Improved Linear Differential Attacks on CubeHash

Shahram Khazaei[1], Simon Knellwolf[2], Willi Meier[2], and Deian Stefan[3]

[1] EPFL, Switzerland
[2] FHNW, Switzerland
[3] The Cooper Union, USA

Abstract. This paper presents improved collision attacks on round-reduced variants of the hash function CubeHash, one of the SHA-3 second round candidates. We apply two methods for finding linear differential trails that lead to lower estimated attack complexities when used within the framework introduced by Brier, Khazaei, Meier and Peyrin at ASIACRYPT 2009. The first method yields trails that are relatively dense at the beginning and sparse towards the end. In combination with the condition function concept, such trails lead to much faster collision attacks. We demonstrate this by providing a real collision for CubeHash-5/96. The second method randomizes the search for highly probable linear differential trails and leads to significantly better attacks for up to eight rounds.

Keywords: hash function, differential attack, collision, linearization, SHA-3, CubeHash.

1 Introduction

Hash functions are important cryptographic primitives that transform arbitrary-length input messages into fixed-length message digests. They are used in many applications, notably in commitment schemes, digital signatures and message authentication codes. To this end they are required to satisfy different security properties, one of them is collision resistance. Informally, a hash function is collision resistant if it is practically infeasible to find two distinct messages m_1 and m_2 that produce the same message digest.

Chabaud and Joux [7] presented the first differential collision attack on SHA-0. Using a linearized model of the hash function, they found message differences that lead to a collision of the original hash function with a higher probability than the birthday bound. Similar strategies were later used by Rijmen and Oswald [12] on SHA-1 and by Indesteege and Preneel [8] on EnRUPT.

Pramstaller *et al.* [11] related the problem of finding highly probable linear differences to the problem of finding low weight codewords of a linear code. A recent work of Brier *et al.* [4] more precisely analyzed this relation for hash functions whose non-linear operations only consist in modular additions. They reformulate the problem of finding message pairs that conform to a linear differential trail to that of finding preimages of zero of a condition function. The

D.J. Bernstein and T. Lange (Eds.): AFRICACRYPT 2010, LNCS 6055, pp. 407–418, 2010.

search for such preimages is accelerated by the implicit use of message modification techniques. Given a linear differential trail, the concept further allows to estimate the corresponding complexity of the collision attack.

The success of the attack essentially depends on finding appropriate linear trails that lead to a low complexity of the attack. Such trails must not only have low weight, but the distribution of the weights along the trail must also be considered. A trail that is dense at the beginning and sparse towards the end yields a lower attack complexity than an arbitrary trail of comparable weight. This is due to freedom degrees use: initial conditions can be more easily satisfied than those towards the end of the trail.

Contribution of this Paper. We apply two different methods for finding appropriate trails for variants of the SHA-3 second round candidate CubeHash. For several round parameters r and message block sizes b we present better collision attacks on CubeHash-r/b than those presented so far. Specifically, we find collisions of CubeHash-5/96 and give a theoretical attack of CubeHash-8/96 with estimated complexity of 2^{80} compression function calls. This improves over the generic attack with complexity of about 2^{128} and is the first collision attack on more than seven rounds.

Previous Results on CubeHash. We refer to part 2. B. 5 of [2] for a complete survey of cryptanalytic results on CubeHash. The currently best collision attacks on CubeHash-r/b for message block sizes $b = 32, 64$ were presented in [4]. For $b = 32$ they present attacks of complexity $2^{54.1}$ and $2^{182.1}$ for four and six rounds, respectively. For $b = 64$ an attack of complexity 2^{203} for seven rounds is given. No collision attack for more than seven rounds was presented so far. Generic attacks are discussed by Bernstein in the appendix of [1].

Organization. Section 2 reviews the linearization framework and the concept of condition function presented in [4]. In Section 3 we describe the hash function CubeHash and define an appropriate compression function. Section 4 details how to find linear differential trails that, in combination with the concept of condition function, lead to successful collision attacks. In Section 5 we analyze CubeHash-5/96 and present a real collision. We conclude in Section 6.

2 Linearization Framework and Condition Function

In this section we fix notations and briefly review the general framework for collision attacks presented in [4] (see [5] for an extended version).

