sends an HTTP request, receives a web page.

Attackers can easily see all of these packets and change the packets.

In theory:

Cryptography stops these attacks.

In theory:

Cryptography stops these attacks.

In practice:

Am I using cryptography?

Are you using cryptography?

In theory:

Cryptography stops these attacks.

In practice:

Am I using cryptography?

Are you using cryptography?

Occasionally yes; usually no.

In theory:

Cryptography stops these attacks.

In practice: Am I using cryptography? Are you using cryptography? Occasionally yes; usually no.

Problem 1:

Most Internet protocols

do not support cryptography.

Why not? Obvious answer: Hard for protocol designers to integrate cryptography.

### Some popular Internet protocols *do* have cryptographic options. Important example: HTTPS.

Some popular Internet protocols *do* have cryptographic options. Important example: HTTPS.

Problem 2:

Most *implementations* 

of these protocols

do not support cryptography.

Why not? Obvious answer: Hard for software authors to integrate cryptography. Much easier to implement the non-cryptographic option. Some popular implementations *do* support cryptography. Example: Apache. Some popular implementations do support cryptography. Example: Apache. Problem 3: Most installations of these implementations do not support cryptography. pprox 99% of the Apache servers on the Internet do not enable SSL. Why not? Obvious answer: Hard for site administrators to turn on the cryptography.

Some important installations *do* support cryptography.

Example: SourceForge has paid for an SSL certificate and set up SSL servers. Try https:// sourceforge.net/account. Some important installations *do* support cryptography.

Example: SourceForge has paid for an SSL certificate and set up SSL servers. Try https:// sourceforge.net/account.

Problem 4: Cryptography is not enabled for most *data* at these installations.

Example: Try https://
sourceforge.net/community.
SourceForge redirects
your browser to http://
sourceforge.net/community.

### Why does SourceForge actively turn off cryptographic protection?

Why does SourceForge actively turn off cryptographic protection?

Obvious answer: Enabling SSL for more than a small fraction of SourceForge connections would massively overload the SourceForge servers.

SourceForge doesn't want to pay for a bunch of extra computers.

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

#### Making progress

Obvious speed questions:

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 SourceForge's communications?

Can crypto be fast enough to protect every Internet packet?

And questions beyond speed:

Can universal crypto be easy to use and administer?

Can universal crypto be easy to implement in software?

Can universal crypto be easy to add to protocols?

Can universal crypto be usable?

U.S. government, last century: "Encryption is dangerous! It can be used by terrorists, drug dealers, pedophiles, and money launderers!" U.S. government, last century:"Encryption is dangerous!It can be used by terrorists,drug dealers, pedophiles,and money launderers!"

I say: Criminals have been using encryption for a long time. Low speed? Hard to use? They use it anyway. We cannot stop them. U.S. government, last century: "Encryption is dangerous! It can be used by terrorists, drug dealers, pedophiles, and money launderers!"

I say: Criminals have been using encryption for a long time. Low speed? Hard to use? They use it anyway. We cannot stop them.

What we can do is improve the speed and usability of cryptography for normal people. My current mission: Cryptographically protect every Internet packet against espionage, corruption, and sabotage.

Confidentiality despite espionage: Spies cannot understand packets.

Integrity despite corruption: Forged packets are detected. User does not see wrong data.

Availability despite sabotage: User *does* see *correct* data.

### Securing DNS

DNSCurve cryptographically protects DNS packets against espionage, corruption, and sabotage.

DNSCurve is only for DNS, but same ideas can be adapted to many other protocols.

Warning: DNSCurve *does not* hide packet length, sender, etc. But it does provide confidentiality for contents of packets, plus strong integrity, availability.

# Packet from DNSCurve client to DNSCurve server:

- Here's my public key.
- Here's an encrypted DNS query.

Client encrypts, authenticates using client's secret key, server's public key.

Server verifies, decrypts using server's secret key, client's public key.

## Packet from DNSCurve server to DNSCurve client:

 Here's an encrypted response.
 Server encrypts, authenticates using server's secret key, client's public key.

Client verifies, decrypts using client's secret key, server's public key. Every packet is authenticated.

Client verifies every packet immediately upon receipt.

If packet fails verification, client discards packet and waits for correct packet.

