Inside the Hypercube

Jean-Philippe Aumasson[1*], Willi Meier[1†], María Naya-Plasencia[2‡], and Thomas Peyrin[3]

[1] FHNW, Windisch, Switzerland
[2] INRIA project-team SECRET, France
[3] Ingenico, France

Some force inside the Hypercube occasionally manifests itself with deadly results.
`http://www.staticzombie.com/2003/06/cube_2_hypercube.html`

Abstract. Bernstein's CubeHash is a hash function family that includes four functions submitted to the NIST Hash Competition. A CubeHash function is parametrized by a number of rounds r, a block byte size b, and a digest bit length h (the compression function makes r rounds, while the finalization function makes $10r$ rounds). The 1024-bit internal state of CubeHash is represented as a five-dimensional hypercube. Submissions to NIST have $r = 8$, $b = 1$, and $h \in \{224, 256, 384, 512\}$.
This paper gives the first external analysis of CubeHash, with

- improved standard generic attacks for collisions and preimages
- a multicollision attack that exploits fixed points
- a study of the round function symmetries
- a preimage attack that exploits these symmetries
- a practical collision attack on a weakened version of CubeHash
- high-probability truncated differentials over the 10-round transform
- an example of collision on CubeHash2/120-512

Our results do not seem to contradict the security claims about CubeHash.

1 CubeHash

Bernstein's CubeHash is a hash function family that includes four functions submitted to the NIST Hash Competition. A CubeHash function is parametrized by a number of rounds r, a block byte size b, and a digest bit length h; the 1024-bit internal state of CubeHash is viewed as a five dimensional hypercube. Submissions to NIST have $r = 8$, $b = 1$, and $h \in \{224, 256, 384, 512\}$.

CubeHash computes the digest of a message as follows:

- initialize a 1024-bit state as a function of (h, b, r)
- append to the message a 1 bit and enough 0 bits to reach a multiple of $8b$ bits
- for each b-byte message block:
  - xor the block into the first b bytes of the state
  - transform the state through the r-round T function
- xor a 1 bit with the 993th bit of the state
- transform the state through $10r$-round T
- output the first h bits of the state

*Supported by the Swiss National Science Foundation under project no. 113329.
†Supported by GEBERT RÜF STIFTUNG, project no. GRS-069/07.
‡Supported in part by the French Agence Nationale de la Recherche under contract ANR-06-SETI-013-RAPIDE.

Let $x[0], \ldots, x[31]$ represent the 1024-bit state as an array of 32-bit words. The transform function T makes r identical rounds, where each round computes (see also Fig. 1):

$$
\begin{array}{ll}
\textbf{for } i = 0, \ldots, 15: & x[i+16] = x[i+16] + x[i] \\
\textbf{for } i = 0, \ldots, 15: & y[i \oplus 8] = x[i] \\
\textbf{for } i = 0, \ldots, 15: & x[i] = y[i] \lll 7 \\
\textbf{for } i = 0, \ldots, 15: & x[i] = x[i] \oplus x[i+16] \\
\textbf{for } i = 0, \ldots, 15: & y[i \oplus 2] = x[i+16] \\
\textbf{for } i = 0, \ldots, 15: & x[i+16] = y[i] \\
\textbf{for } i = 0, \ldots, 15: & x[i+16] = x[i+16] + x[i] \\
\textbf{for } i = 0, \ldots, 15: & y[i \oplus 4] = x[i] \\
\textbf{for } i = 0, \ldots, 15: & x[i] = y[i] \lll 11 \\
\textbf{for } i = 0, \ldots, 15: & x[i] = x[i] \oplus x[i+16] \\
\textbf{for } i = 0, \ldots, 15: & y[i \oplus 1] = x[i+16] \\
\textbf{for } i = 0, \ldots, 15: & x[i+16] = y[i]
\end{array}
$$

2 Improved standard generic attacks

The author of CubeHash presented [3] the following "standard preimage attack":

- from (h, b, r) compute the initial state S_0
- from the h-bit image plus some arbitrary $/1024 - h)$ bits, invert $10r$ rounds and the "xor 1" to get a state S_f before finalization
- find two n-block sequences that map S_0 (forward) and S_f (backward), respectively, to two states that share the last $(1024 - 8b)$ bits

There are 2^{nb} possible n-block inputs and one looks for a collision over $(1024-8b)$ bits. For a success chance $1 - 1/e \approx 0.63$ one thus requires 2^{512-4b} trials in each direction, that is, $2nb > 1024-8b$, i.e., $n > 512/b-4$. In total the number of evaluations of T is approximately

$$2 \times \left(\frac{512}{b} - 4\right) \times 2^{512-4b} \approx 2^{522-4b-\log b} \ .$$

Furthermore, [3] estimates that each round of T needs 2^{11} "bit operations"; the above formula gives about $2^{533-4b-\log b+\log r}$ bit operations.