2.1 Fixed-Input-Length Compression Function

To any hash function we can attribute a fixed-input-length compression function $\mathsf{Compress} : \{0,1\}^m \rightarrow \{0,1\}^h$, with $m \geq h$, such that a collision for the compression function directly translates to a collision of the hash function[1] (e.g., we

[1] Note that this notion of compression function does not coincide with the frequently used compression function in the context of Merkle-Damgård and other iterated constructions.

can just restrict the domain of the hash function). We suppose that the only non-linear operations of the compression function consist of modular additions of w-bit words. For any input M to the compression function denote $\mathbf{A}(M)$ and $\mathbf{B}(M)$ the concatenation of all left, and, respectively right addends that are added in the course of the computation of $\mathsf{Compress}(M)$. Analogously define $\mathbf{C}(M)$ as the concatenation of the corresponding carry words. Thus, if n_a is the number of additions effected in the course of one evaluation of the compression function, each of $\mathbf{A}(M), \mathbf{B}(M)$ and $\mathbf{C}(M)$ contains $n_a w$ bits.

2.2 Linearization and Raw Probability

Let $\mathsf{Compress}_{\mathsf{lin}}$ be the linear function obtained by replacing all modular additions of $\mathsf{Compress}$ by XORs. For an input $\varDelta$ to this linearized compression function denote $\boldsymbol{\alpha}(\varDelta)$ and $\boldsymbol{\beta}(\varDelta)$ the concatenation of the left, and, respectively right addends that are XORed in the course of the computation of $\mathsf{Compress}_{\mathsf{lin}}(\varDelta)$, setting their most significant bits to zero[2].

We say that a message M *conforms to the trail of* $\varDelta$ if for all $i = 0, \ldots, n_a - 1$

$$((A^i \oplus \alpha^i) + (B^i \oplus \beta^i)) \oplus (A^i + B^i) = \alpha^i \oplus \beta^i,$$

where A^i, B^i, α^i and β^i respectively denote the ith w-bit word of $\mathbf{A}(M)$, $\mathbf{B}(M)$, $\boldsymbol{\alpha}(\varDelta)$ and $\boldsymbol{\beta}(\varDelta)$. According to Lemma 2 in [4], the probability that a randomly chosen M conforms to the trail of $\varDelta$ is given by

$$p_{\varDelta} = 2^{-\mathrm{wt}(\boldsymbol{\alpha}(\varDelta) \vee \boldsymbol{\beta}(\varDelta))},$$

where $\mathrm{wt}(\cdot)$ denotes the Hamming weight. If $\varDelta$ lies in the kernel of the linearized compression function, $p_{\varDelta}$ is a lower bound for the probability that the message pair $(M, M \oplus \varDelta)$ is a collision of the compression function. We call $p_{\varDelta}$ the *raw probability* of $\varDelta$ and $y = -\log_2(p_{\varDelta})$ the number of conditions imposed by $\varDelta$.

2.3 The Condition Function

Let $\varDelta$ be in the kernel of the linearized compression function. The condition function $\mathsf{Condition}_{\varDelta}$ has the same domain as the compression function, but outputs Y of lengths $y = \mathrm{wt}(\boldsymbol{\alpha}(\varDelta) \vee \boldsymbol{\beta}(\varDelta))$. To shorten the notation we omit the argument M to $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\varDelta$ to $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Additionally, we use subscripts to denote bit positions, e.g., $\mathbf{A}_i$ is the ith bit of $\mathbf{A}$. Let $i_0, \ldots, i_{y-1}$ be the bit positions of the y non-zero bits in $\boldsymbol{\alpha} \vee \boldsymbol{\beta}$. Define the condition function $Y = \mathsf{Condition}_{\varDelta}(M)$ by

$$Y_j = (\boldsymbol{\alpha}_{i_j} \oplus \boldsymbol{\beta}_{i_j})\mathbf{C}_{i_j} \oplus \boldsymbol{\alpha}_{i_j}\mathbf{B}_{i_j} \oplus \boldsymbol{\beta}_{i_j}\mathbf{A}_{i_j} \oplus \boldsymbol{\alpha}_{i_j}\boldsymbol{\beta}_{i_j} \text{ for } j = 0, \ldots, y-1.$$

By Proposition 1 in [4], the problem of finding a message M that conforms to the trail of $\varDelta$ is equivalent to the problem of finding M such that $\mathsf{Condition}_{\varDelta}(M) = 0$.

