Symmetric States and their Structure: Improved Analysis of CubeHash

Niels Ferguson, Stefan Lucks, and Kerry A. McKay*

[1] Microsoft Corp., niels@microsoft.com
[2] Bauhaus-Universität Weimar, stefan.lucks@uni-weimar.de
[3] George Washington University, kerry@gwu.edu

Abstract. This paper provides three improvements over previous work on analyzing CubeHash, based on its classes of symmetric states: (1) We present a detailed analysis of the hierarchy of symmetry classes. (2) We point out some flaws in previously claimed attacks which tried to exploit the symmetry classes. (3) We present and analyze new multicollision and preimage attacks. For the default parameter setting of CubeHash, namely for a message block size of $b = 32$, the new attacks are slightly faster than 2^{384} operations. If one increases the size of a message block by a single byte to $b = 33$, our multicollision and preimage attacks become much faster – they only require about 2^{256} operations. This demonstrates how sensitive the security of CubeHash is, depending on minor changes of the tunable security parameter b.

1 Introduction

The CubeHash family of hash functions [1] is a round-2 candidate for the SHA-3 algorithm. CubeHash$r/b-h$ depends on three parameters r, b, and h and produces h-bit hash values using r rounds per message block; b is the size of a message block (in bytes). Typically $h \in \{224, 256, 384, 512\}$, $b \in \{1, \ldots, 128\}$, and $r \in \{1, 2, 3, \ldots\}$. As mentioned by the author of CubeHash [1], and later studied in some detail by other researchers [2], there exist certain symmetries through the round function. Once the state is in a class of symmetric states, it suffices to choose the all-zero message block to preserve the symmetry.

Since the original submission with its parameter recommendation has been criticized as being too slow, its author later tweaked CubeHash, proposing a variant for normal operations for users not concerned with attacks using 2^{348} operations. The only difference to the conservative but very slow variant for formal operations is the choice of the parameter b: the formal case requires $b = 1$, while the normal choice is $b = 32$ and "aimed at sensible users" [3].

1.1 Our Contributions

In this paper, we refine the results from [2]. Our main results are the following.

(1) We provide a precise analysis of the complete hierarchy of symmetry classes in CubeHash.
(2) We have a closer look at two attacks from [2] to exploit the symmetry classes. As it turns out, the claimed attack complexities from [2] are too optimistic.
(3) We describe new attacks and analyze their complexities:
   - Multicollision and preimage attacks for the CubeHash with $b = 32$. The attacks are slightly faster than the 2^{384} operations claimed in [3] (the multicollision attack takes time $2^{381.2}$, the preimage attack takes time $2^{283.7}$).
   - Multicollision and preimage attacks for CubeHash with $b = 33$, which are a lot faster than the claimed 2^{384} operations (generating a k-collision takes time $\lceil \log_2(k) \rceil \times 2^{256}$, finding a preimage takes time 3×2^{256}).

The number r of rounds is irrelevant for our attacks. The hierarchy of symmetry classes doesn't depend on the output size h. Neither do our multicollision and preimage attacks – though for small h, such as $h \in \{224, 256\}$, our attacks are not always an improvement over the generic standard attacks.

* Work supported in part by the National Science Foundation grant CCF 0830576

1.2 CubeHash

All versions of CubeHash have a 128-byte state, represented as 32 32-bit words. These words are denoted x_{00000} to x_{11111} as the first to last words, respectively. The hash function can be viewed as the following three steps: initialization, message processing, and finalization.

Round Function The round function is an ARX (addition, rotation, xor) function, where addition is performed modulo 2^{32}. Each round consists of 10 steps:

1. for each (j, k, l, m), $x_{1jklm} = x_{0jklm} \boxplus x_{1jklm}$
2. for each (j, k, l, m), $x_{0jklm} = x_{0jklm} \lll 7$
3. for each (k, l, m), swap x_{00klm} with x_{01klm}
4. for each (j, k, l, m), $x_{0jklm} = x_{0jklm} \oplus x_{1jklm}$
5. for each (j, k, m), swap x_{1jk0m} with x_{1jk1m}
6. for each (j, k, l, m), $x_{1jklm} = x_{1jklm} \boxplus x_{0jklm}$
7. for each (j, k, l, m), $x_{0jklm} = x_{0jklm} \lll 11$
8. for each (j, l, m), swap x_{0j0lm} with x_0j1lm
9. for each (j, k, l, m), $x_{0jklm} = x_{0jklm} \oplus x_{1jklm}$
10. for each (j, k, l), swap x_{1jkl0} with x_{1jkl1}

Observe that the round function is a permutation over the set of states, and one can run it backwards as efficiently as one can run it in forward direction.

Initialization The initial state is computed by setting $x_{00000} = \frac{h}{8}$, $x_{00001} = b$, $x_{00010} = r$, and all other state words are 0. $10r$ rounds are then applied to reach the initial state. This creates a different initial value for each parameter selection. For any fixed set of parameters, we write H_0 for this initial state.

