

eBASH:

ECRYPT Benchmarking
of All Submitted Hashes

[http://bench.cr.yp.to
/ebash.html](http://bench.cr.yp.to/ebash.html)

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 49 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
77 implementations of
38 hash functions in 18 families.

[http://bench.cr.yp.to
/results-hash.html](http://bench.cr.yp.to/results-hash.html)

already shows

measurements on 71 machines;
101 machine-ABI combinations.

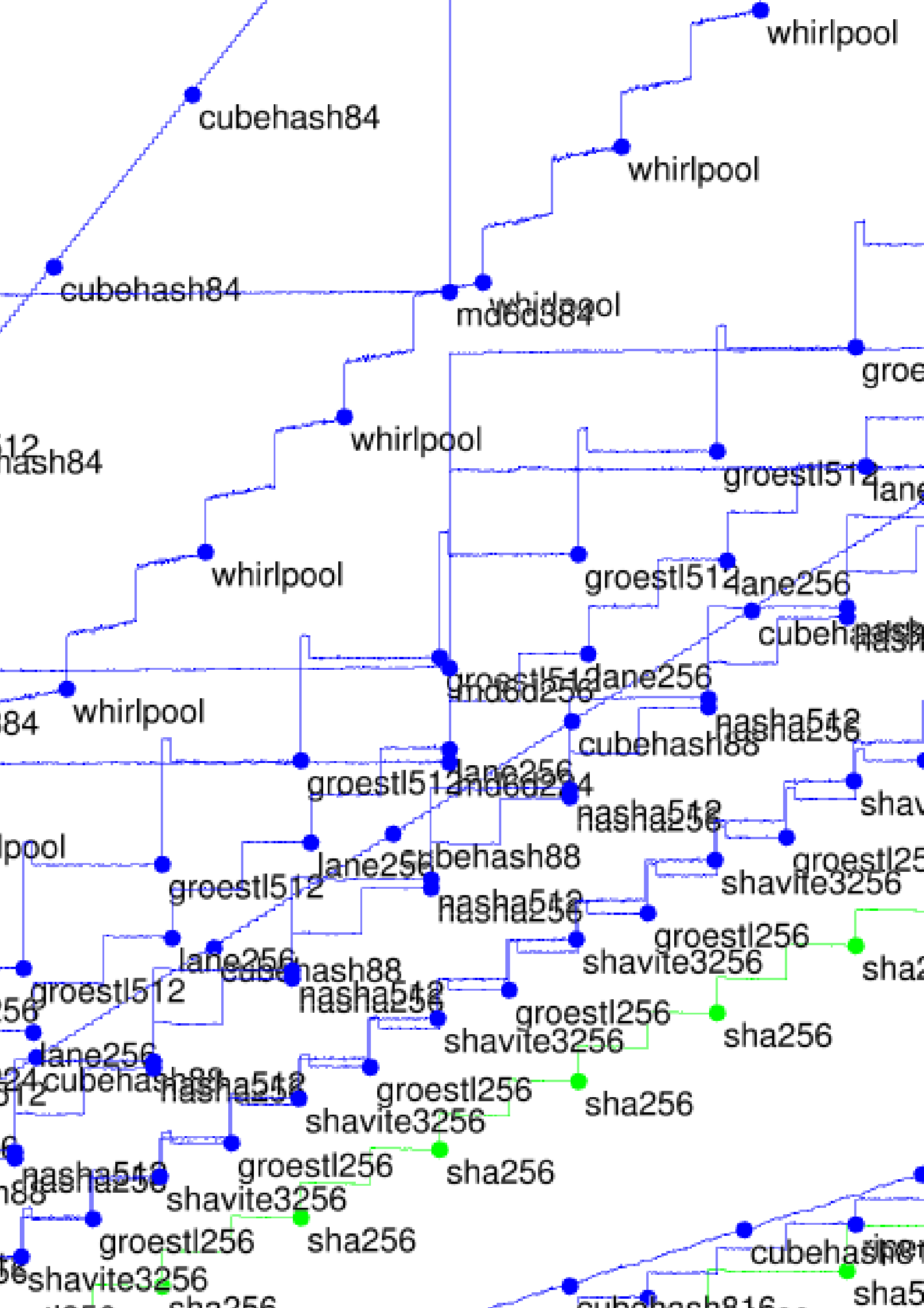
Each implementation is
recompiled 1226 times
with various compiler options
to identify best working option
for implementation, machine.

e.g. 1536 bytes, katana (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			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.



Submitter → 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](http://bench.cr.yp.to/call-hash.html)

Also easy for third parties
to run the benchmark suite.