Suppose an ideal situation where we are given partitions $\bigcup_{i=1}^{\ell} \mathcal{M}_i = \{0, \ldots, m-1\}$ and $\bigcup_{i=0}^{\ell} \mathcal{Y}_i = \{0, \ldots, y-1\}$ such that for $j = 0, \ldots, \ell$ the output bits

[2] The most significant bits of each addition are linear.

with position indices in $\mathcal{Y}_j$ only depend on the input bits with position indices in $\bigcup_{i=1}^{j} \mathcal{M}_i$. Then we expect to find an M such that $\mathsf{Condition}_\Delta(M) = 0$ after

$$c_\Delta = \sum_{i=0}^{\ell} 2^{q_i}$$

evaluations of the condition function, where $q_i = |\mathcal{Y}_i| + \max(0, q_{i+1} - |\mathcal{M}_{i+1}|)$ for $i = \ell - 1, \ldots, 0$ and $q_\ell = |\mathcal{Y}_\ell|$. We call c_Δ the *theoretical complexity* of Δ.

We refer to [5] for a method to approximately determine such partitions and suitable probabilities 2^{-p_i} to keep track of the non-ideality of these partitions. Theoretical complexities in this paper are computed including the probabilities 2^{-p_i}. We further refer to [5] for instructions on implementing the search algorithm with negligible memory requirement.

3 Description of CubeHash

CubeHash [2] is a second-round candidate of the SHA-3 competition [10] of the National Institute of Standards and Technology. The function is designed with parameters r, b, and h which are the number of rounds, the number of bytes per message block, and the hash output length (in bits), respectively. We denote the parametrized function as CubeHash-r/b. The initial proposal of CubeHash-8/1 was tweaked to CubeHash-16/32 which is about 16 times faster and is now the official proposal for all digest lengths $h = 224, 256, 384$ or 512. Third-party cryptanalysis with larger values of b and fewer number of rounds r is explicitly encouraged.

3.1 Algorithm Specification

CubeHash operates on 32-bit words. It maintains a 1024-bit internal state X which is composed of 32 words $X_0, \ldots, X_{31}$. The algorithm is composed of five steps:

1. Initialize the state X to a specified value that depends on (r, b, h).
2. Pad the message to a sequence of b-byte input blocks.
3. For every b-byte input block:
   - XOR the block into the first b-bytes of the state.
   - Transform the state through r identical rounds.
4. Finalize the state: XOR 1 into X_{31} and transform the state through $10r$ identical rounds.
5. Output the first h bits of the state.

A round consists of the following steps:

- Add X_i into $X_{i \oplus 16}$, for $0 \le i \le 15$.
- Rotate X_i to the left by seven bits, for $0 \le i \le 15$.

- Swap X_i and $X_{i\oplus 8}$, for $0 \leq i \leq 7$.
- XOR $X_{i\oplus 16}$ into X_i, for $0 \leq i \leq 15$.
- Swap X_i and $X_{i\oplus 2}$, for $i \in \{16, 17, 20, 21, 24, 25, 28, 29\}$.
- Add X_i into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
- Rotate X_i to the left by eleven bits, for $0 \leq i \leq 15$.
- Swap X_i and $X_{i\oplus 4}$, for $i \in \{0, 1, 2, 3, 8, 9, 10, 11\}$.
- XOR $X_{i\oplus 16}$ into X_i, for $0 \leq i \leq 15$.
- Swap X_i and $X_{i\oplus 1}$, for $i \in \{16, 18, 20, 22, 24, 26, 28, 30\}$.

In this paper we consider the variants CubeHash-r/b with $b = 32, 64$ and 96, always assuming $h = 512$.

3.2 Defining the Compression Function

Following [4] we define a fixed-input-length compression function $\mathsf{Compress}$ for CubeHash. This compression function is parametrized by a 1024-bit initial value V and compresses t ($t \geq 1$) b-byte message blocks $M = M^0 \| \cdots \| M^{t-1}$. The output $H = \mathsf{Compress}(M, V)$ consists of the last $1024 - 8b$ bits of the internal state after tr round transformations processing M.

