How cryptographic benchmarking goes wrong

Daniel J. Bernstein

Thanks to NIST 60NANB12D261 for funding this work, and for not reviewing these slides in advance.

PRESERVE, ending 2015.06.30, was a European project "Preparing Secure Vehicle-to-X Communication Systems".

Project cost: 5383431 EUR, including 3850000 EUR from the European Commission.

"About PRESERVE": "The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges."

Cars already include many CPUs. Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification ... a dedicated cryptographic co-processor is likely to be necessary."

PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more." For 180nm core says max 100MHz, 100 verif/second.

Compare to, e.g.,
IAIK NIST P-256 ECC Module:
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm ("UMC L180GII
technology using Faraday f180
standard cell library (FSA0A_C),
9.3744 μm^2/GE; worst case
conditions (temperature 125°C,
core voltage 1.62V)").

Signature verification will be
somewhat slower than scalarmult.
Still close to 100× more efficient
than the PRESERVE estimates.

Let's go back to PRESERVE's core argument for an ASIC.

Central claim: "As discussed in [32], a Pentium D 3.4 GHz processor needs about" 5ms (i.e., 17 million CPU cycles) for signature verification.

[32] is "Petit, J., Mammeri, Z., 'Analysis of authentication overhead in vehicular networks', Third Joint IFIP Wireless and Mobile Networking Conference (WMNC), 2010."

[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of $230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."

Vehicles will communicate safety information. "All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. ... In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. ... Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]"

[10] (in [32]) is “Petit J., ‘Analysis of ECDSA Authentication Processing in VANETs’, 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009.”

[10] says “ECDSA was implemented using MIRACL and following the Fig.1.” For NIST P-224/P-256 on “Pentium D 3.4GHz workstation”: 2.50ms/3.33ms to sign, 4.97ms/6.63ms to verify.

Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles), 0.049ms to verify (163206 cycles).

Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles), 0.049ms to verify (163206 cycles).

This chip didn't exist in 2009. Compare instead to single core of 65nm 2.4GHz Core 2 ("2007 Intel Core 2 Quad Q6600").

0.065ms to sign (156843 cycles), 0.232ms to verify (557082 cycles).

2012 Bernstein–Schwabe
on 720MHz ARM Cortex-A8:
0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos 3110
in Samsung Galaxy S (2010);
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 in
Amazon Kindle 4 (2011); ...
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.

180nm 32-bit 2GHz Willamette
("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles)
for Curve25519 scalarmult
using floating-point multiplier.
Integer multiplier is much slower!

Nobody has ever bothered
adapting this to signatures.
Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual core):
same basic microarchitecture,
more instructions, faster clock.
Ed25519 would be >10× faster
on one core than Petit's software.

Bad ECDSA-NIST-P-256 design certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: $2\times$ slower.

2000 Brown–Hankerson–López–Menezes on 400MHz Pentium II: 4.0ms/6.4ms (1.6/2.6 million cycles) for double scalarmult inside NIST P-224/P-256 verif.

2001 Bernstein, $\approx 1.6\times$ faster: 0.7 million cycles on Pentium II for NIST P-224 scalarmult.

2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.

How did Petit manage to use
17 million cycles for P-224 verif,
22 million cycles for P-256 verif?

Presumably some combination of
bad mulmod and bad curve ops.

Why did Petit reimplement
ECDSA, using MIRACL for the
underlying arithmetic?

Why did Petit not simply cite
previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?

Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++.
Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says "For elliptic curves over prime fields, OpenSSL has the best performance under all platforms."

More general situation:
Paper analyzes impact of
crypto upon an application.

If the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

If the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem!
More likely to be published.
More likely to motivate
funding to fix the problem.

Obvious question whenever an application considers crypto deployment: "Is it fast enough?"

Many random methodologies for answering this question. Which CPU to test? What to take from literature and libraries? Reuse mulmod, or curve ops, or more?

Slowest, least competent answers are most likely to be published.

Situation is fully explainable by randomness + natural selection. There's no evidence that Petit *deliberately* slowed down crypto.

Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.

