High-speed cryptography for mobile devices

D. J. Bernstein
University of Illinois at Chicago,
Technische Universiteit Eindhoven

Picture credits:
`geeky-gadgets.com`; Star Trek

The Internet of Things

Andrew Myers, Stanford Report, 2011.02.11:

“His wine cellar is networked. Cerf can monitor and control the temperature, humidity and other important information from his smartphone.”

“Welcome to the ‘Internet of things,’ a much-discussed vision of a tomorrow in which virtually every electronic device—ovens, stereos, toasters, wine cellars—will be networked.”

. . . "Security, however, is the real looming cloud, Cerf said."

. . . "Security, however, is the real looming cloud, Cerf said."

Security (confidentiality,
integrity, availability)
of wireless communication
relies critically on crypto.

. . . "Security, however, is the real looming cloud, Cerf said."

Security (confidentiality, integrity, availability) of wireless communication relies critically on crypto.

Frequently asked question:
Can all these tiny devices keep up with the crypto?

. . . "Security, however, is the real looming cloud, Cerf said."

Security (confidentiality, integrity, availability) of wireless communication relies critically on crypto.

Frequently asked question:
Can all these tiny devices keep up with the crypto?

Question in this talk:
Can the smartphone keep up with the crypto?

Conventional wisdom:

Crypto for tiny devices
is much more challenging
than smartphone crypto.

Smartphones have big CPUs.
Tiny devices usually have
much smaller CPUs.
Expect CPU gap to *increase*
with deployment of many
ultra-low-cost devices.

$\Rightarrow$ Study smartphone crypto
only as an easy warmup before
studying crypto for tiny devices.

This wisdom is flawed.

Easy to see how
smartphone CPU can be
the most troublesome bottleneck.

This wisdom is flawed.

Easy to see how
smartphone CPU can be
the most troublesome bottleneck.

Communication is centralized:
many tiny devices
are talking to *one* smartphone.

This wisdom is flawed.

Easy to see how
smartphone CPU can be
the most troublesome bottleneck.

Communication is centralized:
many tiny devices
are talking to *one* smartphone.

As tiny-device cost drops,
expect dramatic increase in
number of tiny devices,
and thus load on smartphone.

Will smartphone CPU power
increase so dramatically?


SAMSUNG


SAMSUNG

Smartphone/tablet CPUs

iPad 1 (2010) contains 45nm
Apple A4 system-on-chip.

Apple A4 contains
1GHz ARM Cortex A8 CPU core
+ PowerVR SGX 535 GPU.

Cortex A8 CPU core
supports ARMv7 instruction set,
including NEON vector insns.

iPhone 4 (2010)
also contains Apple A4.

45nm 1GHz Samsung Exynos 3110 in Samsung Galaxy S (2010) contains Cortex A8 CPU core.

45nm 1GHz TI OMAP3630 in Motorola Droid X (2010) contains Cortex A8 CPU core.

45nm? 800MHz Freescale i.MX50 in Amazon Kindle 4 (2011) contains Cortex A8 CPU core.

40nm/55nm? Allwinner A10 (2012) in set-top boxes etc., reportedly $7 in volume, contains Cortex A8 CPU core.

More ARMv7+NEON cores:

2$\times$ Cortex A9 in Apple A5 in iPad 2 (2011), iPhone 4 (2011);

4$\times$ Cortex A9 in Nvidia Tegra 3 in Asus Eee Pad Transformer Prime (2011);

2$\times$ Krait in Qualcomm MSM8960 Snapdragon S4 in HTC One XL (2012);

2$\times$ Cortex A15 in Samsung Exynos 5250 in Google Nexus 10 (2012);

etc.

ARMv7+NEON universal?

Not quite.

Some exceptions:

ARM1136 in Qualcomm
MSM7200A in Samsung
GT i7500 Galaxy (2009),
first Samsung Android phone.

Cortex A9 *without* NEON
in Nvidia Tegra 2
in Motorola Droid X2 (2011).

Intel Atom Z2460
in Motorola RAZR I (2012).

High-speed cryptography

Typical question:
"How fast is AES-128-CTR?"

25 Cortex A8 cycles/byte for Polyakov code in OpenSSL; not protected against timing attacks.

19 Cortex A8 cycles/byte for 2012 Bernstein–Schwabe; protected against timing attacks. Based on bitsliced software from 2009 Käsper–Schwabe.

Better question:
"How fast is high-security encryption using a secret key?"

Much tougher for implementors.
Vary cipher *and* implementation, instead of just implementation.

Better question:
"How fast is high-security encryption using a secret key?"

Much tougher for implementors.
Vary cipher *and* implementation, instead of just implementation.

Better results for users.
AES is designed for an oversimplified CPU model, ignoring CPU design trends and physical hardware costs:
AES is designed to use *loads*.

ECRYPT Stream Cipher Project (eSTREAM), 2004–2008, selected portfolio of four software ciphers: HC-128, Rabbit, Salsa20/12, SOSEMANUK. (Also some hardware ciphers.)

Salsa20 (2005 Bernstein): 20-round ARX stream cipher, 256-bit key; permutation-based (single-key Even–Mansour) cipher in counter mode.

Salsa20 is designed to use *vectorized arithmetic*.