A colliding message pair $(M, M \oplus \Delta)$ for $\mathsf{Compress}$ directly extends to a collision of CubeHash by appending a pair of message blocks $(M^t, M^t \oplus \Delta^t)$ such that Δ^t erases the difference in the first $8b$ bits of the internal state. The difference Δ^t is called *erasing block difference.*

When searching for collisions of $\mathsf{Compress}$, the parameter V is not restricted to be the initial value of CubeHash. Specifically, V can be the state after processing some message prefix M^{pre}. Thus, a pair of colliding messages for the hash function then has the general form

$$(M^{\mathrm{pre}} \| M \| M^t \| M^{\mathrm{suff}}, M^{\mathrm{pre}} \| M \oplus \Delta \| M^t \oplus \Delta^t \| M^{\mathrm{suff}})$$

for an arbitrary message suffix M^{suff}.

4 Constructing Linear Differentials

We linearize the compression function of CubeHash to find message differences that can be used for a collision attack as described in Section 2. Specifically, we are interested in finding differences with low theoretical complexity. As a first approach one can search for differences with a high raw probability.

4.1 Searching for High Raw Probability

Let $\mathsf{Compress}_{\mathsf{lin}}$ be the linearization of $\mathsf{Compress}$ obtained by replacing all modular additions in the round transformation with XORs and setting $V = 0$. Using the canonical bases, $\mathsf{Compress}_{\mathsf{lin}}$ can be written as a matrix $\mathcal{H}$ of dimension $(1024 - 8b) \times 8bt$. Let τ be the dimension of its kernel. As noted in [4], the matrix $\mathcal{H}$ does not have full rank for many parameters r/b and t, and one can find differences with high a raw probability (imposing a small number of conditions) in the set of linear combinations of at most λ kernel basis vectors, where $\lambda \geq 1$ is chosen such that the set can be searched exhaustively. The results heavily depend on the choice of the kernel basis. Table 1 compares the minimal number of conditions for $\lambda = 3$ for two different choices of the kernel basis. The results in the first three rows are obtained using the same algorithm as in [4] to determine the bases. The results in the last three rows are obtained using the more standard procedure implemented for example in the Number Theory Library of Shoup [13].

Table 1. Minimal number of conditions found for $\lambda = 3$ using two different algorithms to determine the kernel bases

b/r	4	5	6	7	8	16
32	156	1244	400	1748	830	2150
64	130	205	351	447	637	1728
96	62	127	142	251	266	878
32	189	1952	700	2428	830	2150
64	189	1514	700	1864	637	1728
96	67	128	165	652	329	928

The inverse raw probability is an upper bound of the theoretical complexity, and as such, we expect that differences with a high raw probability have a low theoretic complexity. However, a higher raw probability does not always imply a lower theoretic complexity. There are differences with lower raw probability that lead to a lower theoretic complexity than that of a difference with a higher raw probability. Hence, when searching for minimal complexity of the collision attack, simply considering the number of conditions imposed by a difference is not sufficient.

4.2 Searching for Sparsity at the End

As previously observed in [3,7,9,11,14], conditions in early steps of the computation can be more easily satisfied than those in later steps. This is due to message modifications, (probabilistic) neutral bits, submarine modifications and other freedom degrees use. Similar techniques are used implicitly when using a dependency table to find a preimage of the condition function (and thus a collision for the compression function). This motivates the search for differences Δ such that $\boldsymbol{\alpha}(\Delta) \vee \boldsymbol{\beta}(\Delta)$ is sparse at the end. In general, however, this is not the case for trails found using the above method and, in contrast, most are sparse in the beginning and dense at the end. This is due to diffusion of the linearized compression function.

