

How fast is cryptography?

D. J. Bernstein

University of Illinois at Chicago

Joint work with:

Tanja Lange

Technische Universiteit Eindhoven

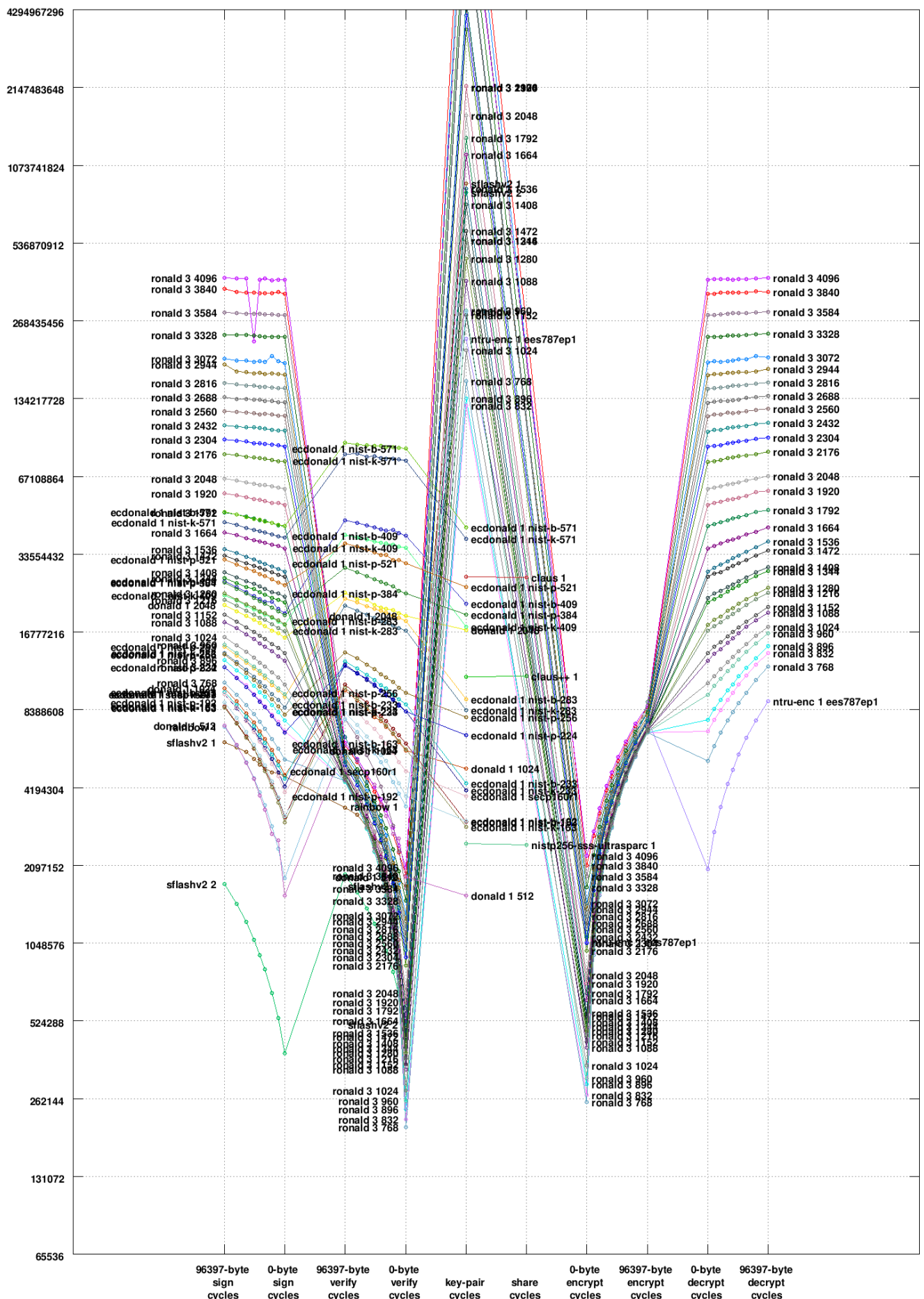
Part of the eBATS project

(ECRYPT Benchmarking
of Asymmetric Systems):

www.ecrypt.eu.org/ebats

... in ECRYPT's VAMPIRE lab:

www.ecrypt.eu.org



donald 1024 is software
that uses the OpenSSL library
to implement 1024-bit DSA.