Message processing The message must be padded to a multiple of b bytes. CubeHash first appends a 1, and then as many 0's as necessary to ensure the message length is a multiple of $8b$ bits. A padded message M consists of k blocks and is processed one b-byte block at a time. Given the initial state H_0 and message blocks M_1, M_2, ..., state H_j is reached by applying r rounds to $H_{j-1} \oplus (M_0 << (128 - b))$. That is, the message block M_j is exclusive-ored with the *first* b bytes of state H_{j-1}, before applying the round function r times.

Given an initial state H_0 and a message (or a message prefix) M, whose length is a multiple of the message block length, we write

$$H(H_0, M)$$

for the internal state which we get by mixing in the M. As the round function is invertible, one can just as well compute H_0 from M and $H(H_0, M)$.

Finalization The finalization step occurs after all blocks have been through the message processing step. If M is the entire message, then the internal state is $H_{i-1} = H(H_0, M)$. Finalization begins by exclusive-oring 1 into the last state word of H_{i-1} ($x_{11111} = x_{11111} \oplus 1$) to achieve state H_i. Then $10r$ round functions are applied to H_i to obtain the H_{i+1}. Finally, H_{i+1} is truncated to the required hash size. I.e., the hash of M consists of the first $\frac{h}{8}$ bytes of H_{i+1}.

Parameter Recommendations The parameter h is defined by the SHA-3 digest length. For the first round of the SHA-3 submission, Dan Bernstein, the author of CubeHash, proposed $r = 8$ and $b = 1$. As it turned out, CubeHash was very slow, compared to most of the other SHA-3 candidates. For the second round, he fixed r to $r = 16$ and proposed two variants of CubeHash with different choices for b

- $b = 32$ for "normal" operations, *"aimed at sensible users"* [3], and
- $b = 1$ for "formal" operations, for users *"concerned with attacks using* 2^{348} *operations"* [3].

Note that the "normal" variant is 16 times faster than the original proposal, and its speed appears to be competitive to the speed of other SHA-3 candidates. On the other hand, the "formal" variant is twice as slow as the original submission.

1.3 Related Work

Generic attacks, i.e., attacks which model r applications of CubeHash's round function as an arbitrary or random permutation over 1024-bit states, have already been considered in an appendix of the original CubeHash submission and, more diligently, in [2]. The core observation is that by choosing a message block the attacker can determine $8b$ bits of the internal state. Thus, if the other $1024 - 8b$ state bytes collide, one can enforce a collision by appropriately selecting the next message blocks. Accordingly, this attack finds collisions in the 1024-bit state after trying out about $\sqrt{2^{1024-8b}} = 2^{512-4b}$ messages. Similarly, one can find preimages by a meet-in-the-middle attack by the equivalent of trying out $2 \times 2^{512-4b} = 2^{513-4b}$ messages. The preimage attack can also be applied to find a preimage of the all-zero state. If both the old state and the current message block are zero, then the new state is zero again, i.e., $H(0, 0) = 0$. Once we have found a message M_0 with $H(H_0, M_0) = 0$, we actually have found an arbitrary-size multicollision, since

$$0 = H(H_0, M_0) = H(H_0, M_0||0) = H(H_0, M_0||0||0) = \ldots$$

Symmetric properties of the round function have been mentioned in the submission document and where further studied in [2]. Symmetry occurs when properties of the input to the round function are preserved in the output. If the property is equalities between state words, there are 15 distinct symmetry classes with 67 subsets [2]. Once a state conforms to a symmetric state, the state cannot get out of the symmetry until a nonzero block is mixed in.

We stress that for every symmetry class C_i, the round function is a permutation over the set of states in C_i. Thus, mixing in an all-zero block, including the application of r rounds, doesn't change the symmetry class one is in, i.e., by mixing in an all-zero block, one can neither get into a symmetry class, nor leave it.

Note that symmetry is not present in the initial state, and symmetry in the message processing is destroyed in finalization.

For completeness, the symmetry classes are shown in table 1.

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

Table 1. Symmetry Classes [2]

Several collisions and preimage attacks have been demonstrated on variants of CubeHash [4] [5] [6] [7] [8] [9] [10]. A linearization framework[11] and statistical approach[12] have also been applied.

2 Symmetry Hierarchy

The 15 symmetry classes presented in [2] are very useful in analyzing CubeHash. We add further structure to these symmetry classes by placing them into a hierarchy that describes how classes relate to each other. In particular, we provide further structure to describe the intersection of symmetry classes.

Let S be the state as an array of 32-bit words. Let $V = \{0,1\}^4$, the space of all 4-bit vectors, and D be a linear subspace of V. Then there is a symmetry in CubeHash where a state that has $\forall d \in D$ and $\forall i \in (0,..,15), S[i] = S[i \oplus d]$ and $S[16+i] = S[16+i] \oplus S[16+(i \oplus d)]$.