We note that the linearized round transformation of CubeHash is invertible and let $\mathsf{Compress}^{\mathsf{b}}_{\mathsf{lin}}$ be defined in the same way as $\mathsf{Compress}_{\mathsf{lin}}$ but with inverse linearized round transformations. Suppose that $\Delta' = \Delta^0 \| \cdots \| \Delta^{t-1}$ lies in the

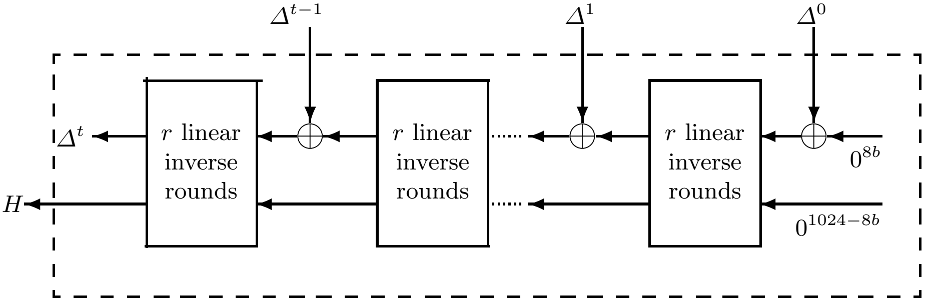
Δ^{t-1}
Δ^1
Δ^0
Δ^t
H
r linear inverse rounds
r linear inverse rounds
r linear inverse rounds
0^{8b}
$0^{1024-8b}$

Fig. 1. Computation of $\mathsf{Compress}^{\mathsf{b}}_{\mathsf{lin}}$ on input $\Delta' = \Delta^0\|\cdots\|\Delta^{t-1}$. If Δ' lies in the kernel, $H = 0$ and $\Delta = \Delta^t\|\cdots\|\Delta^1$ lies in the kernel of $\mathsf{Compress}_{\mathsf{lin}}$.

kernel of $\mathsf{Compress}^{\mathsf{b}}_{\mathsf{lin}}$ and Δ^t equals the (discarded) first $8b$ bits of the state after the tr linear inverse round transformations processing Δ' as shown in Fig. 1. Then, the difference $\Delta = \Delta^t\|\cdots\|\Delta^1$ lies in the kernel of $\mathsf{Compress}_{\mathsf{lin}}$ and Δ^0 is the corresponding erasing block difference. As for the linearized compression function we determine a basis of the kernel of $\mathsf{Compress}^{\mathsf{b}}_{\mathsf{lin}}$ and exhaustively search for linear combinations of at most λ kernel basis vectors of high raw probability. Due to the diffusion of the inverse transformation, these trails tend to be dense at the beginning and sparse at the end.

4.3 Randomizing the Search

The kernel of $\mathcal{H}$ contains 2^τ different elements. The above method finds the best difference out of a subset of $\sum_{i=1}^{\lambda}\binom{\tau}{i}$ elements. We may find better results by increasing λ or by repeating the search for another choice of the basis. Using ideas from [11] we propose an alternative search algorithm, that works well for many variants of CubeHash and does not decisively depend on the choice of the kernel basis.

Let $\Delta_0, \ldots, \Delta_{\tau-1}$ be a kernel basis of $\mathsf{Compress}_{\mathsf{lin}}$ and denote $\mathcal{G}$ the matrix whose τ rows consist of the binary vectors $\Delta_i\|\boldsymbol{\alpha}(\Delta_i)\|\boldsymbol{\beta}(\Delta_i)$ for $i = 0, \ldots, \tau-1$. Elementary row operations on $\mathcal{G}$ preserve this structure, that is, the rows always have the form $\Delta\|\boldsymbol{\alpha}(\Delta)\|\boldsymbol{\beta}(\Delta)$ where Δ lies in the kernel of $\mathsf{Compress}_{\mathsf{lin}}$ and its raw probability is given by the Hamming weight of $\boldsymbol{\alpha}(\Delta) \vee \boldsymbol{\beta}(\Delta)$. For convenience, we call this the raw probability of the row (instead of the raw probability of the first $|\Delta|$ bits of the row). Determine $i_{\max}$, the index of the row with the highest raw probability. Then iterate the following steps:

1. Randomly choose a column index j and let i be the smallest row index such that $\mathcal{G}_{i,j} = 1$ (choose a new j if no such i exists).
2. For all row indices $k = i+1, \ldots, \tau-1$ such that $\mathcal{G}_{k,j} = 1$:
   - add row i to row k,
   - set $i_{\max} = k$ if row k has higher raw probability than row $i_{\max}$.
3. Move row i to the bottom of $\mathcal{G}$, shifting up rows $i+1, \ldots, \tau-1$ by one.