Some space measurements
for donald 1024:

148 bytes in secret key.

128 bytes in public key.

40-byte overhead
to sign 23-byte message.

40-byte overhead
to sign 147-byte message.

40-byte overhead
to sign 709-byte message.

Timings of donald 1024 on
a 2137MHz Intel Core 2 Duo,
katana, in 64-bit mode:

1041400 cycles

to sign a 59-byte message.

(This is median of many
successive measurements:

1034320 1034360 1038944

1044200 1040432 1041920

1041400 1045296 1049856 etc.)

1246312 cycles

to verify the signature.

1084424 cycles

to generate a key pair.

Timings of donald 1024 on
an 800MHz Intel Pentium M 6d8,
atlas, in 32-bit mode:

2778564 cycles
to sign a 59-byte message.

3329580 cycles
to verify the signature.

(3303976 3336154 3298904
3297910 3892409 3356303
3301620 3298139 3319748 etc.

Note the 3892409 jump.)

2939180 cycles
to generate a key pair.

And: amd64 AMD Athlon 64 X2;
amd64 AMD Opteron 250; same;
amd64 Intel Pentium 4 f43;
ia64 HP Itanium II; same;
ppc32 Motorola PowerPC G4;
sparcv9 Sun UltraSPARC IV;
x86 Intel Pentium 52c, 133MHz!;
x86 AMD Athlon 622;
x86 Intel Pentium III 68a;
x86 Intel Pentium III 6b1; same;
x86 Intel Pentium 4 f12;
x86 Intel Xeon f25; same;
x86 Intel Pentium 4 f26;
x86 Intel Pentium 4 f29; same;
x86 Intel Pentium 4 f41.

Signing	793647	Itanium II
59 bytes	842993	Athlon 64 X2
with	1041400	Core 2 Duo
donald	2176845	P4 f43
1024	2778564	PM 6d8
	2990808	Athlon 622
	3626841	PIII 6b1
	3729408	PIII 68a
	4185764	P4 f26
	4252822	P4 f41
	4411412	Xeon f25
	4665480	P4 f12
	4727730	USPARC IV
	5704242	P1 52c
	5732928	PowerPC G4

Signing	1805770	Itanium II
59 bytes	2387880	Athlon 64 X2
with	3207200	Core 2 Duo
donald	6861675	P4 f43
2048	8775306	PM 6d8
	9727110	Athlon 622
	11942839	PIII 6b1
	12023567	PIII 68a
	12953340	P4 f41
	13145028	P4 f26
	13279464	Xeon f25
	14527024	P4 f12
	15906325	USPARC IV
	18835618	P1 52c
	20117584	PowerPC G4

ecdona1d uses OpenSSL
to implement ECDSA
with various elliptic curves.

secp160r1: 60-byte secret,
40-byte public, 40-byte overhead.

nist-k-163: 63, 42, 42. And b.

nist-p-192: 72, 48, 48.

nist-p-224: 84, 56, 56.

nist-k-233: 90, 60, 60.

nist-p-256: 96, 64, 64.

nist-k-283: 108, 72, 72.

nist-p-384: 144, 96, 96.

nist-k-409: 156, 104, 104.

nist-p-521: 198, 132, 132.

nist-k-571: 216, 144, 144.

