Cryptanalysis of CubeHash

Eric Brier and Thomas Peyrin

Ingenico, France
eric.brier@ingenico.com
thomas.peyrin@ingenico.com

Abstract. CubeHash is a family of hash functions submitted by Bernstein as a SHA-3 candidate. In this paper, we provide two different cryptanalysis approaches concerning its collision resistance. Thanks to the first approach, related to truncated differentials, we computed a collision for the CubeHash-1/36 hash function, i.e. when for each iteration 36 bytes of message are incorporated and one call to the permutation is applied. Then, the second approach, already used by Dai, much more efficient and based on a linearization of the scheme, allowed us to compute a collision for the CubeHash-2/4 hash function. Finally, a theoretical collision attack against CubeHash-2/3, CubeHash-4/4 and CubeHash-4/3 is described. This is currently by far the best known cryptanalysis result on this SHA-3 candidate.

Key words: hash functions, CubeHash, collision.

1 Introduction

A cryptographic hash function is a very important tool in cryptography, used in many applications such as digital signatures, authentication schemes or message integrity. One of its main and most important security feature is the collision resistance: finding two messages M and M' leading to the same hash value should require at least $2^{n/2}$ operations, where n is the output length of the hash function. Wang *et al.* [14, 16, 15, 17] recently showed that most standardized hash functions (e.g. MD5 [12] or SHA-1 [10]) are not collision resistant. As a response, the National Institute of Standards and Technology (NIST) opened a public competition [9] to develop a new cryptographic hash algorithm that will be called SHA-3. 51 submissions met the minimum submission requirements, and had been accepted as the first round candidates. Among them, CubeHash is a new hash function designed by Bernstein [3] and currently under evaluation. One of its advantages is its simplicity of description which makes the analysis much easier for a cryptanalyst. This proposal can be considered as a stream-cipher oriented hash function. It maintains a big 1024-bit internal state, in which b-byte of message are incorporated at each iteration. After adding such a message block, a fixed permutation is then applied r time. Generally, the bigger is r (or the smaller is b) the harder it should be for the attacker to break the collision resistance of CubeHash-r/b.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

Previous results. The first analysis of CubeHash was proposed by Aumasson *et al.* [1, 2] in which the authors showed some non-random properties for several versions of CubeHash as well as a collision for CubeHash-2/120. Later, thanks to a linearization approach, Dai [6] computed a collision for CubeHash-1/45 and CubeHash-2/89. Those results were soon improved to CubeHash-2/12 [7].

Our contributions. In this paper, we provide two different cryptanalysis approaches concerning the collision resistance. The first approach, related to truncated differences, allowed us to compute a real collision for CubeHash-1/36. The second approach, i.e. linearizing the scheme, gives us much better results: we provide a real collision for CubeHash-2/4 and theoretical collision attacks against CubeHash-2/3, CubeHash-4/4 and CubeHash-4/3. This is currently the best known cryptanalysis result on CubeHash. To give an insight of the broken schemes, CubeHash-4/3 speed is about 26 cycles/byte, comparable to the speed of SHA-2 [10].

2 Description of CubeHash

We refer to the specifications of CubeHash [3] for the complete description of the scheme. CubeHash-r/b-h is the h-bit output version of CubeHash, for which b bytes of message are incorporated and r calls to the internal permutation F is applied at each iteration. The function can hash messages up to $2^{128} - 1$ bits. After an appropriate padding, the message is therefore divided into block M_i of b bytes each. A 1024-bit internal state is maintained, divided into 32 words X_i of 32 bits each and initialized by the following process: set the first three state words X_0, X_1, X_2 to the integers $h/8$, b, r respectively and set the remaining state words to 0. Then apply on the state $10 \times r$ times the internal permutation F.

After the initialization, the message blocks are treated iteratively: exclusive or the incoming b-byte message block onto the first b bytes of the internal state and apply the iteration function, composed of r times the application of the internal permutation F. When all the message blocks have been treated, exclusive or the integer 1 to the state word X_{31} and apply $10 \times r$ times the internal permutation F without incorporating message blocks anymore. Finally, output the h first bits of the internal state.

The internal permutation F uses very simple operations: rotation, exclusive or, modular addition and word swapping. It is composed of 10 steps (represented graphically in Figure 1):

1. Add X_i into $X_{i \oplus 16}$, for $0 \leq i \leq 15$.
2. Rotate X_i on the left by seven bits, for $0 \leq i \leq 15$.
3. Swap X_i and $X_{i \oplus 8}$, for $i \in [0, 1, 2, 3, 4, 5, 6, 7]$.
4. Xor $X_{i \oplus 16}$ into X_i, for $0 \leq i \leq 15$.
5. Swap X_i and $X_{i \oplus 2}$, for $i \in [16, 7, 20, 21, 24, 25, 28, 29]$.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

6. Add X_i into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
7. Rotate X_i on the left by eleven bits, for $0 \leq i \leq 15$.
8. Swap X_i and $X_{i\oplus 4}$, for $i \in [0, 1, 2, 3, 8, 9, 10, 11]$.
9. Xor $X_{i\oplus 16}$ into X_i, for $0 \leq i \leq 15$.
10. Swap X_i and $X_{i\oplus 1}$, for $i \in [16, 18, 20, 22, 24, 26, 28, 30]$.