Table 2. Minimal number of conditions found with the randomized search. Values in boldface improve over the values in Table 1.

b/r	4	5	6	7	8	16
32	156	1244	**394**	1748	830	2150
64	130	205	**309**	447	637	1728
96	**38**	127	**90**	251	**151**	**709**

Remark. A more specific choice of the column index j in the first step does not lead to better results. In particular, we tried to prioritize choosing columns towards the end, or for every chosen column in $\boldsymbol{\alpha}(\Delta)$ to also eliminate the corresponding column in $\boldsymbol{\beta}(\Delta)$.

Table 2 shows the best found raw probabilities after 200 trials of 600 iterations. Estimating the corresponding theoretic complexities as described in Section 2.3 yields the improved collision attacks presented in Table 3.

Table 3. Logarithmic theoretical complexities of improved collision attacks

b/r	4	5	6	7	8	16
32			180			
64			132			
96	7		51		80	

For CubeHash-r/b there is a generic collision attack with complexity of about 2^{512-4b}. For $b > 64$ this is faster than the generic birthday attack on hash functions with output length $h = 512$. For $b = 96$, specifically, the generic attack has a complexity of about 2^{128}. Our attacks clearly improve over this bound.

5 Collision for CubeHash-5/96

This section illustrates the findings of Section 4.2 and provides a collision for CubeHash-5/96.

5.1 Linear Differentials

We consider two linear differences found by the methods of Section 4.1 and 4.2 respectively. Both consist of two 96-byte blocks, a first block that lies in the kernel of the linearized compression function and a second one that is the corresponding erasing block difference. They are given by

$$\begin{aligned}
\mathit{\Delta}_a^0 = {}& \texttt{40000000 00000000 40000000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 00200000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 00000000 00000000 00000000 00000040}\\
& \texttt{00000000 00000040 00000000 00020000 00000000 00000000},\\
\mathit{\Delta}_a^1 = {}& \texttt{01000111 01000111 00000000 00000000 8008002A 00000000}\\
& \texttt{08000022 00000000 00000000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 11040000 00000000 40000101 01000111}\\
& \texttt{00000000 00000000 00002208 00000000 08002000 00000000}
\end{aligned}$$

and

$$\begin{aligned}
\mathit{\Delta}_b^0 = {}& \texttt{08000208 08000208 00000000 00000000 40000100 00000000}\\
& \texttt{00400110 00000000 00000000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 0800A000 00000000 08000888 08000208}\\
& \texttt{00000000 00000000 40011000 00000000 00451040 00000000},\\
\mathit{\Delta}_b^1 = {}& \texttt{80000000 00000000 80000000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 00400000 00000000 00000000 00000000}\\
& \texttt{00000000 00000000 00000000 00000000 00000000 00000080}\\
& \texttt{00000000 00000080 00000000 00040000 00000000 00000000}.
\end{aligned}$$

The number of conditions imposed by $\mathit{\Delta}_a$ and $\mathit{\Delta}_b$ are 127 and 134, respectively. Despite its lower raw probability, $\mathit{\Delta}_a$ has a theoretical complexity of $2^{31.9}$ compression function calls, which is much less than $2^{68.7}$ for $\mathit{\Delta}_b$. As discussed above, this is due to the different structure of $\boldsymbol{\alpha} \vee \boldsymbol{\beta}$. We recall that y denotes the Hamming weight of $\boldsymbol{\alpha} \vee \boldsymbol{\beta}$ and let

$$y_i = \sum_{k=32i}^{32(i+1)} \mathrm{wt}(\alpha^k \vee \beta^k).$$

That is, y_i is the number of conditions imposed by a difference at round i. Table 4 compares the values y_i for $\mathit{\Delta}_a$ and $\mathit{\Delta}_b$. The conditions imposed in the first two rounds can easily be satisfied by appropriate message modifications, and thus, do not significantly increase the complexity of the attack — contrary to the conditions imposed in the last two rounds.

Due to its low theoretical complexity, we can use $\mathit{\Delta}_b$ to find a collision and to confirm empirically the estimated theoretical complexity.