Signing	1539854	Athlon 64 X2
59 bytes	1564616	Core 2 Duo
with	2078855	Itanium II
ec	2843992	P4 f43
donald	3075466	PM 6d8
nist-	3235785	USPARC IV
p-192	3341610	Athlon 622
	3444960	PowerPC G4
	3887873	PIII 6b1
	4275010	PIII 68a
	4501108	P4 f26
	4517456	Xeon f25
	4619136	P4 f12
	5277690	P4 f41
	8934562	P1 52c

Verification on Pentium 4 f41:

1883115 donald 512

5226810 donald 1024

6292440 ecdonald nist-p-192

6465570 ecdonald secp160r1

8614845 ecdonald nist-p-224

10061108 ecdonald nist-k-163

10968060 ecdonald nist-b-163

11182065 ecdonald nist-p-256

15582517 donald 2048

19568820 ecdonald nist-k-233

21496155 ecdonald nist-b-233

25563608 ecdonald nist-p-384

35552528 ecdonald nist-k-283

39450120 ecdonald nist-b-283

But wait, there's more!

ronald: RSA signatures.

sflashv2, contributed by

Goubin/Courtois/Icart:

SFLASHv2 MQ signatures.

rainbow, contributed by

Ding/Schmidt:

Rainbow MQ signatures.

bls, contributed by Scott:

pairing-based short signatures.

Many different parameters.

Total: 110 signature systems.

Let's compare a few.

Verification on Pentium 4 f29:

216488 ronald 1024

421428 sflashv2

535888 ronald 2048

1643700 ronald 4096

2939944 rainbow

5108824 donald 1024

5621296 ecdonald p-192

9692816 ecdonald p-256

15037960 donald 2048

30479432 bls

Signing, same CPU:

327292 sflashv2

1254616 rainbow

2498280 bls

4212528 donald 1024

4527152 ecdonald p-192

8021896 ecdonald p-256

11041108 ronald 1024

12417592 donald 2048

57801612 ronald 2048

318750812 ronald 4096

Key-pair generation, same CPU:

4380304 donald 1024

4410428 ecdonald p-192

7819568 ecdonald p-256

14435940 donald 2048

20060932 bls

226258044 ronald 1024

275282008 rainbow

447858256 sflashv2

2273511680 ronald 2048

23087093020 ronald 4096

Key bytes and (23, 709) overhead:

72	48	48	48	ecd	p-192
96	64	64	64	ecd	p-256
20	120	20	20	bls	
148	128	40	40	donald	1024
1024	128	105	43	ronald	1024
276	256	40	40	donald	2048
2048	256	233	43	ronald	2048
4096	512	489	43	ronald	4096
2823	19266	37	37	sflashv2	
20107	31680	43	43	rainbow	

Note the 43-byte overhead for ronald 4096 to sign 709-byte message. “Message recovery.”

But wait, there's more!

`claus` and `claus++`:

Classic DH mod a 1024-bit prime,
using OpenSSL and GMP/NTL.

`curve25519-gaudry`, contributed
by Gaudry: ECDH mod $2^{255} - 19$.

`nistp256-sss-ultrasparc`,
contributed by Nawaz/Gong:
ECDH using NIST P-256.

`ntru-enc`, contributed by Etzel.

`surf127eps`, contributed by
Gaudry/Houtmann/Thomé:
genus-2 HECDH mod $2^{127} - 735$.

DH cycles on Pentium 4 f29:

3105704 surf127eps

3119684 curve25519-gaudry

13644364 claus++

24422484 claus

DH cycles on Core 2 Duo:

591504 curve25519-gaudry

658128 surf127eps

4555240 claus++

5734944 claus

DH cycles on UltraSPARC:

2537798 nistp256-sss

11237085 claus++

27377275 claus

Complete database of
time/space/etc. measurements
is available online
in a documented format
designed for easy parsing.

Many signature systems,
encryption systems, DH;
many message lengths;
22 different machines;
many successive measurements;
3550495 lines overall.
80 megabytes compressed.

Collecting the measurements

Built an API for BATs
(Benchmarkable Asymmetric
Tools) such as `donald`.

API specifies functions such as
`keypair` for BATs to implement.