For $D = V$, this yields a state class that has 2 free words, A and B, and 28 words fixed by the values of A and B. This symmetric state class can only take on 2^{64} different values. Following the notation of [2], that state class has the form:

`AAAAAAAA AAAAAAAA BBBBBBBB BBBBBBBB`

Let D be the linear subspace of all 3-dimensional vectors in V. There are 15 such subspaces, each representing a state that has 4 free words, and the other 28 are defined by equality relations. The result is a class that contains $2^{32+32+32+32} = 2^{128}$ distinct state values. These 15 3-d symmetry classes are listed in table 2.

$3d_1$:	`AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD`
$3d_2$:	`AAAABBBB AAAABBBB CCCCDDDD CCCCDDDD`
$3d_3$:	`AAAABBBB BBBBAAAA CCCCDDDD DDDDCCCC`
$3d_4$:	`AABBBBAA AABBBBAA CCDDDDCC CCDDDDCC`
$3d_5$:	`AABBBBAA BBAAAABB CCDDDDCC DDCCCCDD`
$3d_6$:	`ABBAABBA ABBAABBA CDDCCDDC CDDCCDDC`
$3d_7$:	`ABBAABBA BAABBAAB CDDCCDDC DCCDDCCD`
$3d_8$:	`ABBABAAB ABBABAAB CDDCDCCD CDDCDCCD`
$3d_9$:	`ABBABAAB BAABABBA CDDCDCCD DCCDCDDC`
$3d_{10}$:	`AABBAABB AABBAABB CCDDCCDD CCDDCCDD`
$3d_{11}$:	`AABBAABB BBAABBAA CCDDCCDD DDCCDDCC`
$3d_{12}$:	`ABABABAB ABABABAB CDCDCDCD CDCDCDCD`
$3d_{13}$:	`ABABABAB BABABABA CDCDCDCD DCDCDCDC`
$3d_{14}$:	`ABABBABA ABABBABA CDCDDCDC CDCDDCDC`
$3d_{15}$:	`ABABBABA BABAABAB CDCDDCDC DCDCCDCD`

Table 2. 3-dimensional symmetry classes

Let D be the linear subspace of all 2-dimensional symmetries. Then there are 35 distinct 2-dimensional subspaces, yielding states with 8 free words and 24 words defined by relations. The complete list of 2-dimentsional states appears in table 5.

The 15 classes of [2] (shown in table 1) correspond to the nontrivial 1-dimensional subspaces. Only 16 words are free and the other 16 are completely determined by the free words.

Finally, there is a symmetry class where all words are the same. In total, this yields 67 distinct symmetry classes, corresponding to the 67 subsets of [2]. The intersection between two symmetry classes can be represented as the linear span of the union of their subspaces.

Figure 1 depicts the symmetry class hierarchy. Each 1-dimensional symmetry class is part of 7 2-dimensional symmetry classes, and each 2-dimensional symmetry class contains 3 1-dimensional symmetry classes. Each 2-dimensional symmetry class is also part of 3 3-dimensional symmetry classes and each 3-dimensional symmetry class contains 7 2-dimensional symmetry classes. All 15 3-dimensional symmetry classes are part of the single 4-dimensional symmetry class.

Consider the 2-dimensional state `AAAABBBB CCCCDDDD EEEEFFFF GGGGHHHH` for example. It is part of 3-dimensional classes $3d_1$, $3d_2$, and $3d_3$, and contains 1-dimensional classes C_1, C_2, and C_3.

If one selects one symmetry class from each level in the graph such that the higher-dimensional symmetry classes always contain the lower dimensional one, then you can find a total of $15 \times 7 \times 3 = 315$ different symmetry hierarchies.

Split States Each of the 315 symmetry hierarchies leads to a different split state representation of the form $T[0], T[1], T[2..3], T[4...7], T[8..15], T[16..31]$, where there is a simple linear mapping between S and T and we have the property that if $T[16..31]$ is zero at the start of the round function, then it is zero at the end of the round function. In fact, for $i \in \{0, 1, 2, 4, 8, 16\}$, if $T[i..31]$ is zero at the start of a round function, then $T[i..31]$ is zero at the end.

2.1 Traversing the Hierarchy

It is easy to see that the intersection of 1-dimensional states leads to 2-dimensional states, but since there is no way to leave a symmetric state, it seems difficult to traverse the hierarchy from higher dimensions to lower ones in such a way that the higher dimension state is not part of a symmetry class in the higher dimension. For example, it is unclear how to go from a 2-dimensional state to a 1-dimensional state such that the 1-dimensional state is symmetric, but the 2-dimensional state is not in a 2-dimensional symmetry class. We show that there is a way using portions of the state. In particular, if the state is divided into halves or quarters, where each half or quarter belongs to a higher dimension symmetric state, a lower dimension symmetry may be achieved if the combination of higher-level symmetry class patterns allows it. Thus, the hierarchy can be traversed from 1 dimension to 4 dimensions through intersection, and from 3 dimensions to 1 dimension in this way.

