CubeHash efficiency estimates (2.B.2)

Daniel J. Bernstein *

Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045
cubehash@box.cr.yp.to

This document is a statement of CubeHash's estimated computational efficiency and memory requirements in hardware and software across a variety of platforms.

General comments. CubeHashr/b xors a b-byte input block to the first b bytes of its 1024-bit state, applies an invertible transformation consisting of r identical rounds, and then moves on to the next input block. Consequently CubeHashr/b uses r rounds for each b-byte block; the time taken by CubeHashr/b is roughly linear in r/b.

CubeHashr/b–h has an initial overhead of $10r$ rounds to build an initial state given the round count r, the block size b, and the output length h. This overhead can be trivially eliminated at the expense of 128 bytes of constant storage for each desired (r, b, h).

CubeHashr/b–h also has a final overhead of $10r$ rounds, approximately the cost of hashing an additional $10b$ bytes. For comparison, SHA–256 uses between 8 and 72 bytes of padding, and SHA–512 uses between 16 and 144 bytes of padding.

Efficiency estimates for the NIST SHA–3 reference platform in 64-bit mode. NIST has described its SHA–3 reference platform as follows: Wintel personal computer, with an Intel Core 2 Duo Processor, 2.4GHz clock speed, 2GB RAM, running Windows Vista Ultimate 64-bit (x64) Edition, using the ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

The original CubeHash submission generated efficiency estimates from the following calculation: The Core 2 Duo can perform three simple 128-bit vector operations in each cycle. Adding one vector of four 32-bit words to another is a single vector operation, so the 32 word additions in a CubeHash round are eight vector operations, estimated to take 8/3 cycles. Similar comments apply to xor. The 32 word rotations can be implemented as eight copies, sixteen shifts, and eight xors, estimated to take 32/3 cycles. An additional 8/3 cycles are required for eight vector permutations. Together with loop overhead each round is therefore *estimated* to take 20 cycles.

Estimated clock cycles for a 512-bit message digest: $200r$ cycles to compute the initial state; $20r$ cycles for each b-byte message block; $200r$ cycles to compute the digest. In other words: $20r/b$ cycles per byte, with an overhead of $20b$ bytes.

* The author was supported by the National Science Foundation under grant ITR–0716498. Date of this document: 2009.09.14.

Estimated clock cycles for a 384-bit message digest: Same.

Estimated clock cycles for a 256-bit message digest: Same.

Estimated clock cycles for a 224-bit message digest: Same.

Time-memory tradeoffs: The initial state can be precomputed, as discussed above, eliminating $10b$ bytes of overhead.

Publicly verifiable speed measurements have been collected and published by the eBASH (ECRYPT Benchmarking of All Submitted Hashes) project. In particular, eBASH `supercop-20090702` measured CubeHash16/32 as running at 11.47 cycles/byte on an Intel Core 2 Duo E8400 1067a (`brick`), and at 12.66 cycles/byte on an Intel Core 2 Duo 6f6 (`katana`). For comparison, SHA-512 runs at 12.60 cycles/byte on `brick`, and SHA-256 runs at 15.53 cycles/byte.

Note that Intel has sold several different CPUs under the name "2.4GHz Core 2 Duo": the 2.4GHz Core 2 Duo E4600 (Allendale microarchitecture), the 2.4GHz Core 2 Duo E6600 (Conroe), the 2.4GHz Core 2 Duo E8135 (Penryn), the 2.4GHz Core 2 Duo P8600 (Penryn), the 2.4GHz Core 2 Duo SP9400 (Penryn), the 2.4GHz Core 2 Duo T7700 (Merom), and the 2.4GHz Core 2 Duo T8300 (Penryn). NIST should specify which of these CPUs is in fact the SHA–3 reference platform.