Table 4. Number of conditions per round theoretical complexities

	y_1	y_2	y_3	y_4	y_5	y	$\log_2(c_\Delta)$
a	14	17	23	30	43	127	68.7
b	44	36	25	17	12	134	31.9

Table 5. Partition sets corresponding to the trail Δ_b for CubeHash-5/96. Numbers in $\mathcal{M}_i$ are byte indices, whereas numbers in $\mathcal{Y}_i$ are bit indices.

i	$\mathcal{M}_i$	$\mathcal{Y}_i$	q_i
0	$\emptyset$	$\emptyset$	0.00
1	$\{2, 6, 66\}$	$\{1, 2\}$	2.00
2	$\{10, 1, 9, 14, 74, 5, 13, 65, 17, 70\}$	$\{5\}$	1.35
3	$\{73, 7, 16, 19, 18, 78, 25, 37, 41\}$	$\{23, 24\}$	2.00
4	$\{69, 77, 24, 33\}$	$\{21, 22\}$	2.00
5	$\{50, 89\}$	$\{12, 13\}$	2.00
6	$\{20, 27, 45, 88\}$	$\{11\}$	1.15
7	$\{57, 4\}$	$\{38\}$	1.00
8	$\{80\}$	$\{7, 8\}$	2.00
9	$\{38, 40, 81, 3, 28, 32\}$	$\{34\}$	1.24
10	$\{49\}$	$\{41\}$	1.00
11	$\{58\}$	$\{19, 20, 42, 43\}$	4.00
12	$\{91\}$	$\{16, 17\}$	2.00
13	$\{23, 34, 44, 83\}$	$\{29, 30\}$	2.07
14	$\{90\}$	$\{14\}$	1.07
15	$\{15, 26\}$	$\{15\}$	1.07
16	$\{36\}$	$\{37, 55\}$	2.31
17	$\{42, 46, 48\}$	$\{25, 26\}$	2.12
18	$\{56\}$	$\{18, 31, 40\}$	3.01
19	$\{59\}$	$\{48, 79\}$	2.00
20	$\{84, 92, 0\}$	$\{35\}$	1.00
21	$\{82\}$	$\{9, 10, 27, 28, 32, 33\}$	6.04
22	$\{31, 51\}$	$\{44, 56, 64\}$	3.03
23	$\{71\}$	$\{6\}$	1.00
24	$\{11, 54, 67\}$	$\{3\}$	1.00
25	$\{75\}$	$\{78\}$	1.00
26	$\{21, 55\}$	$\{46, 59\}$	2.00
27	$\{63\}$	$\{50\}$	1.00
28	$\{79\}$	$\{45, 49, 65, 70\}$	4.00
29	$\{12\}$	$\{71\}$	1.06
30	$\{22\}$	$\{58, 67, 81, 82, 83\}$	5.00
31	$\{29, 62\}$	$\{63\}$	1.03
32	$\{87, 95\}$	$\{53, 54, 74, 76, 85\}$	5.01
33	$\{39, 47\}$	$\{39\}$	1.01
34	$\{53, 8\}$	$\{69, 88, 89\}$	3.30
35	$\{30\}$	$\{77, 86, 94, 98\}$	5.04
36	$\{60, 61\}$	$\{62, 91, 101, 102\}$	4.35
37	$\{35, 52\}$	$\{61, 90, 103\}$	4.22
38	$\{43\}$	$\{36, 57, 60, 104, 111\}$	5.77
39	$\{64\}$	$\{0\}$	1.33
40	$\{68\}$	$\{4\}$	2.03
41	$\{72\}$	$\{97, 100, 121\}$	8.79
42	$\{76\}$	$\{66, 80, 92, 93\}$	13.39
43	$\{85\}$	$\{47, 112\}$	16.92
44	$\{93\}$	$\{51, 52, 68, 72, 75, 87, 95\}$	22.91
45	$\{86, 94\}$	$\{73, 84, 96, 99, 105, \ldots, 110, 113, \ldots, 132, 133\}$	31.87

