Research sample: Cryptographic protocol design

Daniel J. Bernstein
djb@cr.yp.to

2007.01.15

I've asked many people—other cryptographers, other computer-security researchers, system administrators, end users—to explain to me why the Internet doesn't use cryptography. Let's look at some of the answers:

- "The Internet does use cryptography! I just made an SSL connection to my bank. Firefox showed me the little cryptographic lock in the corner."

  There are, indeed, many SSL web connections, and many Internet phone calls using Skype, and so on. But there are many more Internet connections that have no cryptographic protection and that are trivially vulnerable to espionage and sabotage. Why is there so much unprotected Internet communication?

- "Nobody cares about cryptography. Cryptography is pointless. Attackers are exploiting buffer overflows; they aren't intercepting or forging packets."

  This argument—a condensed version of Steve Bellovin's talk at PKC 2006—is (1) shortsighted and (2) factually incorrect. Regarding (1): To protect the Internet we need to fix buffer overflows *and* packet forgeries *and* a variety of other problems. If we fix some problems and ignore others, attackers will simply start exploiting the others!

  Regarding (2): There is a long history of attackers intercepting and forging packets. Kevin Mitnick's famous 1994 invasion of Tsutomu Shimomura's computers relied on packet forgery; see `http://www.cs.berkeley.edu/~daw/security/shimo-post.txt`. Password interception motivated SSH. Credit-card-number interception motivated SSL. Internet phone-call sniffing (see, e.g., `http://www.securityfocus.com/infocus/1862/1`) is scaring users away from non-cryptographic voice-over-IP software. David Pogue, the New York Times technology columnist, recently posted `http://pogue.blogs.nytimes.com/2007/01/04/04pogue-email/`, all about Wi-Fi interception. Users obviously value cryptography; so why are they so often ignoring it?

- "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. Many network protocols don't even allow cryptographic protection as an option."

  All of this is true; but let's focus for the moment on HTTP. The most popular web server (Apache, running 60% of all web sites according to Netcraft's end-of-2006 survey and over 70% according to SecuritySpace's end-of-2006 survey) and the most popular clients (Internet Explorer, Firefox, etc.) all support some cryptography, specifically SSL—but I've watched traffic passing through a variety of campus networks, hotel networks, cable networks, airport wireless networks, etc., and in each case SSL was used for only a tiny fraction of all web connections. Why?

- "Have you ever tried to set up SSL? I don't want to go through all these extra Apache configuration steps, and I don't want to pay for a certificate, and I don't want to annoy my web-site visitors with self-signed certificates."

  Only about 1% of the Apache servers on the Internet have SSL enabled, according to SecuritySpace; usability is obviously a major issue. But let's focus for the moment on a site that has already paid for a certificate and set up SSL. When I type `https://www.google.com`

or `https://www.google.com/search?as_q=tue` into my browser, Google redirects the browser to `http://www.google.com`. Google does SSL-protect some of its user interactions (try `https://mail.google.com`) but doesn't SSL-protect searches. Why not?

- "Enabling SSL for more than a small fraction of Google searches would overload the Google servers. Google doesn't want to pay for a bunch of extra computers."

  Does cryptography actually have to be so expensive? Can we make it fast enough to protect all of Google's communications? Can we make it fast enough to protect all of the Internet's communications?

  Wikipedia says that SSL accelerators—hardware devices to handle the cryptographic operations in SSL—are being sold by "Fastream Technologies, jetNEXUS, Array Networks, F5 Networks, Hifn, nCipher, Nortel Networks, Radware, and Sun Microsystems." Clearly SSL speed is important for many web-site administrators. I haven't tried to build a list of SSL-bottlenecked web sites, and I don't know that Google really is an example (although I haven't heard any alternate theories for Google's limited use of SSL), but I'm confident that speedups in SSL would have a real-world impact, reducing costs for many sites and making cryptography possible for many more. Similar comments apply to other Internet protocols.

Rethinking public-key signatures

One can use public-key signatures—RSA signatures, for example, or ECDSA signatures—to protect messages against forgery. The sender generates a secret key and a public key. The public key is broadcast through a preexisting channel, assumed secure against forgery. For each new message, the sender uses the secret key to generate a signature of the message. Everyone can verify the signed message using the public key.

Google, for example, could sign each of its web pages. The signature can be reused (given adequate caching) when the same web page is sent to multiple recipients, but there's still a signature computation for each unique web page. This is a fairly expensive computation—millions of CPU cycles for each web page using the fastest available software—and can easily be a bottleneck, depending on the number of web pages sent by Google. There's also a smaller cost for each recipient to verify each web page.

I've realized that there's a faster solution. It's easiest to understand the solution in four steps:

- **Add encryption.** Let's protect not just against forgery but also against eavesdropping, by combining public-key signatures with public-key encryption. Yes, this is slower; bear with me!
- **Merge encryption and signing.** "Public-key signcryption," introduced by Zheng in the 1990s, provides the same protection more effficiently. There is no need to partition signcryption into two components, an encryption component (which spends time worrying about a lack of signatures) and a signing component (which spends time worrying about a lack of encryption); combined algorithms are faster.
- **Eliminate signcryption of multiple messages.** It's silly for a sender to signcrypt two messages to the same recipient. It's much more efficient for the sender to signcrypt one key and use secret-key cryptography to protect both messages.
- **Eliminate signcryption of randomly generated keys.** At this point the only use of public-key operations is for the sender to signcrypt a random key sent to the recipient. But this is silly. There's a special key that has just as much apparent randomness and that's markedly easier to signcrypt: namely, the Diffie-Hellman secret shared between Alice and Bob.

At this point both "public-key encryption" and "public-key signcryption" have been eliminated in favor of the Diffie-Hellman protocol. The sender performs just one Diffie-Hellman computation for each new recipient, each computation costing under a million CPU cycles with my Curve25519 software; the result of this computation, a secret key, is then used at very low cost to protect all of the messages sent to that user. For example, Google could handle a billion new users in just a few days of time on one computer—and, of course, Google can spread computations across many computers.

One might complain about the cost of Google storing shared secrets in a big central database, but the standard technique of "remote storage" (Google stores the user's shared secret as an authenticated encrypted cookie on the user's machine) removes the need for a central database. Similar comments apply to other web sites; a web server might handle many hits every second, but how many *new* users show up every second?

This idea warrants investigation as a low-cost alternative to SSL. I plan to design and implement all necessary protocols and evaluate their real-world performance.

Rethinking DNS security

The Domain Name System is the backbone of the Internet, and one that has so far managed to resist all attempts at cryptographic protection. Twelve years of development of "DNSSEC" have been stymied by apparently insurmountable deployment difficulties.

One problem is that DNS poses severe performance challenges to the cryptographer: a tremendous volume of data has to be signed by heavily loaded servers; signed messages and keys have to fit comfortably into 512-byte packets; signatures have to be verified repeatedly by heavily loaded caches. Another problem is that DNSSEC exposes more site information than unmodified DNS, triggering widespread concern regarding EU privacy regulations. Another problem is that DNSSEC requires extensive modifications in the three major DNS-cache programs, dozens of DNS servers, and hundreds of DNS-database tools.

My assessment is that the last problem, namely the number of different pieces of software that need to be upgraded, causes the most trouble for potential DNSSEC users. But it turns out that a carefully designed protocol can transfer signatures with no modifications to existing servers and existing database tools. My redesigned DNSSEC2 protocol is explained in detail in `http://cr.yp.to/talks.html#2006.10.17`; I'm working on an implementation.