Bit operations per bit of plaintext (assuming precomputed subkeys), as listed in recent Skinny paper:

key	ops/bit	cipher
128	88	Simon: 60 ops broken
128	100	NOEKEON
128	117	Skinny
256	144	Simon: 106 ops broken
128	147.2	PRESENT
256	156	Skinny
128	162.75	Piccolo
128	202.5	AES
256	283.5	AES

Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

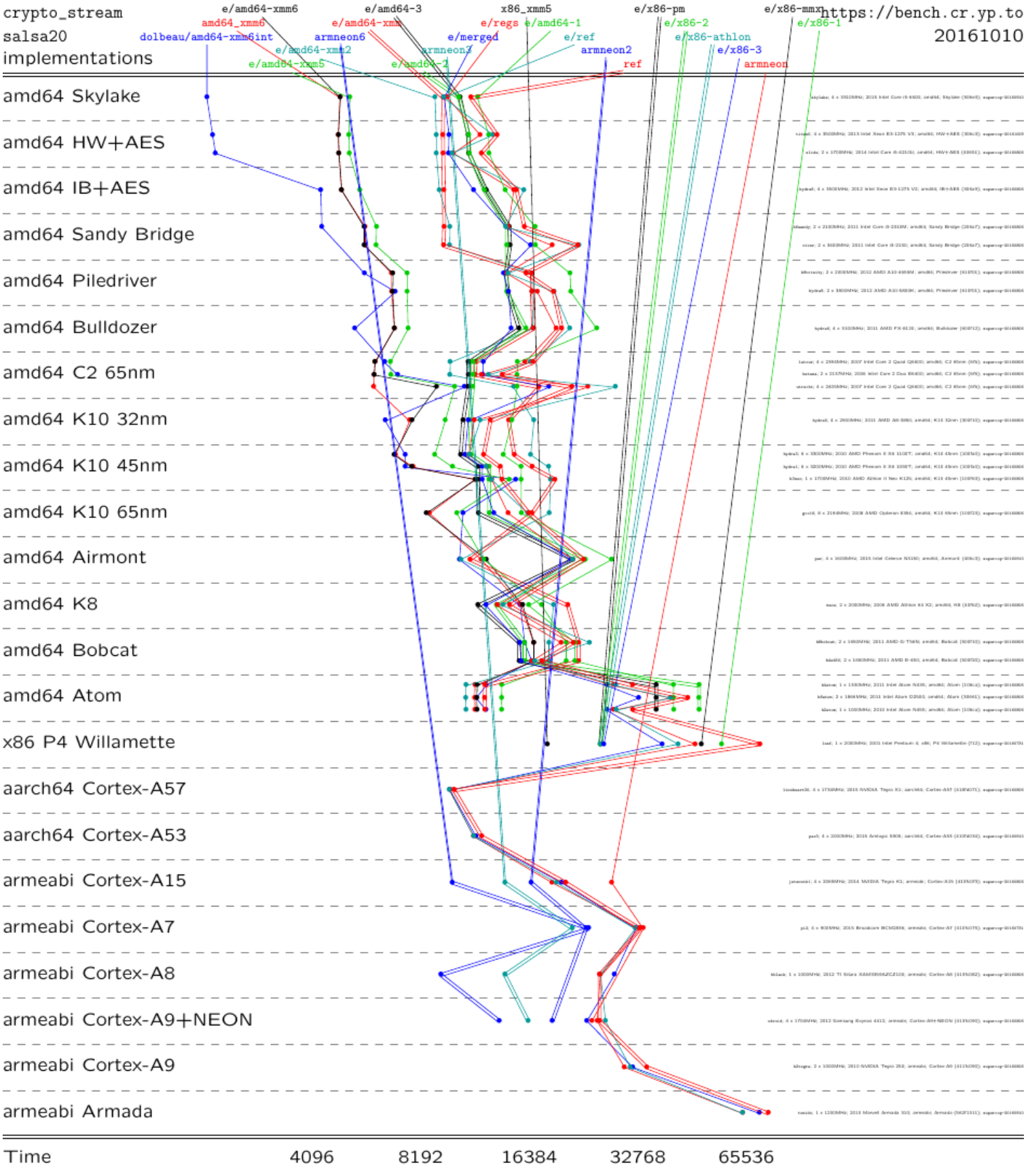
key	ops/bit	cipher
256	54	Salsa20/8
256	78	Salsa20/12
128	88	Simon: 60 ops broken
128	100	NOEKEON
128	117	Skinny
256	126	Salsa20
256	144	Simon: 106 ops broken
128	147.2	PRESENT
256	156	Skinny
128	162.75	Piccolo
128	202.5	AES
256	283.5	AES

Many bad examples to imitate, backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on "optimizing" compiler!