A speed-up of the above attack can be obtained by searching a collision not only in the states resulting of a n-block computation, but in every distinct state reached (i.e. also with the intermediate states). This is made possible by the absence of message length padding. Each call to T gives a new candidate for the collision search; we thus get rid of the $(512/b-4)$ multiplicative factor in the cost estimate. This gives a cost of

$$2 \times 2^{512-4b} = 2^{513-4b}$$

evaluations of T, i.e. $2^{524-4b+\log r}$ bit operations.

The proposed CubeHash-512 has $(h, b, r) = (512, 1, 8)$, our attack thus makes 2^{523} bit operations, against 2^{532} with the original attack. If $r = 8$, our attack needs $b > 3$ to make less than 2^{512} bit operations, against $b > 5$ with the original preimage attack. It is to note that these estimates exclude the non-negligible communication costs.

One can use the same trick to speed-up the standard collision attack [3]; the cost in T evaluations then drops from $2^{521-4b-\log b}$ to 2^{512-4b}.

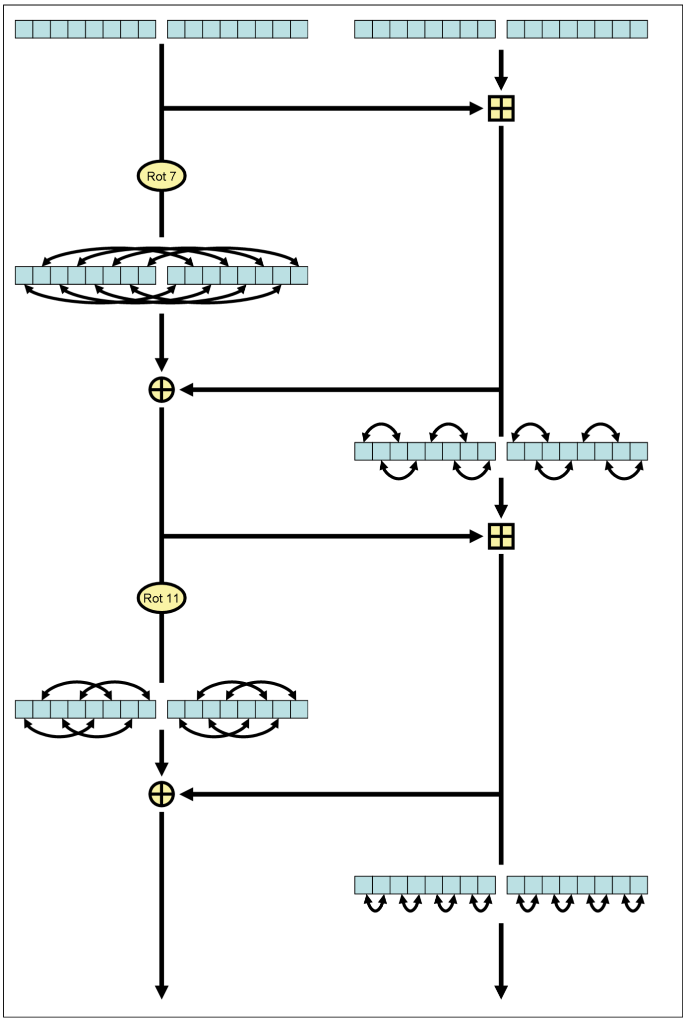

Fig. 1. Schematic view of a CubeHash round.

3 Narrow-pipe multicollisions

Based on the "narrow-pipe" attacks in [4], we show a multicollision attack on CubeHash faster than Joux's [7] or birthday [6,9] methods (for large b's). Our attack requires the same amount of computation as narrow-pipe collisions. It exploits the fact that the null state is a fixed point for the compression function T (regardless of r), and that the message padding doesn't include the message length.