A 2-dimensional symmetric state can be reached from 3-dimensional halves that combined do not belong to a 3-dimensional symmetry class in the following way:

1. Set the left half of the state such that it conforms to the left half of a 3-dimensional symmetry class, $3d_i$, $i \in 1 \dots 15$.
2. Set the right half of the state such that it conforms to the right half of a 3-dimensional symmetry class, $3d_j$, $j \in 1 \dots 15$, $i \neq j$.

All 35 2-dimensional symmetry classes can be reached in this manner. If a particular 2-dimensional state is needed, there are further restrictions on the classes the halves belong to. In order to reach a class $2d_i$, the 3-dimensional halves must belong to $3d_j$ and $3d_k$ such that $2d_i$ is part of $3d_j$ and $3d_k$.

1-dimensional symmetric states can be generated in a similar way using 2-dimensional symmetric class halves. However, not all combinations yield a symmetric state. The left and right halves of the state should both belong to classes that a 1-dimensional class is a part of in the hierarchy. For example, C_1 is part of $2d_1, 2d_2, 2d_3, 2d_8, 2d_9, 2d_{10},$ and $2d_{11}$. If each half is set to a pattern found in these symmetric classes, the state will belong to C_1.

It is convenient to divide the state into left and right halves, but not necessary. It may also be divided into 32-byte quarters, where each quarter is from a symmetric state. We call these quarter-symmetries. There are many duplicate state forms when only quarters are considered. For example, $3d_2$ and $3d_3$ both have the same pattern when only 8 words are viewed. Then we say $3d_2$ to mean either class, since they are equivalent in this context. Thus we only consider one class for these duplicates, as a representative.

Tables 3 and 4 show the quarter-symmetry transitions to lower-dimension symmetry classes. When traversing the hierarchy from 3-dimensional quarter-symmetries to 1-dimensional symmetry classes, the 2-dimensional quarter-symmetries must contain the 3-dimensional quarter-symmetries and be part of the 1-dimensional symmetry class. Note that only states where each quarter is self-contained can be reached via quarter-symmetries. That is, states that have equalities that cross the quarter boundary cannot be reached this way – that traversal must occur with half-symmetries.

3 Claimed Attacks, based on Symmetry Classes

The authors of [2] claim two attacks which take advantage of symmetry in the compression function. We argue that for both of these attacks, the analysis is much to optimistic.

3.1 A Claimed Preimage Attack

The first is a preimage attack that arrives in a symmetric state, $S \in C_i$, by going forward with 2^{500} message blocks. It arrives at another symmetric state, $T \in C_j$, by computing backwards from finalization.

The authors of [2] claim that if C_i and C_j are different symmetry classes one just needs to bridge these by mixing in null messages to eventually fall into the intersection symmetry class $C_i \cap C_j$. This reasoning is flawed, however: Mixing in a null message is a permutation over any symmetry classes. Thus, if one is in a state C_i and mixes in the null message, one can be sure to stay in C_i. But if one is not in C_j (i.e., not in $C_i \cap C_j$), one can never get into C_j (or $C_i \cap C_j$) that way. Similarly for going backward from C_j to $C_i \cap C_j$. Thus, the preimage attack can only work if $C_i = C_j$, I.e., if S and T are in the same symmetry class.

Even if $C_i = C_j$, the claimed complexity of 2×2^{256} steps for finding a way from S to T by mixing in zero-messages is much too optimistic, since mixing in zero-messages is a permutation. Define the "successor" of a state $X \in C_i$ as the new state Y which one gets by mixing in a zero message block. If we view the elements of C_i as the vertices in a directed graph, we draw an edge from X to Y. Since mixing in a zero message is a permutation, this graph will consist of some disjoint cycles. Going from $S \in C_i$ to $T \in C_i$ by mixing in zero-messages is possible if and only if X and Y are on the same cycle. If S and T are not on the same cycle in C_i, the attack will fail. If they are on the same cycle, the attack will succeed eventually. Thus, this part of the attack will take close to 2^{512} steps, instead of the claimed 2×2^{256}. Thus, even if the attack doesn't fail, the message length will be close to $b \times 2^{512}$ bytes.

3.2 A Claimed Collision Attack

The second attack is a collision attack on a weakened version of CubeHash, where the initial state is symmetric. (Note that the CubeHash specification doesn't allow a symmetric initial state.) In that initial state, all words in x_{0jklm} are equal and all words in x_{1jklm} are equal, yielding a state of the form:

```
AAAAAAAA AAAAAAAA BBBBBBBB BBBBBBBB
```

Null messages can be used to cycle through all such symmetric states. With a $b2^{33}$-byte zero message, the authors of [2] expect a collision with probability 0.63. This seems to assume that the round function behaves like a random function over the set of states in the symmetry class. The assumption is false, and the claimed complexity is too optimistic, again, since the round function (and mixing in zero-messages) is a permutation. We expect that one would need a message close to $b \times 2^{64}$ zero-bytes.

4 Exploiting the Degrees of Freedom Available

The core problem for the above attacks from [2] is that the attacker is only allowed to apply a single fixed permutation to a message state $X \in C_i$, which is defined by mixing in an all-zero message block. Note that [2] was written *before* CubeHash was tweaked for improved performance. At that point in time, the default was $b = 1$, and in this case, the only message block one could mix in while maintaining a given symmetry class was the zero-byte.

