eBACS: ECRYPT Benchmarking of Cryptographic Systems

http://bench.cr.yp.to

D. J. Bernstein
University of Illinois at Chicago

Joint work with:
Tanja Lange
Technische Universiteit Eindhoven

European Union has funded
NESSIE project (2000–2003),
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

NESSIE's performance evaluators
tuned C implementations
of many cryptographic systems,
all supporting the same API;
wrote a benchmarking toolkit;
ran the toolkit on 25 computers.

Many specific performance results:
e.g., 24 cycles/byte on P4
for 128-bit AES encryption.

ECRYPT I had five “virtual labs.”
STVL, symmetric-techniques lab, included four working groups.
STVL WG 1, stream-cipher group, ran eSTREAM (2004–2008).

De Cannière *published* eSTREAM benchmarking toolkit.

Stream-cipher implementations matching the benchmarking API were contributed by designers, *published*, often tuned; benchmarked on many computers.

e.g. 18 cycles/byte on P4 for third-party asm AES in toolkit.

2006: VAMPIRE, “Virtual Application and Implementation Lab,” started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 46 public-key implementations matching the benchmarking API. Benchmarked on many computers.

2008: VAMPIRE started eBASC ("ECRYPT Benchmarking of Stream Ciphers") for post-eSTREAM benchmarks.

VAMPIRE also started eBASH ("ECRYPT Benchmarking of All Submitted Hashes").

eBACS ("ECRYPT Benchmarking of Cryptographic Systems") includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

eBASH → public

eBASH has already collected
51 implementations of
28 hash functions in 14 families.

`http://bench.cr.yp.to`
`/results-hash.html`
already shows
measurements on 69 machines;
95 machine-ABI combinations.

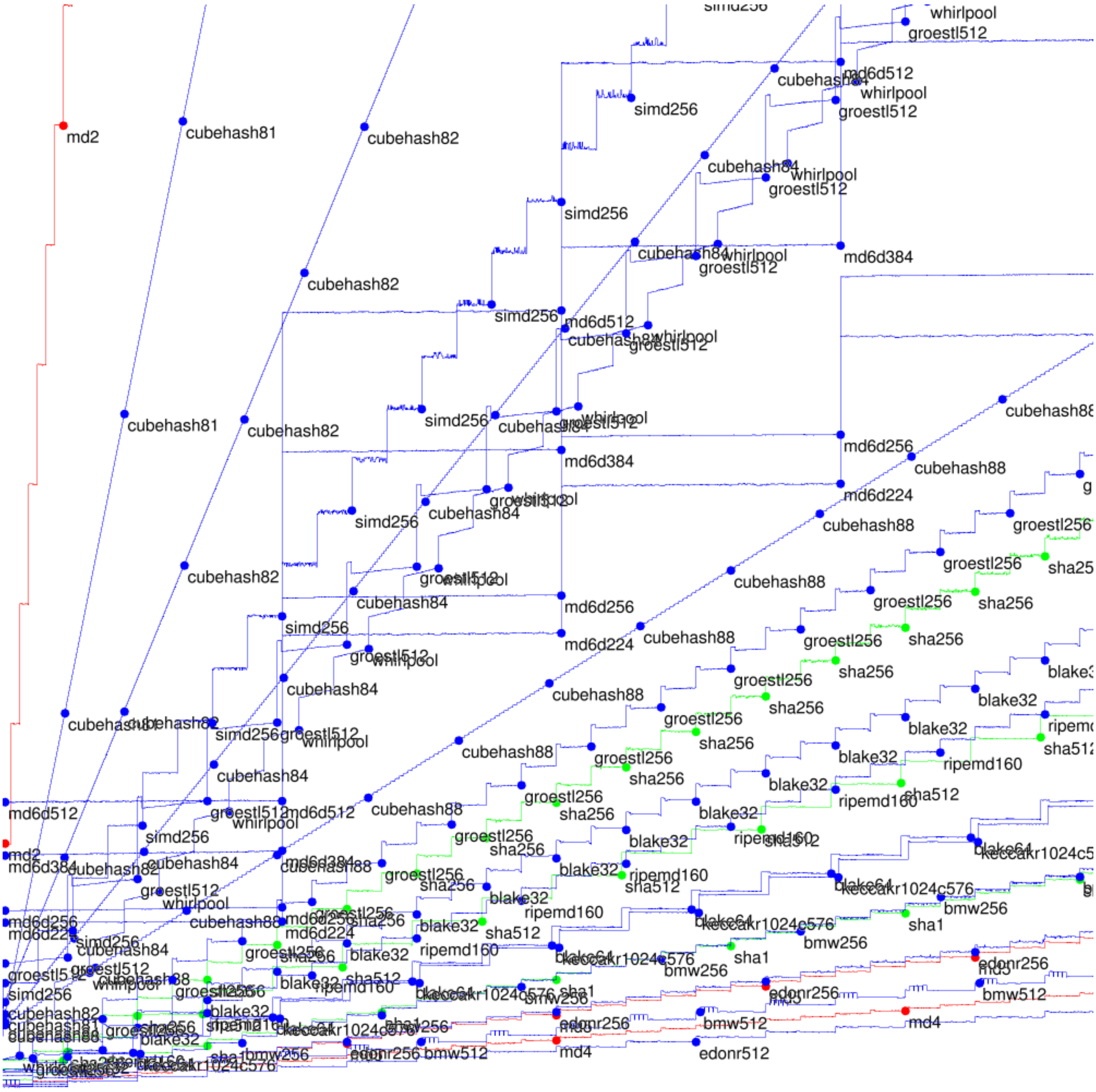
Each implementation is
recompiled 1201 times
with various compiler options
to identify best working option
for implementation, machine.