Starting from an initial state S_0 derived from (h, b, r), one finds two n-block sequences m and m' that map S_0 (forward) and the zero state (backward), respectively, to two states that share the last $(1024 - 8b)$ bits. One finds a connection of the form

$$\begin{aligned} S_0 \oplus m_1 &\xrightarrow{T} S_1 \\ S_1 \oplus m_2 &\xrightarrow{T} \cdots \\ &\cdots \\ \cdots &\xrightarrow{T} S_1' \\ S_1' \oplus m_2' &\xrightarrow{T} 0 \oplus m_1' \end{aligned}$$

Once a path to the zero state is found, one can add an arbitrary number of zero message blocks to maintain a zero state. Colliding messages are of the form

$$m \| m' \| 0 \| 0 \| \ldots \| 0 \| \bar{m},$$

where $\bar{m}$ is an arbitrary sequence of blocks.

Using the technique of §2, this multicollision attack requires approximately 2^{513-4b} evaluations of T. In comparison, a birthday attack finds a k-collision in $(k! \times 2^{n(k-1)})^{1/k}$ trials, and Joux's attacks in $\log k \times 2^{4(128-b)}$. For example, with $h = 512$ and $b = 112$, our attack finds 2^{64}-collisions within 2^{65} calls to T, against $> 2^{512}$ for a birthday attack and 2^{70} for Joux's.

4 On state symmetries

The documentation of CubeHash mentions [5, p.3] the existence of symmetries through the round function, and states that the initialization of CubeHash was designed to avoid them. However [5] gives no detail on those symmetries. In this section, we provide a reasoning that finds all symmetries inherent in the transformation T. In total we are able to show 15 symmetry classes of 2^{512} states each, and show how to exploit these.

4.1 Symmetry classes

If a 32-word state x satisfies $x[0] = x[1]$, $x[2] = x[3]$, ..., $x[30] = x[31]$, then this property is preserved through the transformation T, with probability equal to 1, for any number of rounds. One can represent this symmetry with the pattern (each letter stands for a 32-bit word):

`AABBCCDD EEFFGGHH IIJJKKLL MMNNOOPP .`

In total we found 15 classes of symmetry:

$$
\begin{array}{rl}
C_1: & \texttt{AABBCCDD EEFFGGHH IIJJKKLL MMNNOOPP} \\
C_2: & \texttt{ABABCDCD EFEFGHGH IJIJKLKL MNMNOPOP} \\
C_3: & \texttt{ABBACDDC EFFEGHHG IJJIKLLK MNNMOPPO} \\
C_4: & \texttt{ABCDABCD EFGHEFGH IJKLIJKL MNOPMNOP} \\
C_5: & \texttt{ABCDBADC EFGHFEHG IJKLJILK MNOPNMPO} \\
C_6: & \texttt{ABCDCDAB EFGHGHEF IJKLKLIJ MNOPOPMN} \\
C_7: & \texttt{ABCDDCBA EFGHHGFE IJKLLKJI MNOPPONM} \\
C_8: & \texttt{ABCDEFGH ABCDEFGH IJKLMNOP IJKLMNOP} \\
C_9: & \texttt{ABCDEFGH BADCFEHG IJKLMNOP JILKNMPO} \\
C_{10}: & \texttt{ABCDEFGH CDABGHEF IJKLMNOP KLIJOPMN} \\
C_{11}: & \texttt{ABCDEFGH DCBAHGFE IJKLMNOP LKJIPONM} \\
C_{12}: & \texttt{ABCDEFGH EFGHABCD IJKLMNOP MNOPIJKL} \\
C_{13}: & \texttt{ABCDEFGH FEHGBADC IJKLMNOP NMPOJILK} \\
C_{14}: & \texttt{ABCDEFGH GHEFCDAB IJKLMNOP CDABKLIJ} \\
C_{15}: & \texttt{ABCDEFGH HGFEDCBA IJKLMNOP PONMLKJI}
\end{array}
$$

Each class contains 2^{512} states. If a state belongs to several classes, then its image under T also belongs to these classes; for example if $S \in (C_i \cap C_j)$, then $T(S) \in (C_i \cap C_j)$. We have

$$|C_i \cap C_j| \leq 2^{256}.$$

We thus have $\left|\cup_{i=1}^{15} C_i\right| \geq 15 \times 2^{512} - 105 \times 2^{256} \approx 2^{515.9} \approx 2^{516}$ distinct symmetric states. Note that symmetry is not preserved by the finalization procedure of CubeHash (the "xor 1" breaks any of the above symmetries).