Efficiency estimates for the NIST SHA–3 reference platform in 32-bit mode. Description of platform used to generate the estimates: Wintel personal computer, with an Intel Core 2 Duo Processor, 2.4GHz clock speed, 2GB RAM, running Windows Vista Ultimate 32-bit (x86) Edition, using the ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

The original CubeHash submission generated efficiency estimates from the following calculation: See above for a description of the Core 2 Duo's vector capabilities. Only 8 128-bit vector registers are available in 32-bit mode; this might seem to be enough for CubeHash, but rotation uses an extra register, so some extra loads and stores are required. Each round is therefore *estimated* to take 25 cycles.

Estimated clock cycles for a 512-bit message digest: $250r$ cycles to compute the initial state; $25r$ cycles for each b-byte message block; $250r$ cycles to compute the digest. In other words, $25r/b$ cycles per byte, with an overhead of $20b$ bytes.

Estimated clock cycles for a 384-bit message digest: Same.

Estimated clock cycles for a 256-bit message digest: Same.

Estimated clock cycles for a 224-bit message digest: Same.

Time-memory tradeoffs: The initial state can be precomputed, as discussed above, eliminating $10b$ bytes of overhead.

eBASH `supercop-20090702` measured CubeHash16/32 as running at 12.74 cycles/byte on an Intel Core 2 Duo E8400 1067a (`brick`) in 32-bit mode, and at 14.07 cycles/byte on an Intel Core 2 Duo 6f6 (`katana`) in 32-bit mode. For comparison, SHA-512 runs at 17.76 cycles/byte on `brick` in 32-bit mode, and SHA-256 runs at 15.33 cycles/byte in 32-bit mode.

Efficiency estimates for 8-bit processors. Description of platform used to generate the estimates: Atmel ATmega8, 16MHz clock speed, 1024 bytes of static RAM.

The original CubeHash submission estimated an ATmega8 round to take roughly 2000 cycles. This estimate was derived from the following comparison to the Core 2 Duo: a Core 2 Duo cycle performs 12 32-bit operations, typically equivalent to 48 8-bit operations, each of which takes 1 cycle on the ATmega8; loads and stores are estimated to cost an extra factor of approximately 2. Note that the CubeHash state fits comfortably into the static RAM on the ATmega8, and that the hypercube structure in CubeHash permits some locality of reference.

Estimated clock cycles for a 512-bit message digest: $20000r$ cycles to compute the initial state; $2000r$ cycles for each b-byte message block; $20000r$ cycles to compute the digest. In other words: $2000r/b$ cycles per byte, with an overhead of $20b$ bytes.

Estimated clock cycles for a 384-bit message digest: Same.

Estimated clock cycles for a 256-bit message digest: Same.

Estimated clock cycles for a 224-bit message digest: Same.

Time-memory tradeoffs: The initial state can be precomputed, as discussed above, eliminating $10b$ bytes of overhead.

Efficiency estimates for ASICs. Description of platform used to generate the estimates: ASIC built with a standard 130nm process.

Estimates were generated from calculations shown below.

Estimated gate count: A fully unrolled, purely combinatorial round uses 1024 full adders (each 10 gate equivalents) and 1024 xors (each 4 gate equivalents), plus 1024 D-type flip-flops (each 8 gate equivalents) to store the resulting state, for a total of approximately 23000 gate equivalents. There are several ways that this area can be improved without sacrificing speed, for example using the Ling adder. (On the other hand, to improve clock speed one might also consider using more gates to reduce adder latency.) The benefit of full unrolling is that each round takes just one clock cycle.

Estimated clock cycles for a 512-bit message digest: $10r$ cycles to compute the initial state; r cycles for each b-byte message block; $10r$ cycles to compute the digest. In other words: r/b cycles per byte, with an overhead of $20b$ bytes.

Estimated clock cycles for a 384-bit message digest: Same.

Estimated clock cycles for a 256-bit message digest: Same.

Estimated clock cycles for a 224-bit message digest: Same.

Time-memory tradeoffs: The parallelism and regular structure of CubeHash allow a wide variety of choices for the hardware implementor.