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

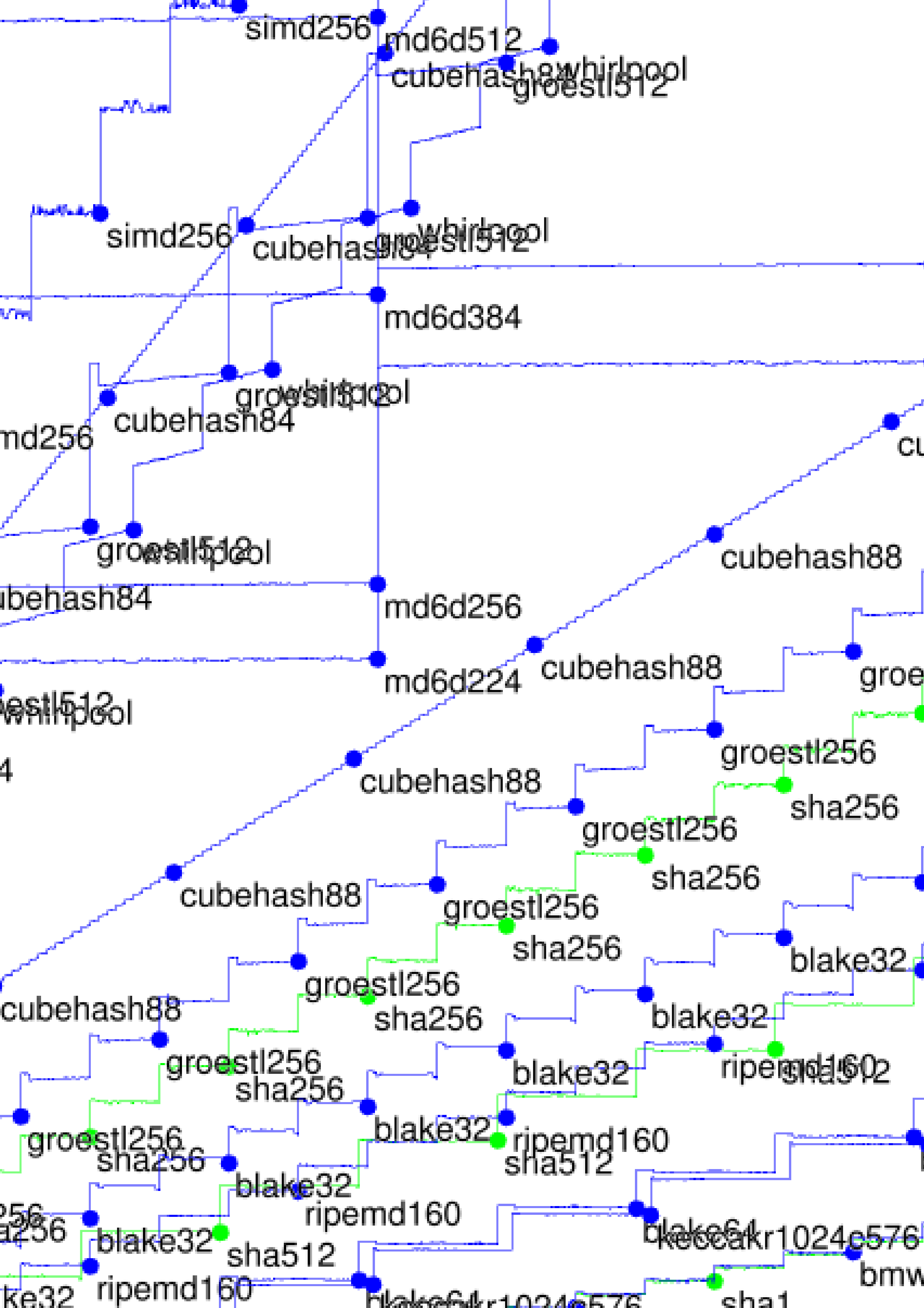
25%	50%	75%	hash
3.53	3.54	3.56	edonr512
4.89	4.89	4.90	bmw512
6.51	6.53	6.53	md5
6.68	6.68	6.69	edonr256
9.18	9.21	9.22	bmw256
9.51	9.53	9.58	sha1
11.94	11.94	11.97	keccakr1024c576
12.22	12.24	12.26	blake64
16.21	16.25	16.25	sha512
16.81	16.81	16.82	ripemd160
19.44	19.46	19.46	blake32
23.57	23.67	23.72	sha256
26.83	26.85	26.86	groestl256
			etc.

Tables show medians, quartiles
of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.



md2
cubehash81
cubehash82
cubehash84
cubehash88
simd256
whirlpool
groestl512
groestl256
md6d512
md6d384
md6d256
md6d224
sha256
sha512
sha1
blake32
blake64
ripemd160
keccakr1024c576
bmw256
bmw512
edonr256
edonr512
md4
md5



simd256
md6d512
groestl512
whirlpool
simd256
md6d384
cubehash84
md6d256
md6d224
cubehash88
cubehash88
cubehash88
cubehash88
cubehash88
groestl256
sha256
groestl256
sha256
groestl256
sha256
groestl256
sha256
groestl256
sha256
groestl256
sha256
blake32
blake32
blake32
blake32
blake32
blake32
ripemd160
sha512
ripemd160
ripemd160
sha512
ripemd160
sha512
sha1

Submitter $\rightarrow$ eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Define hash function in `hash.c`,
e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
        ,in,inlen*8,out);
  return 0; }
```

Send to the mailing list
the URL of a `tar.gz`
with one directory
`crypto_hash/yourhash/ref`
containing `hash.c` etc.

Measurements magically appear!
Much easier than trying
to do your own benchmarks.

More details and options:
http://bench.cr.yp.to
/call-hash.html

Also easy for third parties
to run the benchmark suite.