5.2 Finding the Collision

According to Section 3.2, we construct a collision for the hash function out of a collision for the compression function. Using a dependency table at byte-level, we obtained a partition of the condition function attributed to Δ_b^0 (see Table 5). Then, using the tree-based backtracking algorithm proposed in [4], we found several collisions after $2^{22.41}$ to $2^{32.25}$ condition function calls. One of them, found after $2^{29.1}$ condition function calls, is given by

$$\begin{aligned}
M^{\mathrm{pre}} = {} & \texttt{F06BB068 487C5FE1 CCCABA70 0A989262 801EDC3A 69292196} \\
& \texttt{8848F445 B8608777 C037795A 10D5D799 FD16C037 A52D0B51} \\
& \texttt{63A74C97 FD858EEF 7809480F 43EB264C D6631863 2A8CCFE2} \\
& \texttt{EA22B139 D99E4888 8CA844FB ECCE3295 150CA98E B16B0B92}, \\
M^0 = {} & \texttt{3DB4D4EE 02958F57 8EFF307A 5BE9975B 4D0A669E E6025663} \\
& \texttt{8DDB6421 BAD8F1E4 384FE128 4EBB7E2A 72E16587 1E44C51B} \\
& \texttt{DA607FD9 1DDAD41F 4180297A 1607F902 2463D259 2B73F829} \\
& \texttt{C79E766D 0F672ECC 084E841B FC700F05 3095E865 8EEB85D5}.
\end{aligned}$$

For $M^1 = 0$, the messages $M^{\mathrm{pre}} \| M^0 \| M^1$ and $M^{\mathrm{pre}} \| M^0 \oplus \Delta_b^0 \| M^1 \oplus \Delta_b^1$ collide to the same digest

$$\begin{aligned}
H = {} & \texttt{C2E51517 C503746E 46ECD6AD 5936EC9B} \\
& \texttt{FF9B74F9 2CEA4506 624F2B0B FE584D2C} \\
& \texttt{56CD3E0E 18853BA8 4A9D6D38 F1F8E45F} \\
& \texttt{2129C678 CB3636D4 D865DE13 410E966C}
\end{aligned}$$

under CubeHash-5/96. Instead of $M^1 = 0$, any other M^1 can be chosen and, moreover, the colliding messages can be extended by an arbitrary message suffix M^{suff}.

6 Conclusion

In this paper we used two methods for finding improved linear differential trails for CubeHash. The method of backward computation lead to the first practical collision attack on CubeHash-5/96. The randomized search yielded new highly probable differential trails which lead to improved collision attacks for up to eight rounds. Both methods may also apply to collision attacks on other hash functions.

Our analysis did not lead to an attack on the official CubeHash-16/32.

Acknowledgements

This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The second author is supported by the Hasler Foundation www.haslerfoundation.ch under project number 08065. The third author is supported by GEBERT RÜF STIFTUNG under project number GRS-069/07.

References

1. Bernstein, D.J.: Cubehash. Submission to NIST (2008)
2. Bernstein, D.J.: Cubehash. Submission to NIST, Round 2 (2009)
3. Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 290–305. Springer, Heidelberg (2004)
4. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization Framework for Collision Attacks: Application to CubeHash and MD6. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 560–577. Springer, Heidelberg (2009)
5. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization Framework for Collision Attacks: Application to CubeHash and MD6 (extended version). Cryptology ePrint Archive, Report 2009/382 (2009), http://eprint.iacr.org
6. Brier, E., Peyrin, T.: Cryptanalysis of CubeHash. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 354–368. Springer, Heidelberg (2009)
7. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
8. Indesteege, S., Preneel, B.: Practical Collisions for EnRUPT. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 246–259. Springer, Heidelberg (2009)
9. Naito, Y., Sasaki, Y., Shimoyama, T., Yajima, J., Kunihiro, N., Ohta, K.: Improved Collision Search for SHA-0. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 21–36. Springer, Heidelberg (2006)
10. National Institute of Standards and Techonolgy. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithms (SHA-3) Family. Federal Register, 72 (2007)
11. Pramstaller, N., Rechberger, C., Rijmen, V.: Exploiting coding theory for collision attacks on SHA-1. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 78–95. Springer, Heidelberg (2005)
12. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 58–71. Springer, Heidelberg (2005)
13. Shoup, V.: NTL: A Library for doing Number Theory. Version 5.5.2, http://www.shoup.net/ntl
14. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)