4.2 Finding all symmetry classes

Now we prove that the classes $C_1, \ldots, C_{15}$ capture all the possible symmetries of CubeHash's transform T. A symmetry class can be represented as a set of pairs (i, j), where each (i, j) means $x[i] = x[j]$. For example, C_1 can be described by the set

$$
\begin{array}{cccccccc}
(0,1) & (2,3) & (4,5) & (6,7) & (8,9) & (10,11) & (12,13) & (14,15) \\
(16,17) & (18,19) & (20,21) & (22,23) & (24,25) & (26,27) & (28,29) & (30,31)
\end{array}
$$

We want a symmetry class to propagate through one round of the scheme with probability equal to one. It is easy to see that this condition imposes that the equality constraints at the left and at the right branch of the scheme must be the same (because of the intra-word rotations that are only present in the left branch of the scheme). That is, for any relation (i, j) with $0 \leq i, j \leq 15$, we must also have the relation $(i + 16, j + 16)$. In other words, a symmetry pattern is the same for the left and for the right branch. We thus only need to consider 16-word symmetry patterns.

To describe all possible symmetries, we start by fixing $(0, k)$, for a fixed k in $\{1, \ldots, 15\}$. We then compute T backwards to indentify the relations implied by $(0, k)$: the first substitution and xor encountered force us to have

$$(0, k)\ (4, k \oplus 4).$$

Then, the second substitution and the modular addition force to have (note that the intra-word rotations can be omitted since they leave the symmetry pattern unchanged)

$$(0, k)\ (4, k \oplus 4)\ (1, k \oplus 1)\ (5, k \oplus 5).$$

The third substitution and xor yield

$$\begin{array}{cccc}(0,k) & (4,k\oplus 4) & (1,k\oplus 1) & (5,k\oplus 5)\\ (2,k\oplus 2) & (6,k\oplus 6) & (3,k\oplus 3) & (7,k\oplus 7).\end{array}$$

Finally, the last substitution and the modular addition imply

$$\begin{array}{cccc}(0,k) & (4,k\oplus 4) & (1,k\oplus 1) & (5,k\oplus 5)\\ (2,k\oplus 2) & (6,k\oplus 6) & (3,k\oplus 3) & (7,k\oplus 7)\\ (8,k\oplus 8) & (12,k\oplus 12) & (9,k\oplus 9) & (13,k\oplus 13)\\ (10,k\oplus 10) & (14,k\oplus 14) & (11,k\oplus 11) & (15,k\oplus 15).\end{array}$$

Eventually, each symmetry that contains the relation $(0,k)$—i.e., $x[0]=x[k]$—also has the relations $(i,k\oplus i)$, $1,\dots,15$. Therefore, we have 15 distinct wordwise symmetry classes, of the form

$$(i,k\oplus i), i=0,\dots,15$$

for $k\in\{1,\dots,15\}$. Each class contains 2^{512} states. For example, the case $k=1$ provides directly C_1, and more generally $k=i$ corresponds to C_i.

4.3 Exploiting symmetric states

Preimages. Given a target digest, one can make a preimage attack similar to that in §2, and exploit symmetric states for the connection. The attack goes as follows:

- from the initial state, reach a symmetric state (of any class) by using $2^{1024-516-8}=2^{500}$ message blocks
- from a state before finalization, reach (backwards) another symmetric state (not necessarily of the same class)
- from these two symmetric states in classes C_i and C_j, use null message blocks in both directions to reach two states in $C_i\cap C_j$
- find a collision by trying $\sqrt{|C_i\cap C_j|}$ messages in each direction

Complexity of steps 1 and 2 is about 2^{501} computations of T. The cost of steps 3 and 4 depends on i and j; but it is upper bounded by 2×2^{256} operations.

Thus, in any case, the total complexity is about 2^{501} calls to T. This attack, however, finds messages of unauthorized size (more than 2^{256} bytes!).