Below, we will consider the cases $b = 32$ (now the default for CubeHash) and $b = 33$ and exploit the additional degrees of freedom. This allows us to choose different nonzero message blocks, which can be mixed in while still maintaining a given symmetry symmetry class:

- For $b = 32$, the defined default setting of CubeHash "normal" operations, we restrict ourselves to the symmetry classes $C_1, \ldots, C_7$.

– If we consider a slightly weaker variant of CubeHash, where the adversary can control one additional byte, i.e., $b = 33$, then our approach works for all symmetry classes $C_1, \ldots, C_{15}$.

Note that the first eight words (i.e., the first 32 bytes) of a state in $C_1, \ldots, C_7$ are independent from any of the remaining 24 words. Also note that the state defines a certain pattern (e.g., "ABBACDDC" for C_i), but apart from that pattern, there are no restrictions on the state. If we are in such a state and mix in any 8-word message block which just follows that pattern, we will remain in that state. Of course, the all-zero message block follows such a pattern, but there are $2^{128} - 1$ nonzero message blocks which also follow the required pattern.

In the case of symmetry classes $C_8, \ldots, C_{15}$, the first eight state words are identical to the next eight state words. Thus, we need to control more than the first eight state words. But if $b = 33$, we can arbitrarily choose one byte in the ninth state word. Thus, apart from the all-zero message block, there are $2^8 - 1 = 255$ nonzero message blocks (with 31 zero-bytes and two identical nonzero bytes), to can choose from.

5 Our attacks

In this section, we describe our attacks, which are mainly based on the above observations. A vital part for the attack is, however, to *get into a symmetric state.* Since the internal state H_0 of CubeHash is not symmetric, any attack based on exploiting these symmetries must first find a message M such that $H(H_0, M)$ is symmetric:

→ Generate random message blocks as message M_{1x} of length kb bytes (for some integer k) until one can find a b-byte message M_{1y} such that

$$H(H_0, (M_{1x}||M_{1y})) = H_1$$

is symmetric.

First, consider $b = 33$. If we don't care about which symmetry class we are in, then we statistically expect to succeed after trying out about $2^{256}/8 = 2^{253}$ different M_{1x}. Our chances to reach either of the states C_1, C_2, …, C_7 with that amount of work is negligible; we rather expect to reach any of the states in classes C_i for $i \in \{8, \ldots, 15\}$. If we want to get into a fixed class $C_i \in \{C_8, \ldots, C_{15}\}$, we need about 2^{256} attempts.

Now consider the case $b = 32$. If we are trying to reach a fixed state C_i in $C_1, \ldots, C_7$, then we expect to succeed after 2^{384} attempts. If we don't care which state in $C_1, \ldots, C_7$ we will get, we need about $2^{384}/7 \approx 2^{381.2}$ attempts.

Of course, a smaller b makes it even harder to reach a symmetric state. [2] did consider the extreme case $b = 1$, which has been the recommended value at the time [2] has been written. In that case, all symmetry classes are equally hard to get into, and if we don't care which class we will get into, we have to try $2^{504}/15 \approx 2^{500.1}$ message blocks $M_{1,x}$.

Note that the same approach works backward: Instead of starting with an arbitrary initial H_0 and searching a message $M_{1x}||M_{1y}$, such that the state $H_1 = H(H_0, M_{1x}||M_{1y})$ is symmetric, we can start with a final state H_3 and search for some message M such that the state H_2 with $H_3 = H(H_2, M)$ is symmetric. The amount of work depends on the symmetry class we are targeting in the exactly same way as it does for the forward direction.

5.1 Multi-Collisions on CubeHash

Note that any of the symmetry classes $C_1, \ldots, C_{15}$ contains 2^{512} states. Thus, once our state is symmetric, we now can find different messages with colliding internal states by trying out about $\sqrt{2^{512}} = 2^{256}$ messages. (Of course, all message blocks of all the messages we consider must follow the pattern determined by the symmetry class, such as "ABBACDDC" in the case of C_3.) This allows us to mount a Joux-style multicollision attack [13]: finding a k-collision takes the time sequentially computing $\lceil \log_2(k) \rceil$ collisions, each in time 2^{256}.

For $b = 32$, the complexity to find a k-collision, including the time for initially getting into a symmetric state is

$$2^{381.2} + \lceil \log_2(k) \rceil \times 2^{256},$$

which is dominated by $2^{381.2}$, i.e., by the time for getting into a symmetric state, for any imaginable k.

For $b = 33$, finding k-collisions is much faster. It can be done in time

$$2^{253} + \lceil \log_2(k) \rceil \times 2^{256} \;\approx\; \lceil \log_2(k) \rceil \times 2^{256}.$$

Thus, in this case the complexity is dominated by the Joux-multicollision part.