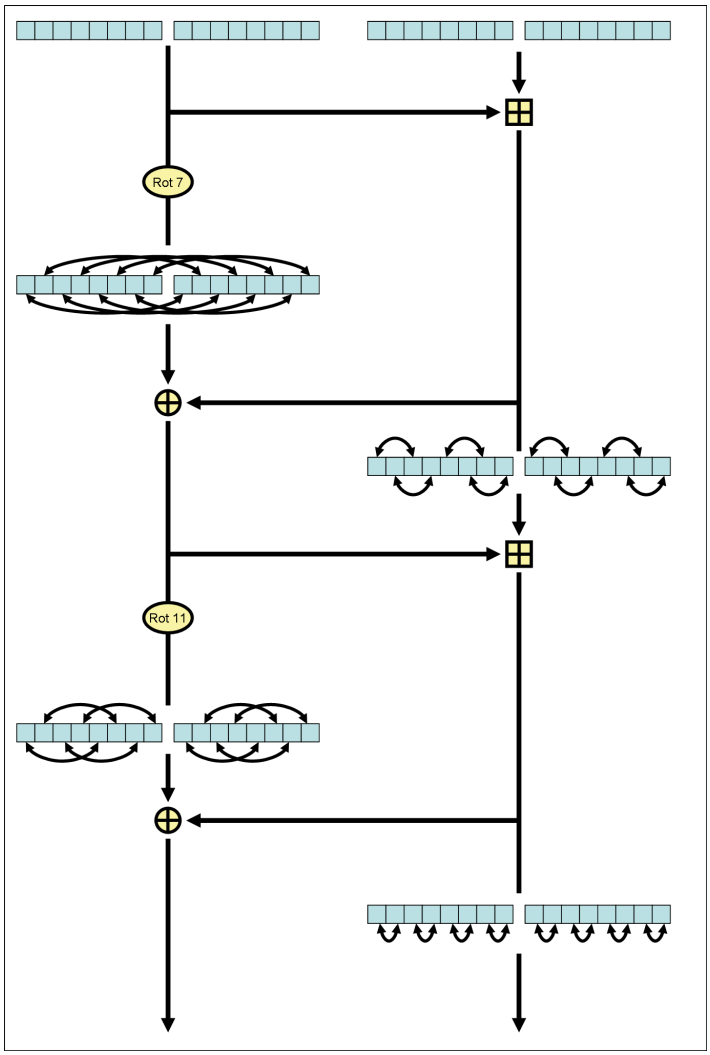

Fig. 1. Internal permutation F in `CubeHash`. Each cell represents a 32-bit word.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

3 Truncated differential paths

In this section, we depict a cryptanalysis approach for `CubeHash` regarding its collision resistance, based on two very generic differential paths. The technique is related to truncated differences, originally introduced by Knudsen [8] to cryptanalyze block ciphers and later utilized by Peyrin [11] in the hash functions setting. These one-round differential paths can potentially be used to build more complex differential characteristics on any number of rounds per iteration. Our results provide theoretical collision attacks for `CubeHash`-$2/b$ with $r \leq 2$ and $b \geq 36$ and as a proof of concept, we computed a collision for the 512-bit version of `CubeHash`-$1/36$.

3.1 The differential paths

In the following, we say that a 32-bit word is *active* when a non-zero difference exists on this word. We denote a differential path for the internal permutation by $A \longmapsto B$, where A and B are 32-bit words for which each bit represents one internal word (the LSB will denote X_{31} and the MSB will denote X_0). Said in other words, we only check if the internal words are active or not, whatever are the values of the non-zero differences. For example, `0x05000000` $\longmapsto$ `0x00000800` means that we have X_5 and X_7 active on the input, and that we expect only X_{20} to be active on the output of the permutation.

