eBASH: ECRYPT Benchmarking of All Submitted Hashes

http://bench.cr.yp.to/ebash.html

D. J. Bernstein
University of Illinois at Chicago

Joint work with:
Tanja Lange
Technische Universiteit Eindhoven

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 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:

<table>
<thead>
<tr>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>edonr512</td>
</tr>
<tr>
<td>4.46</td>
<td>4.46</td>
<td>4.46</td>
<td>bmw512</td>
</tr>
<tr>
<td>5.29</td>
<td>5.30</td>
<td>5.38</td>
<td>edonr256</td>
</tr>
<tr>
<td>7.08</td>
<td>7.08</td>
<td>7.08</td>
<td>skein512</td>
</tr>
<tr>
<td>8.29</td>
<td>8.30</td>
<td>8.30</td>
<td>sha1</td>
</tr>
<tr>
<td>8.39</td>
<td>8.39</td>
<td>8.47</td>
<td>bmw256</td>
</tr>
<tr>
<td>9.59</td>
<td>9.59</td>
<td>9.60</td>
<td>cubehash832</td>
</tr>
<tr>
<td>9.67</td>
<td>9.76</td>
<td>9.76</td>
<td>shabal512</td>
</tr>
<tr>
<td>11.29</td>
<td>11.30</td>
<td>11.30</td>
<td>keccak\text{r1024c576}</td>
</tr>
<tr>
<td>11.47</td>
<td>11.49</td>
<td>11.54</td>
<td>simd256</td>
</tr>
<tr>
<td>12.08</td>
<td>12.08</td>
<td>12.08</td>
<td>blake64</td>
</tr>
<tr>
<td>12.05</td>
<td>12.09</td>
<td>12.09</td>
<td>blake32</td>
</tr>
<tr>
<td>14.83</td>
<td>14.83</td>
<td>14.85</td>
<td>sha512</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>
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.
Define output size in api.h:

#define CRYPTOgetBytes 64
Define output size in api.h:

```c
#define CRYPTO_BYTES 64
```

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

```c
#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.