5.2 Preimages for CubeHash

Consider a hash value Z of size h (e.g., $h = 512$). Write H_0 for the initial state. The attacker's goal is to find a message M, such that CubeHash(M)=Z. The basic structure for the attack is the following:

1. Extend Z by $(1024 - h)$ bit to a full 1024-bit state and run the finalization backwards to get a state H_4.
2. Search for a message prefix M_1 and a $H_1 = H(H_0, M_1)$ in a symmetry class C_i.
3. Search for a postfix M_4 and H_3 in the same C_i with $H(H_3, M_4) = H_4$.
4. Apply a meet-in-the-middle approach to search two message parts M_2 and M_3 and a state $H_2 \in C_i$ with $H(H_1, M_2) = H_2$ and $H(H_2, M_3) = H_3$.

Jointly, the message parts $M_1, \ldots, M_4$ form a message $M = (M_1 \,||M_2 \,||M_3 \,||M_4)$ with $H(H_0, M) = H_4$ and thus CubeHash(M)=Z. (Of course, the length of each of M_1, M_2, M_3, and M_4 must be a multiple of the block length.)

The first three steps in the attack are resemble the attack from [2], except that our attacker is allowed to mix in nonzero message blocks, and H_3 must be in the same symmetry class as H_1. Accordingly, H_2 must be in the very same symmetry class, rather than in the intersection of two different symmetry classes.

The first step of this attack is trivial. Since the size of any symmetry class is 2^{256}, we expect the fourth step to work if we try out 2^{256} candidates for M_2 and the same number of candidates for M_3. Thus, the fourth step requires 2×2^{256} steps.

For the analysis of the second and the third step, we distinguish between $b = 32$ and $b = 33$.

First, consider $b = 32$. In this case, we are restricted to $C_i \in \{C_1, \ldots, C_7\}$. In the second step, we don't care which of the seven possible sets C_i we get into. Thus this part of the attack requires $2^{381.2}$ units of time. But then, the second step requires 2^{384} units of time, since C_i is fixed. There is a little trick to improve this:

- Repeat the second step twice, to get into two states $H_1 \in C_i$ and $H_1' \in C_1' \neq C_i$. This takes time $2^{384}/7 + 2^{384}/6 < 2^{384}/3 \approx 2^{382.4}$.
- In the third step, we succeed if $H_3 \in C_i \cup C_i'$. Thus, the third step only takes time 2^{383}.

The second and the third step together need time $\approx 2^{382.4} + 2^{383} \approx 2^{383.7}$. This dominates the attack complexity for $b = 32$.

Now consider $b = 33$. In the second, we don't care which of the 15 C_i we get into, and we need to try out $2^{256}/8 = 2^{253}$ messages. In this case, the third step takes time 2^{256}, thus, the overall attack complexity is $3 \times 2^{256} + 2^{253} \approx 3 \times 2^{256}$. The little trick from above can reduce this a bit, but since the meet-in-the-middle part in the fourth step needs 2×2^{256} steps alone, we will not get *much* better than 3×2^{256} for $b = 33$ and the complete preimage attack.

6 Final Remarks

Note that $b = 32$ is sufficient for our approach, but not necessary. If $b \leq 4$, there seems to be no way to choose nonzero message blocks while maintaining a given state. But $b = 5$ would suffice for our attacks. The symmetry class C_1 defines a pattern "AAB...", without any further occurrence of "A". Thus, we can freely choose one byte in the second message word, and then ensure that the first message word is identical to the second message word. Adapting our attacks to the $b = 5$ case would slow the attacks down a lot, since getting into a state in C_1 from a non-symmetric initial state would take time 2^{472}.

Summary and Conclusion In the current paper, we have provided a detailed analysis of the hierarchy of symmetry classes of CubeHash. We demonstrated that certain attacks presented previously are more complex than claimed by their authors. Finally, we presented and analyzed some attacks on CubeHash on our own:

- For CubeHash with $b = 32$ (the default for normal operations), we presented multicollision and preimage attacks with a complexity of slightly less than 2^{384} hash operations, and
- for CubeHash with $b = 33$ (slightly weaker than the default), we presented multicollision and preimage attacks with a complexity of slightly more than 2^{256} operations.

Since the author of CubeHash explicitly disregards attacks beyond 2^{384} operations for the default $b = 32$ (he recommends an extremely slow "formal" mode with $b = 1$ to ensure resistance against such extreme attacks), our attacks on CubeHash are small improvements over the best attacks known to the designer of CubeHash. What we consider remarkable is the deep drop of security if one increases the tunable security parameter just by one, from $b = 32$ to $b = 33$. We would expect a "conservative proposal with a comfortable security margin" (as written in the revised CubeHash specification), to be less sensitive to minor changes of the tunable security parameters.