Attacker can stop correct packet by flooding the network, but this consumes many more attacker resources than sending a few forged packets.  $\Rightarrow$  Many fewer victims. How does DNSCurve client retrieve server's public key? Does it send more packets? No! DNS architecture: DNS client learns IP address of .ubuntu.com DNS server from .com DNS server.

The .com server says: "The ubuntu.com DNS server is named ns3 and has IP address 209.6.3.210." The name ns3 was selected by the ubuntu.com administrator and given to .com.

To announce his DNSCurve server's public key, the ubuntu.com administrator changes the name ns3 to an encoding of the public key.

The DNSCurve client sees the public key, begins cryptographically protecting communication with that server.

#### Cryptography in DNSCurve

Critical cryptographic operations:

Encrypt and authenticate packet using server's secret key and client's public key.

Verify and decrypt packet using client's secret key and server's public key.

Need serious security, not something breakable today by Storm, NSA, ... (and next decade by academics). Could use public-key encryption (e.g., 4096-bit RSA encryption) and public-key signatures (e.g., 4096-bit RSA signatures).

But why use two separate public-key operations? Combined operations are faster.

Why use *signatures* that everyone can verify? Better to use *authenticators* verifiable by the recipient. When client and server exchange several messages, why use several separate public-key operations?

Classic "hybrid" speedup: Client and server use public-key operations to share a secret, and use secret-key cryptography to protect many messages.

Elliptic-curve cryptography: Client has secret key c, public key Curve(c). Server has secret key s, public key Curve(s). Client, server can cache shared secret Curve(*cs*), use secret-key cryptography to protect many messages. Introduced in 1985. Today's best attacks against random elliptic curves use as much computer power as 1985's best attacks.

1990s: ECC security criteria were standardized by IEEE P1363.

NIST used IEEE P1363 procedure to create several standard curves, such as the "P-256" curve.

More recent research recommends extra criteria to simplify and acclerate secure implementations. NIST P-256 flunks those criteria. The new "Curve25519" curve passes the IEEE P1363 criteria and the extra criteria. DNSCurve uses Curve25519.

#### So how fast is it?

New public-domain "Networking and Cryptography library", http://nacl.cace-project.eu:

crypto\_box encrypts and authenticates a packet.

Can split crypto\_box into crypto\_box\_beforenm, crypto\_box\_afternm to cache and reuse shared secret.

crypto\_box\_open verifies and decrypts a packet.

Using this software, a low-cost PC with a 2.4GHz Core 2 Quad CPU can encrypt and authenticate 50 billion packets/day to 500 million clients.

Also highly space-efficient: 32 bytes for a public key; similar overhead per packet. The *total* load on .com is 38 billion packets/day from 5 million clients.

"Project Titan":

The .com operators

are spending \$10000000

to be ready for a 200Gbps flood.

A worst-case 200Gbps

cryptographic flood

can be handled by a few thousand

PCs running this software.

#### DNSSEC vs. DNSCurve

DNSSEC was designed to minimize server load by precomputing signatures. "No per-query crypto."

DNSCurve does per-query crypto and is clearly fast enough.

(Is DNSSEC actually faster for servers than DNSCurve? "NSEC3" needs many hashes and database lookups. Huge signature databases punish the CPU's cache.) DNSSEC's approach hurts security.

It eliminates encryption, leaks private DNS databases, makes DNSSEC vulnerable to replay attacks, encourages low-security cryptographic choices (640-bit to 1024-bit RSA for fast signature verification), and enables amplification.

DNSCurve avoids all this.

Frederico Neves issued a challenge on Wednesday: Can anyone actually exploit DNSSEC's leaks to find the \*.sec3.br names?

By exploiting DNSSEC I've now computed 23 of the 26 names. Examples: douglas, pegasus, rafael, security, unbound, while42, zz--zz.

Thanks to Tanja Lange at Eindhoven for assistance.

DNSSEC's approach hurts programmers and users.

DNSSEC has to generate, store, and often regenerate signatures, plus complications: NSEC3 etc. DNSSEC forces changes in hundreds of DNS management tools, DNS servers, etc. that DNSCurve already protects.

After fifteen years of work, the DNSSEC software changes are still very far from done. That's why .org's new "signatures" are easily breakable. Christian Grothoff, yesterday:

"Good security is more costly and harder to understand and deploy than bad security."

Not always. Fear of bad performance often leads to designs with bad security, bad implementability, bad usability, and mediocre performance.

When we instead design secure, easy-to-use systems, sometimes they turn out to have perfectly acceptable performance!