One can find preimages of reasonable size by using a variant of the above attack: suppose $b>4$, from the initial state reach a state in a given class C_i, do the same backwards from a state before finalization. For a given b, the complexity of reaching a symmetric state depends on the C_i considered. Then one seeks a collision within C_i by trying messages preserving the symmetry: for example, if $b=5$ and $C_i=C_1$, then one has to preserve the equality $x[0]=x[1]$ and shall thus pick 5-byte messages of the form `XOOOX` (each digit stands for a byte). Since any C_i contains 2^{512} states, the cost of finding a collision within C_i is about 2^{256} trials in each direction.

Below we give a class example C_i that is the easiest to reach, depending on the value of b:

- $5\le b<9$: one of the best classes is C_1, which gives $(1024-2\times 4\times 8)/2=480$ equations to verify
- $9\le b<17$: one of the best classes is C_2, which give $(1024-2\times 8\times 8)/2=448$ equations to verify
- $17\le b<33$: one of the best classes is C_4, which gives $(1024-2\times 16\times 8)/2=384$ equations to verify

- $33 \le b < 65$: one of the best classes is C_8, which gives $(1024 - 2 \times 32 \times 8)/2 = 256$ equations to verify

If n equations have to be verified, the cost of reaching a symmetric state is about 2^n evaluations of T. Compared to the preimage attack in §2, the best speed-up obtained from a given C_i is when $b = 4d + 1$, where d is the number of 32-bit words that separate the first repetition of two words.

To illustrate this attack, let's study in more detail the case of C_1:

- if $b \equiv 0 \bmod 8$, there are $(1024 - 8b)/2 = 512 - 4b$ equations to satisfy, thus about 2^{512-4b} calls to T are necessary
- if $b \equiv 4 \bmod 8$, there are only $(1024 - 8b - 32)/2 = 496 - 4b$ equations to satisfy, because one has no condition on the first state word not xored with the message block
- generalizing, when $b \bmod 8 \le 4$, about $2^{512-4(b+(b \bmod 4))}$ calls to T are necessary
- when $b \bmod 8 > 4$, there are $(1024 - 8b - 32 + 8(b \bmod 4))/2$ equations to satisfy, which gives a cost $2^{496-4(b-(b \bmod 4))}$

The general formula for the number of equations is

$$512 - 32\lfloor b/8 \rfloor - 32\lfloor (b \bmod 8)/4 \rfloor - [(\lfloor (b \bmod 8)/4 \rfloor + 1) \bmod 2] \times 8(b \bmod 4) \ .$$

In the best case ($b \equiv 4 \bmod 8$), the attack is 2^{15} times faster than that in §2 (in the worst case, $b \equiv 0 \bmod 8$, it has the same complexity). Note that when $b = 5$, the attack makes about 2^{481} calls to T, against 2^{493} with the attack in §2.

Collisions on a weakened CubeHash. The initialization of CubeHash never leads to a symmetric initial state. Here we present a practical collision attack that would apply if the initial state were symmetric, and in $C_1 \cap C_8$.

Suppose that the initial state of CubeHashr/b-h is in $C_1 \cap C_8$, i.e. is of the form

`AAAAAAAA AAAAAAAAA BBBBBBBBB BBBBBBBB` .

If one hashes the $b2^{33}$-byte message that contain only zeros, then each of the 2^{33} intermediate states is an element of $C_1 \cap C_8$. Assuming that T acts like a random permutation of $C_1 \cap C_8$, one will find two identical states with probability about 0.63, which directly gives a collision.

5 Truncated differentials over T

This section shows how to detect non-randomness over the 10-round T transform. We start from a weight-64 difference to reach a weight-1 difference after 3 rounds with high probability; this *nonlinear* differential was discovered by simply computing backwards from the weight-1 difference.

We consider the input difference `80000000` in $x[16]$. The word $x[16]$ was chosen because $x[16] \cdots x[31]$ diffuse less in the first rounds than $x[0] \cdots x[15]$. We set a difference `80000000` to minimize the impact of carries.