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

Reality is more complicated:



SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.

Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG $\Rightarrow$ cost of sampling $\Rightarrow$ cost of signing.

Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

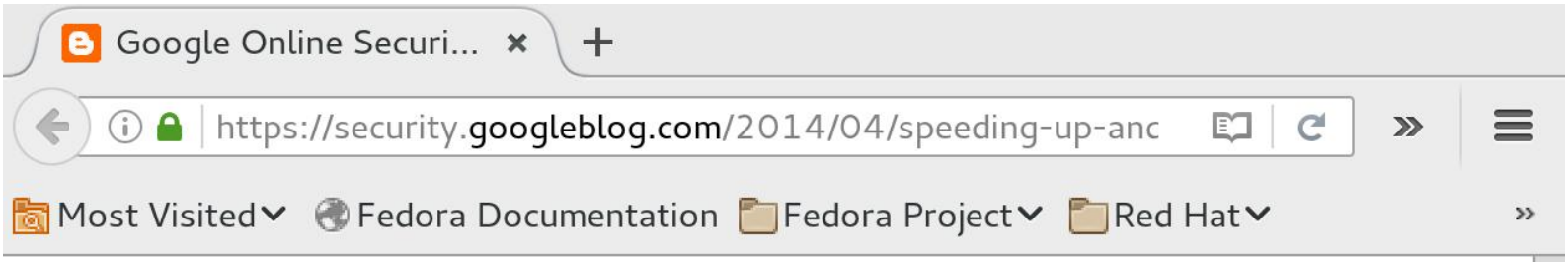
383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports 0.92 cycles/byte for AES-256-CTR, 1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?



Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

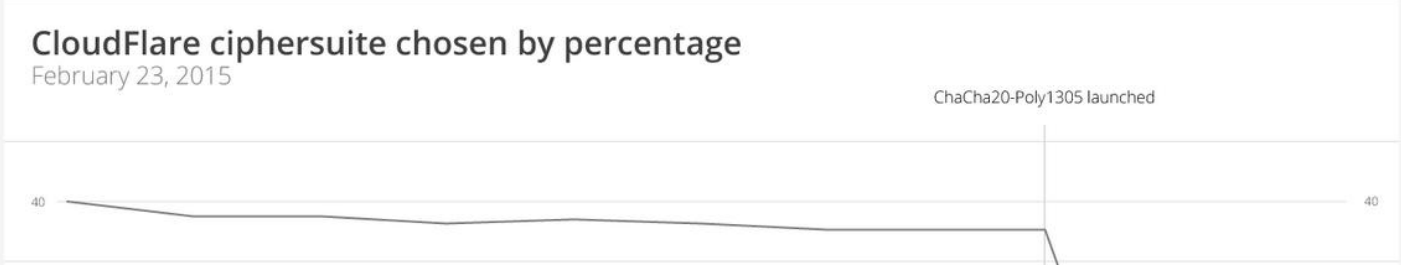
Posted by Elie Bursztein, Anti-Abuse Research Lead