References

1. Bernstein, D.J.: Cubehash specification (2.b.1). Submission to NIST (Round 2) (2009)
2. Aumasson, J.P., Brier, E., Meier, W., Naya-Plasencia, M., Peyrin, T.: Inside the hypercube. In: ACISP. Volume 5594 of LNCS., Springer (2009) 202–213
3. Bernstein, D.J.: Cubehash parameter tweak: 16 times faster. Available online (2009)
4. Khovratovich, D., Nikolic', I., Weinmann, R.P.: Preimage attack on cubehash512-r/4 and cubehash512-r/8. Available online (2008)
5. Dai, W.: Collisions for cubehash1/45 and cubehash2/89. Available online (2008)
6. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Attack for cubehash-2/2 and collision for cubehash-3/64. NIST mailing list (local link) (2009)
7. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Real collisions for cubehash-4/64. NIST mailing list (local link) (2009)
8. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Real collisions for cubehash-4/48. NIST mailing list (local link) (2009)
9. Aumasson, J.P.: Collision for cubehash2/120-512. NIST mailing list (local link) (2008)
10. Brier, E., Peyrin, T.: Cryptanalysis of cubehash. Available online (2009)
11. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization framework for collision attacks: Application to cubehash and md6. Cryptology ePrint Archive, Report 2009/382 (2009) urlhttp://eprint.iacr.org/.
12. Bloom, B., Kaminsky, A.: Single block attacks and statistical tests on cubehash. Cryptology ePrint Archive, Report 2009/407 (2009) http://eprint.iacr.org/.
13. Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: CRYPTO. (2004) 306–316

A Appendix

Symmetric class	2d quarter-symmetry classes
C_1	1, 2, 3, 8(to 11)
C_2	1, 6, 7, 12(to 15)
C_3	1, 4, 5, 16(to 19)
C_4	2, 4, 6, 20(to 23)
C_5	2, 5, 7, 24(to 27)
C_6	3, 5, 6, 28(to 31)
C_7	3, 4, 7, 32(to 35)

Table 3. 2d quarter-symmetries to symmetry classes (equivalent quarters in parentheses)

Symmetric class	3d quarter-symmetry classes
C_1	1, 2(3), 4(5), 10(11)
C_2	1, 2(3), 12(13), 14(15)
C_3	1, 2(3), 6(7), 8(9)
C_4	1, 6(7), 10(11), 12(13)
C_5	1, 8(9), 10(11), 14(15)
C_6	1, 4(5), 8(9), 12(13)
C_7	1, 4(5), 6(7), 14(15)
$2d_1$	1, 2(3)
$2d_2$	1, 10(11)
$2d_3$	1, 4(5)
$2d_4$	1, 6(7)
$2d_5$	1, 8(9)
$2d_6$	1, 12(13)
$2d_7$	1, 14(15)

Table 4. 3d quarter-symmetries to symmetry classes (equivalent quarters in parentheses)

$2d_1$:	AAAABBBB	CCCCDDDD	EEEEFFFF	GGGGHHHH
$2d_2$:	AABBAABB	CCDDCCDD	EEFFEEFF	GGHHGGHH
$2d_3$:	AABBBBAA	CCDDDDCC	EEFFFFEE	GGHHHHGG
$2d_4$:	ABBAABBA	CDDCCDDC	EFFEEFFE	GHHGGHHG
$2d_5$:	ABBABAAB	CDDCDCCD	EFFEFEEF	GHHGHGGH
$2d_6$:	ABABABAB	CDCDCDCD	EFEFEFEF	GHGHGHGH
$2d_7$:	ABABBABA	CDCDDCDC	EFEFFEFE	GHGHHGHG
$2d_8$:	AABBCCDD	AABBCCDD	EEFFGGHH	EEFFGGHH
$2d_9$:	AABBCCDD	BBAADDCC	EEFFGGHH	FFEEHHGG
$2d_{10}$:	AABBCCDD	CCDDAABB	EEFFGGHH	GGHHEEFF
$2d_{11}$:	AABBCCDD	DDCCBBAA	EEFFGGHH	HHGGFFEE
$2d_{12}$:	ABABCDCD	ABABCDCD	EFEFGHGH	EFEFGHGH
$2d_{13}$:	ABABCDCD	BABADCDC	EFEFGHGH	FEFEHGHG
$2d_{14}$:	ABABCDCD	CDCDABAB	EFEFGHGH	GHGHEFEF
$2d_{15}$:	ABABCDCD	DCDCBABA	EFEFGHGH	HGHGFEFE
$2d_{16}$:	ABBACDDC	ABBACDDC	EFFEGHHG	EFFEGHHG
$2d_{17}$:	ABBACDDC	BAABDCCD	EFFEGHHG	FEEFHGGH
$2d_{18}$:	ABBACDDC	CDDCABBA	EFFEGHHG	GHHGEFFE
$2d_{19}$:	ABBACDDC	DCCDBAAB	EFFEGHHG	HGGHFEEF
$2d_{20}$:	ABCDABCD	ABCDABCD	EFGHEFGH	EFGHEFGH
$2d_{21}$:	ABCDABCD	BADCBADC	EFGHEFGH	FEHGFEHG
$2d_{22}$:	ABCDABCD	CDABCDAB	EFGHEFGH	GHEFGHEF
$2d_{23}$:	ABCDABCD	DCBADCBA	EFGHEFGH	HGFEHGFE
$2d_{24}$:	ABCDBADC	ABCDBADC	EFGHFEHG	EFGHFEHG
$2d_{25}$:	ABCDBADC	BADCABCD	EFGHFEHG	FEHGEFGH
$2d_{26}$:	ABCDBADC	CDABDCBA	EFGHFEHG	GHEFHGFE
$2d_{27}$:	ABCDBADC	DCBACDAB	EFGHFEHG	HGFEGHEF
$2d_{28}$:	ABCDCDAB	ABCDCDAB	EFGHGHEF	EFGHGHEF
$2d_{29}$:	ABCDCDAB	BADCDCBA	EFGHGHEF	FEHGHGFE
$2d_{30}$:	ABCDCDAB	CDABABCD	EFGHGHEF	GHEFEFGH
$2d_{31}$:	ABCDCDAB	DCBABADC	EFGHGHEF	HGFEFEHG
$2d_{32}$:	ABCDDCBA	ABCDDCBA	EFGHHGFE	EFGHHGFE
$2d_{33}$:	ABCDDCBA	BADCCDAB	EFGHHGFE	FEHGGHEF
$2d_{34}$:	ABCDDCBA	CDABBADC	EFGHHGFE	GHEFFEHG
$2d_{35}$:	ABCDDCBA	DCBAABCD	EFGHHGFE	HGFEEFGH