We consider the following nonlinear differential. Input difference (weight-64):

```
18000000 10000000 08000000 30000000
00000040 00000080 00000000 00000000
00400000 00000000 00400000 01000404
00000003 80802002 00000001 81802004
40000000 08000000 00000000 E8020600
00000000 00000100 00000080 41F001C0
00400008 00000008 00400000 01000404
00000005 80802002 00000001 8080200C
```

Difference after one round (weight-26):

```
000E0000 00000000 00000000 00000000
00000000 00000040 00000080 00000040
01000004 00000000 00000004 00000000
00000000 00000000 00002000 00000000
800E0200 00000000 00000000 00000000
00000000 000000C0 00000080 000001C0
00000000 00000004 00000000 00000004
00000000 00002000 0000C000 00000000
```

Difference after two rounds (weight-9):

```
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 01000000
00000000 00002000 00000000 00002000
00000000 00000000 00000000 80000000
00000000 00000000 00000000 00100000
00000000 00000000 00000000 03000000
00000000 00002000 00000000 00002000
```

Difference after three rounds (weight-1):

```
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
80000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

which after another round gives with probability 1 the difference

```
80000000 00000000 80000000 00000000
00000400 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 80000000 00000000 80000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

This differential holds with negligible probability for a random input. But it holds for the input

```
DFB7AA11 7B2872F1 2848B142 64CB0AF9
17DA36E7 320A7AB2 27621CD8 B6E23031
3BCE90DB 0E496C61 AF4156BD 0B4D857F
4379D4C0 D495EAC9 038BD6E5 72A114CC
29065395 824774C3 F0923C34 28F3B2DD
74251DF6 1A562265 BD8EE5E3 DEFDD839
2804D3BE 89417DC3 F001CE4A 6A5328A8
2BEC024E B2306F17 1F2A7C6C 14BC37B6
```

For 32 random bits in $x[25]$ and $x[26]$ (at positions $4, \dots, 19$ in both), the differential is satisfied with probability approximately 0.985.

Note that in the linear model (i.e. when additions are replaced by xors), a differential path starting from the weight-1 difference cycles over 47 rounds. That is, it comes back to the difference `80000000` in $x[16]$ after 47 rounds. In the original model, however, linear differentials are followed with negligible probability.

Based on the above differential, we empirically looked for high-probability truncated differentials, based on the weight-64 input difference, and applying to each output bit a frequency test similar to that in [8, §2.1], with decision threshold 0.001 and 2^{20} samples. We found 4 output bits with p-value less than 0.001, at positions 579, 778, 841, and 842. Over 11 rounds and more, no bias was detected.

This observation is consistent with the fact that, when starting from the weight-1 difference, we could detect non-randomness on up to 7 rounds (now this difference is introduced three rounds later). Note that in a previous version of this article [2], we reached 8 rounds by starting one round before the weight-1 difference.

These observations indicate that 10-round T does not act as a random permutation, and that 10 rounds may not be overkill, as suggested in [4]. But note that the settings used don't correspond to a realistic attack scenario. Furthermore, if we restrict ourselves to differences in the first state byte, and put random bits in the rest of the state, then we observe non-randomness after up to 5 rounds.

6 Collisions for CubeHash1 and CubeHash2

To find a collision over CubeHashr/b-h, it is sufficient to find a high-probability differential over the r-round transform such that all—input and output—differences lie in the b first bytes of the states. We found such differentials for CubeHash2/120-512; for example the input difference

```
70020000 00000000 80000000 00000000
00000400 00000040 00000080 00000040
01000004 00000000 00000004 00000000
00000000 00000000 00002000 00000000
50020600 80000000 00000000 00000000
00000000 000000C0 00000080 000000C0
00000000 0000007C 00000000 00000004
00000000 00002000 00000000 00000000
```

leads after two rounds to a difference `80000000` in $x[16]$ (and no difference elsewhere) with high probability. This allowed us to present an example of collision [1] for CubeHash2/120-512 (see Appendix A). Finding such collisions has a negligible cost, compared to the 2^{32} complexity of the generic attack.

A similar approach allows to find collisions on CubeHash2/114: instead of reaching a difference in $x[16]$, one reaches a difference in $x[17]$, which only requires nonzero input difference in the first 114 bytes.

Collisions can also be found for CubeHash1/106, since one can reach the difference `80000000` in $x[7]$ (and no difference elsewhere) after one round, by putting nonzero differences only in the first 106 bytes.