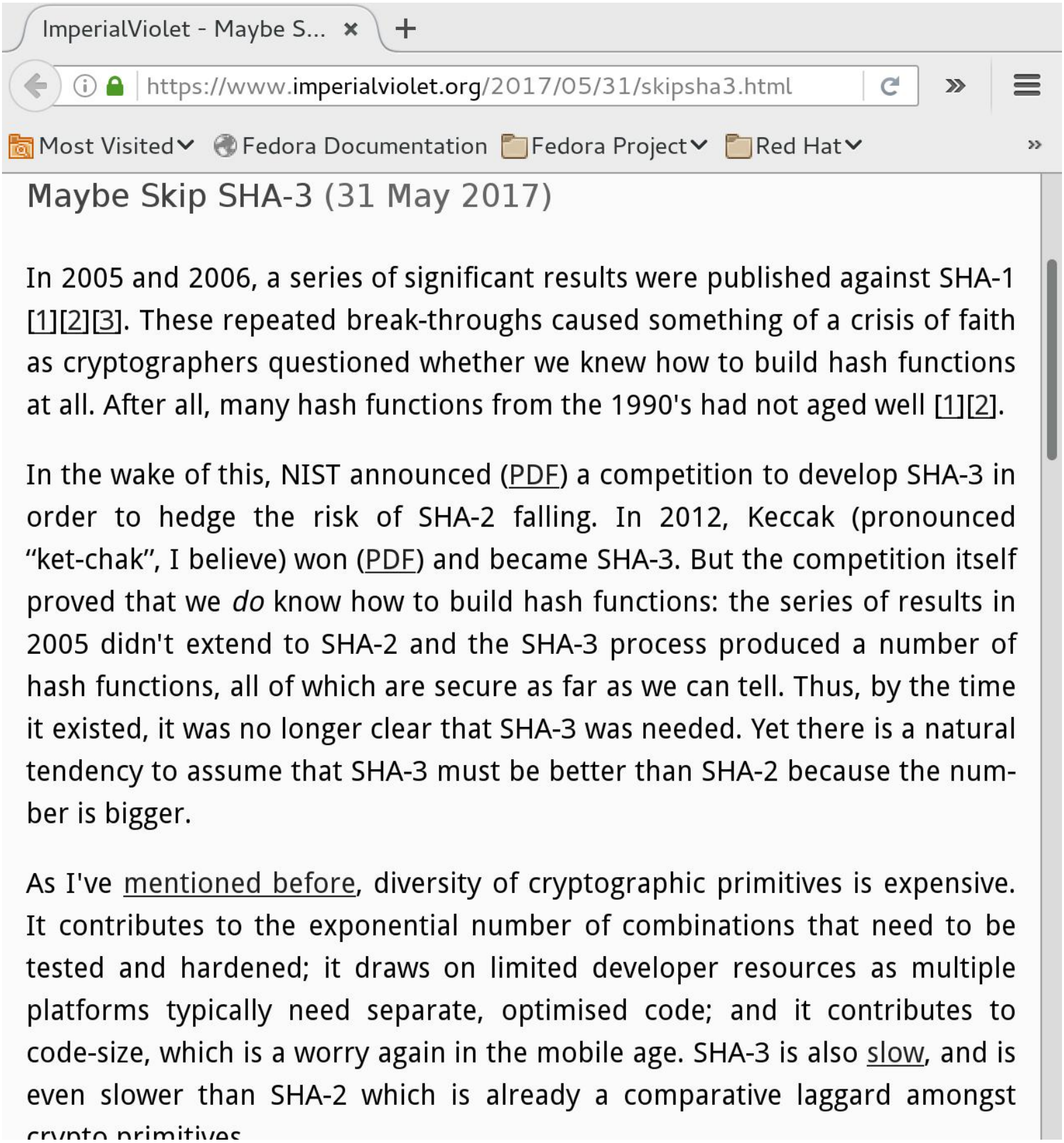
Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated

Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:





Maybe Skip SHA-3 (31 May 2017)

In 2005 and 2006, a series of significant results were published against SHA-1 [1][2][3]. These repeated break-throughs caused something of a crisis of faith as cryptographers questioned whether we knew how to build hash functions at all. After all, many hash functions from the 1990's had not aged well [1][2].

In the wake of this, NIST announced (PDF) a competition to develop SHA-3 in order to hedge the risk of SHA-2 falling. In 2012, Keccak (pronounced "ket-chak", I believe) won (PDF) and became SHA-3. But the competition itself proved that we *do* know how to build hash functions: the series of results in 2005 didn't extend to SHA-2 and the SHA-3 process produced a number of hash functions, all of which are secure as far as we can tell. Thus, by the time it existed, it was no longer clear that SHA-3 was needed. Yet there is a natural tendency to assume that SHA-3 must be better than SHA-2 because the number is bigger.

As I've mentioned before, diversity of cryptographic primitives is expensive. It contributes to the exponential number of combinations that need to be tested and hardened; it draws on limited developer resources as multiple platforms typically need separate, optimised code; and it contributes to code-size, which is a worry again in the mobile age. SHA-3 is also slow, and is even slower than SHA-2 which is already a comparative laggard amongst crypto primitives