Table 5. 2-dimensional symmetric classes

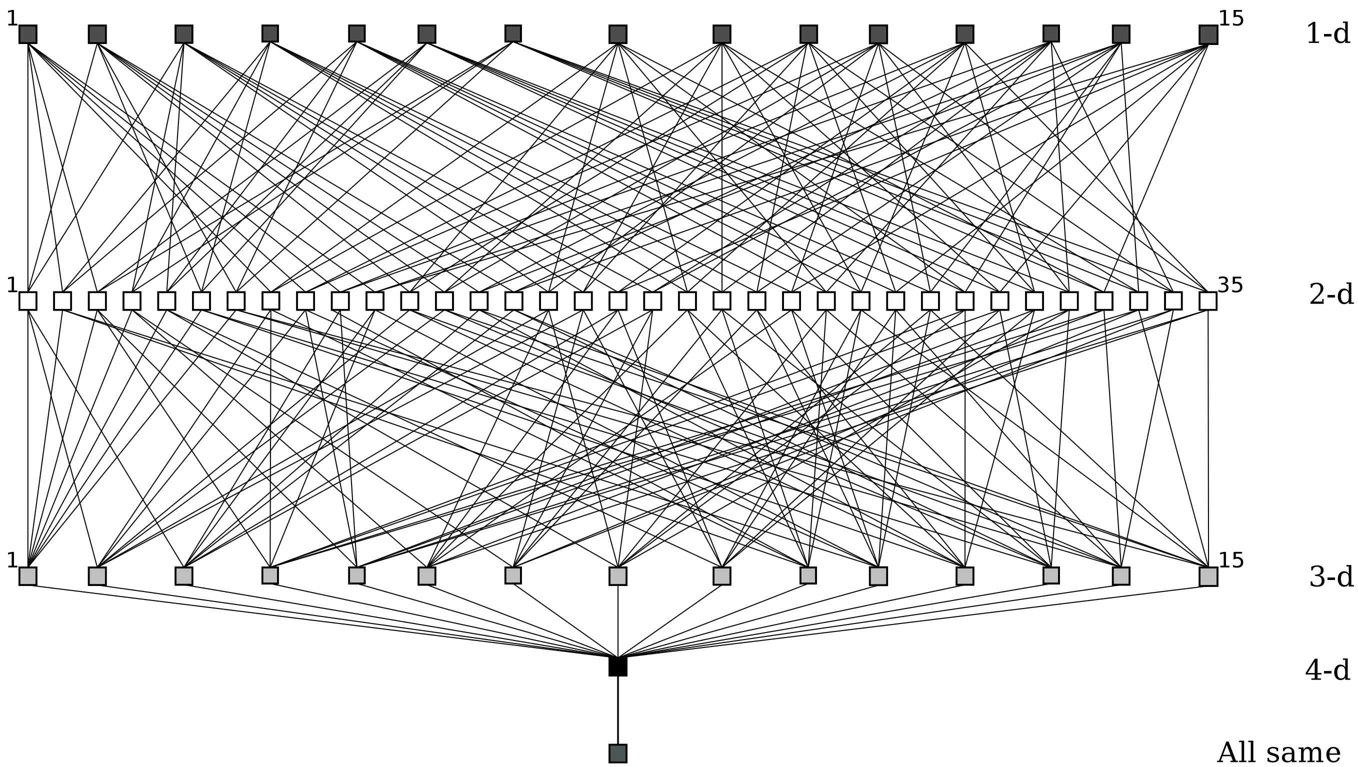

Fig. 1. Dimensional Hierarchy