Salsa20 cryptanalytic papers by Aumasson, Berbain, Biasse, Biryukov, Castro, Crowley, Estevez-Tapiador, Fischer, Ishiguro, Khazaei, Kiyomoto, Kubo, Meier, Miyake, Nakashima, Pelissier, Priemuth-Schmid, Quisquater, Rechberger, Robshaw, Saito, Suzaki, Tsunoo: 2^{249} attack against 8 rounds.

Top-ranked software cipher in polls at SASC 2007, SASC 2008. eSTREAM: 12 rounds is fine. I'm conservative: 20 rounds.

64-byte Salsa20 output block:
320 ARX sequences such as

```
s4 = x0 + x12
x4 ^= (s4 >>> 25)
```

operating on 32-bit integers.
i.e. 5 ARX sequences/byte.

ARM *without* NEON:
2 insns; 1 Cortex A8 cycle.
Sounds like 5 cycles/byte.

Actually >15 cycles/byte:
reg problems, latency problems.

2012 Bernstein–Schwabe: optimize using NEON.

128-bit NEON vector insns: e.g. 4 32-bit ops/cycle.

```
4x a0 = diag1 + diag0
```

Good: many ops/cycle.

Good: simultaneous ARM+NEON instructions.

Good: tons of space in regs.

Bad: $4\times$ *same* op.

Bad: no vector `>>>` .

Salsa20 has $4\times$ same op;
can vectorize within block.

Salsa20 uses counter mode;
can vectorize across blocks.

We vectorize within block,
parallelize across 3 blocks,
use ARM+NEON simultaneously.

<6 cycles/byte,
protected against timing attacks.
Much faster than AES-128.

More crypto operations

Bernstein–Lange–Schwabe:
new cryptographic library,
NaCl ("salt").

Acknowledgments:
code contributions from
Matthew Dempsky (Mochi
Media), Niels Duif (Eindhoven),
Emilia Käsper (Leuven),
Adam Langley (Google),
Bo-Yin Yang (Academia Sinica).

Most of the Internet
is cryptographically unprotected.
Even when crypto is deployed,
it usually isn't secure.

Primary goal of NaCl: Fix this.

`nacl.cr.yp.to`: source
and extensive documentation.

Largest NaCl deployment so far:
DNSCrypt from OpenDNS,
high-security authenticated
encryption for DNS queries.

Critical NaCl design goals:

- No secret load addresses.
- No secret branch conditions.
- No padding oracles.
- Centralize randomness.
- Avoid unnecessary randomness.
- Avoid pure crypto failures.
- Speed.

Case study: EdDSA

1985 ElGamal signatures:
(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod{q}$
and $R, S \in \{0, 1, \ldots, q-2\}$.

Here q is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Signer generates A and R
as secret powers of B;
easily solves for S.

1990 Schnorr improvements:

1. Hash R in the exponent:
$B^{H(M)} \equiv A^{H(R)} R^S$.
Reduces attacker control.

2. Replace three exponents with two exponents:
$B^{H(M)/H(R)} \equiv A R^{S/H(R)}$.
Saves time in verification.

3. Simplify by relabeling S:
$B^{H(M)/H(R)} \equiv A R^S$.
Saves time in verification.

4. Merge the hashes:
$B^{H(R,M)} \equiv A R^S$.
$\Rightarrow$ Resilient to H collisions.

5. Eliminate inversions for signer:
$B^S \equiv RA^{H(R,M)}$.
Simpler, faster.

6. Compress R to $H(R, M)$.
Saves space in signatures.

7. Use half-size H output.
Saves space in signatures.

5. Eliminate inversions for signer:
$B^S \equiv RA^{H(R,M)}$.
Simpler, faster.

6. Compress R to $H(R, M)$.
Saves space in signatures.

7. Use half-size H output.
Saves space in signatures.

Subsequent research: extensive theoretical study of security of Schnorr's system.

5. Eliminate inversions for signer: $B^S \equiv RA^{H(R,M)}$.
Simpler, faster.

6. Compress R to $H(R, M)$.
Saves space in signatures.

7. Use half-size H output.
Saves space in signatures.

Subsequent research: extensive theoretical study of security of Schnorr's system.

But patented. $\Rightarrow$ DSA, ECDSA avoided most improvements.

5. Eliminate inversions for signer: $B^S \equiv RA^{H(R,M)}$.
Simpler, faster.

6. Compress R to $H(R, M)$.
Saves space in signatures.

7. Use half-size H output.
Saves space in signatures.

Subsequent research: extensive theoretical study of security of Schnorr's system.

But patented. $\Rightarrow$ DSA, ECDSA avoided most improvements.

Patent expired in 2008.

EdDSA (CHES 2011 Bernstein–Duif–Lange–Schwabe–Yang):

Use elliptic curves in "complete -1-twisted Edwards" form.
$\Rightarrow$ very high speed,
natural side-channel protection,
no exceptional cases.

Skip signature compression.
Support batch verification.

Use double-size H output,
and include A as input.

Generate R deterministically
as a secret hash of M.
$\Rightarrow$ Avoid PlayStation disaster.

Cortex A8 speed summary

2012 Bernstein–Schwabe:

<6 cycles/byte:
encrypt with Salsa20.
<3 cycles/byte:
authenticate with Poly1305.

ECC (Curve25519) public-key ops:
460200 cycles for DH.
624846 cycles to verify.
244655 cycles to sign.