CubeHash features (2.B.6)

Daniel J. Bernstein *

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

This statement lists and describes the advantages and limitations of the CubeHash family of hash functions. This statement is not meant to exclude the possibility of further advantages and limitations being discovered during the SHA–3 candidate evaluation process.

Small area requirements. CubeHashr/b xors b bytes of input into the first b bytes of a 128-byte state. It then modifies the state in place and moves on to the next b bytes of input. This 128-byte state is small enough to fit into software environments having very little memory. The modifications are simple and regular, so CubeHash can also fit into small area on an FPGA or ASIC.

For comparison, although SHA–256 can store its state *between* blocks in just 32 bytes (plus 8 bytes for a message-length counter), SHA–256 needs at least 128 bytes to process a block: 32 bytes for the current state, 32 bytes for the beginning-of-block state, and 64 bytes for the current segment of the message schedule. SHA–512 needs at least 256 bytes.

Unified implementation across output sizes. A circuit that implements both CubeHashr/b–512 and CubeHashr/b–256, and runs either one at the same speed, is only marginally more expensive than a circuit that implements just CubeHashr/b–256.

Most hash functions become two or three times larger in the 256-and-512-area metric: they require double-size states, separate control circuits, etc. One can expect most hardware implementations to be limited to the smaller size, the same way that most SHA–2 hardware implementations are limited to SHA-256, posing problems for users considering a switch to a longer output size. CubeHash does not have this problem.

Small code size and vector-code size. CubeHash fits into a very small amount of instruction space. Optimized CubeHash implementations using vector instructions are also quite small. This feature is useful not only for small devices but also for busy Internet servers: a CPU has a limited amount of space in its instruction cache, and becomes much slower if frequently used instructions do not fit into that cache.

Parallelizability. Each step of CubeHash consists of 16 independent operations on 32-bit words. The operations are parallelizable and vectorizable, providing

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

tremendous flexibility for the implementor and allowing CubeHash to run at high speeds on a wide variety of computer architectures.

CubeHash does *not* have the sort of global-scale parallelism that would be provided by a message-length tree of block hashes. The advantage of a tree is that it allows very long messages to be split across several processor cores—but the applications that care can achieve the same benefit by building a tree on top of CubeHash, the same way that they have traditionally built a tree on top of SHA–256. (Similar comments apply to incremental hashing etc.) The disadvantage of a tree is that it cannot be implemented in low area.

Other message-digest sizes. CubeHash supports h-bit output lengths for every $h \in \{8, 16, 24, \ldots, 512\}$. In particular, CubeHash supports the required output lengths of 224 bits (28 bytes), 256 bits (32 bytes), 384 bits (48 bytes), and 512 bits (64 bytes). Note that some applications do not need collision resistance and are satisfied with output lengths significantly below 256 bits.

Good security/speed tradeoff. Extensive third-party analysis has culminated in a collision attack against CubeHash7/64–512 estimated to take "only" 2^{203} operations. To put this in perspective, CubeHash7/64 and CubeHash8/64 are considerably faster than MD5 on the NIST reference platform. It is clear that the CubeHash design achieves very high security at low cost.

Perhaps CubeHash8/64 will be broken someday, but there is a massive safety margin between CubeHash8/64 and the original CubeHash8/1, and there is a very comfortable safety margin between CubeHash8/64 and CubeHash16/32.