API was designed to minimize
effort required to write a BAT.
Examples: some BATs handle
long messages; some BATs insist
on short messages; some BATs
are parametrized; some BATs
insist on specific sizes; BATs can
assume that GMP is available.

Built BATMAN (Benchmarking of Asymmetric Tools on Multiple Architectures, Non-interactively) software that measures BATs.

BATMAN tries (e.g.)
donald 1024 on katana
under many compilers,
selects the best compiler
for donald 1024 on katana,
and measures donald 1024
on katana with that compiler.

BAT can specify
multiple “tunings”
to try with each compiler.

To save time,
if a compiler flunks
some simple tests on katana,
BATMAN skips the compiler
for all BATs on katana.

Median of cycle counts is
much more stable than average.
Allows very fast measurements.

Collecting complete database
took tolerable amount
of CPU time:

27819 seconds on katana,
548297 seconds on a
533MHz PowerPC G4, etc.

Why measure many machines?

Performance of cryptography is heavily influenced by CPU (and other machine features).

Switching from μs to cycles reduces CPU dependence but does not eliminate it.

Paper 1: “ X : 20000 P4 cycles.”

Paper 2: “ Y : 15% faster, 17000 PM cycles.” Often Y is slower than X on *both* P4 and PM!

Sometimes Y is faster than X on one CPU but slower on another.

How VAMPIRE adds CPUs:

1. Find machine with usable OS, including reasonable compiler.

Surely we can share?

2. Port GMP/NTL/OpenSSL.

Sometimes quite difficult.

Need better config scripts.

3. Set up a BATMAN account.

4. Port BATMAN. Usually easy.

5. Run BATMAN. Very easy.

(Can imagine fully automatic timing of subsequent BATs in appropriate sandboxes.)

Security evaluations

Do users want the
smallest, fastest cryptosystems?
Not exactly.

Users want the
smallest, fastest cryptosystems
*that provide an
acceptable security level.*

Can reduce time and space
by reducing security level.

Example: donald 512
is faster than donald 1024.

eBATS API allows BAT to state its conjectured security level.

Much harder to verify than time/space measurements, but still extremely important for users.

VAMPIRE plans to highlight security-aware comparisons in the next eBATS report.

Question: Should BATtacks be separated from BATs, allowing cryptanalysts to submit separate declarations?

More BATs; faster BATs

Existing BATs cover
many public-key systems.

Often state-of-the-art software.

Thanks for all the contributions!

Implement additional systems?

e.g. more post-quantum systems?

Speed up existing systems?

Modify systems to save space?

Protect against timing attacks
and other side-channel attacks?

VAMPIRE is developing
more BATs . . . and so are you!
e.g. mceliece-1 from Sendrier.

Some BAT speedups
will need API extensions:

- Signers with state:
e.g., Merkle hash trees.
- Batch operations:
e.g., batch DSA verification.
- (k, m, s) compression:
e.g., Bleichenbacher's
vanishing RSA.

The eBATS competition:
Build the most efficient
public-key software.

Speed improvements should be
easy to express in BATs.

If not, let us know!

Space improvements should be
easy to express in BATs.

If not, let us know!

All improvements should be
visible in the eBATS results.

If not, let us know!

More types of cryptography

ciphercycles: toolkit for benchmarking secret-key authenticated encryption.

Heavy reuse of BATMAN structure and software.

Results presented at SASC 2007.

Identity-based encryption?

Hash functions?

Merge everything into

ECRYPT VAMPIRE Grand

Unified Cryptographic-Primitive

Benchmarking Toolkit? Need

better acronym than EVGUCPBT.

Timeline

2005.08: D.VAM.1,
survey of operation counts.

2006.02: D.VAM.7,
initial plans for eBATS.

2007.03: D.VAM.9,
comprehensive report on
first-stage measurements.

2007.06.15: **Submit more BATs!**

2007.07: Second report.

2007.12: Third report.

2008.05: Fourth report.