Acknowledgments

The authors wish to thank Eric Brier for his helpful comments regarding symmetry classes. They are indebted to Dan Bernstein for commenting on previous versions of this paper, and for pointing out a few errors.

References

1. Jean-Philippe Aumasson. Collision for CubeHash2/120-512. NIST mailing list, 4 Dec 2008, 2008. Copy at `http://ehash.iaik.tugraz.at/uploads/a/a9/Cubehash.txt`.
2. Jean-Philippe Aumasson, Willi Meier, Mara Naya-Plasencia, and Thomas Peyrin. Inside the hypercube. Cryptology ePrint Archive, Report 2008/486, 2008. version 20081124:132635.
3. Daniel J. Bernstein. CubeHash appendix: complexity of generic attacks. Submission to NIST, 2008.
4. Daniel J. Bernstein. CubeHash attack analysis (2.B.5). Submission to NIST, 2008.
5. Daniel J. Bernstein. CubeHash specification (2.B.1). Submission to NIST, 2008.
6. Persi Diaconis and Frederick Mosteller. Methods for studying coincidences. *Journal of the American Statistical Association*, 84(408):853–861, 1989.
7. Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO*, 2004.
8. NIST. SP 800-22, a statistical test suite for random and pseudorandom number generators for cryptographic applications, 2001.
9. Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. In *ICISC*, 2006.

A Collision for CubeHash2/120-512

The following 2880-bit messages collide through CubeHash2/120-512. First message (as a sequence of bytes):

```
43CACBA20E63FF78D505D9F9850EE62C9B45B188AE22E
9FEC4FEE220E5C3A9AE6F06868CD0A1122AE38B386F13
58C0FBC3746E574BEB5D6E09399B4084D4D787E6C820B
FE6615F68C8EA490686609E2A65833582C4806EB0C21B
78F45F76346A689B52D3D1F6CF5311DE4ED0B365DDB15
76907DC0326A2EB2737D5297D036CF400AE27132751CF
BF88DDFECF810CEB4AAD133BDBD21D7334CE9C9FC977C
A46B5AE61BFF61618B1ED193268667B0ADDD220AFDE2A
416090293996BAB0E62CEAC10B60B87AAC0E088B9199D
029288D878180034668C6BB9DE64CA89DCE4C284AD41B
F38414D3E4D27A5DF41A428842CCDEF0F1F2F3F4F5F6F
7F8F9FAFBFCFDFEFF000102030405060708090A0B0C0D
0E0F101112131415161718191A1B1C1D1E1F202122232
425262728292A2B2C2D2E2F303132B33435363738393A
3B3C3D3E3F404142434445464748494A4B4C4D4E4F505
152535455565758595A5B5C5D5E5F6061626364656667
```

Second message:

```
43CACBA20E63FF78D505D9F9850EE62C9B45B188AE22E
9FEC4FEE220E5C3A9AE6F06868CD0A1122AE38B386F13
58C0FBC3746E574BEB5D6E09399B4084D4D787E6C820B
FE6615F68C8EA490686609E2A65833582C4806EB0C21B
78F45F76346A689B52D3D1F6CF5311DE4ED0B365DDB15
76907DC0326A2EB2737D7597D036CF400AE27132751CF
BF88DDFECFC10CEB4A2D133BDB921D7334CA9C9FC877C
A46B5AA61BFF61618B1ED193268667B0ADDD2208FDE2A
416090293990B8E0E62CEAC10B60B87AAC0E088B9199D
029E88D87810003466806BB9DE64CA89DCE30284AD41B
F38414D7E4D27A5DF41A428862CCDEF0F1F2F3F4F5F6F
7F8F9FAFBFCFDFEFF000102030405060708090A0B0C0D
0E0F101112131415161718191A1B1C1D1E1F202122232
425262728292A2B2C2D2E2F3031323333435363738393A
3B3C3D3E3F404142434445464748494A4B4C4D4E4F505
152535455565758595A5B5C5D5E5F6061626364656667
```

Their common digest is

```
C48C99A0D37E1EB2AC7C42EDF4EDEC7AB73B7689506856CA458770096FC4A38B
19016DCA3834F1F5805B3E47F3C38C705B4A25F5C2801D41A15CDF9E3603C61A
```