In this paper, we use two distinct differential paths for the internal permutation (see also Figure 2):

$$\Delta_1 \ : \ \texttt{0xa0800000} \longmapsto \texttt{0x0a020000}$$

$$\Delta_2 \ : \ \texttt{0x0a020000} \longmapsto \texttt{0xa0800000}$$

One can see that the two-round differential path composed of Δ_1 and Δ_2 has the interesting property that the input and output active words patterns are the same. In the difference mask `0xa0800000`, only the nine first 32-bit words of the internal state are active. Therefore, we can look for an attack on `CubeHash`-r/36 by using the differential paths Δ_1 and Δ_2 successively. In this section, we consider that the attacker insert 9 32-bit words of message at each iteration, i.e. he has full control on $X_0, \ldots, X_8$ at the beginning of each iteration.
We denote by $X^{\lll y}$ the rotation of y positions on the left applied to the word X and by $\overline{X}$ the complement value of X. Also, X' will represent the second member of the pair when X is an active word. For each differential path Δ_1 and Δ_2, a system of four equations on 32-bit words must be satisfied:

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

System 1 (for Δ_1) :

$$\begin{aligned}
(X_{24} + X_8) \oplus X_0^{\lll 7} &= (X_{24} + X'_8) \oplus X'^{\lll 7}_0 \\
X_8 + [(X_{26} + X_{10}) \oplus X_2^{\lll 7}] &= X'_8 + [(X_{26} + X_{10}) \oplus X'^{\lll 7}_2] \\
X_0 + [(X_{18} + X_2) \oplus X_{10}^{\lll 7}] &= X'_0 + [(X_{18} + X'_2) \oplus X_{10}^{\lll 7}] \\
X_2 + [(X_{16} + X_0) \oplus X_8^{\lll 7}] &= X'_2 + [(X_{16} + X'_0) \oplus X'^{\lll 7}_8]
\end{aligned}$$

System 2 (for Δ_2) :

$$\begin{aligned}
(X_{30} + X_{14}) \oplus X_6^{\lll 7} &= (X_{30} + X'_{14}) \oplus X'^{\lll 7}_6 \\
X_{14} + [(X_{28} + X_{12}) \oplus X_4^{\lll 7}] &= X'_{14} + [(X_{28} + X_{12}) \oplus X'^{\lll 7}_4] \\
X_6 + [(X_{20} + X_4) \oplus X_{12}^{\lll 7}] &= X'_6 + [(X_{20} + X'_4) \oplus X_{12}^{\lll 7}] \\
X_4 + [(X_{22} + X_6) \oplus X_{14}^{\lll 7}] &= X'_4 + [(X_{22} + X'_6) \oplus X'^{\lll 7}_{14}]
\end{aligned}$$

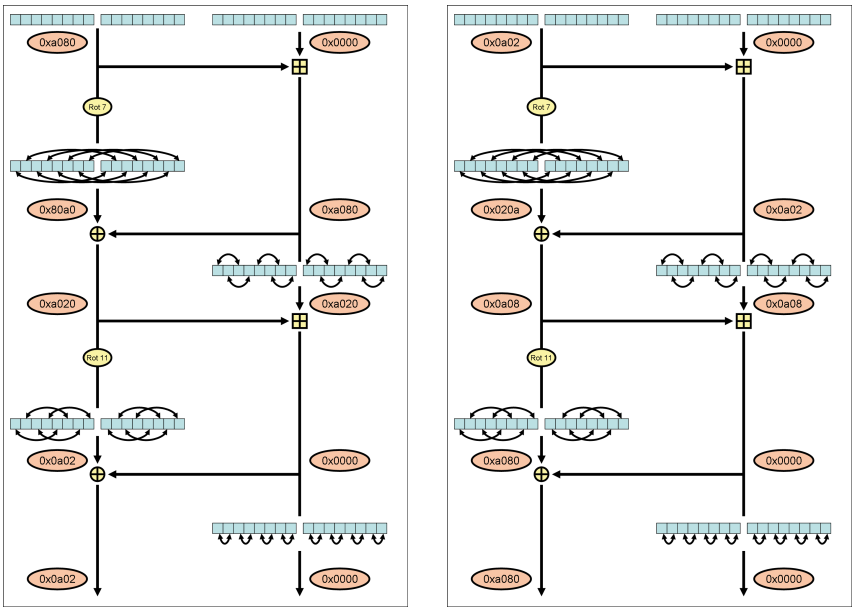

Fig. 2. Differential paths Δ_1 (left) and Δ_2 (right) for the internal permutation of `CubeHash`. A cell stands for a 32-bit word of the internal state and the boxes represent in hexadecimal display the active words during the computation.

We describe here a fast method to resolve the first system of equations. Exactly the same method will also apply for the second system, so we will only

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

focus on the first one. The internal state words X_{10}, X_{16}, X_{18}, X_{24} and X_{26} are given inputs of the system. The words X_0, X'_0, X_2, X'_2, X_8 and X'_8 are fully controlled by the attacker. Our method directly finds a solution for the three first equations. We will repeat this process several times until the last equation is also verified. More precisely, if we assume the equations to be independent, we will repeat the process 2^{32} times on average in order to get a solution for the complete system.

We now describe how to solve the three first equations of the system. First, we pick random values for X_2 and X'_2, such that $X_2 \neq X'_2$. We can rewrite the second and third equations respectively, so we directly get

$$X'_8 - X_8 = A \tag{1}$$
$$X'_0 - X_0 = B \tag{2}$$

where A and B are constant terms. We then set $Y = X_8 + X_{24}$ and $Y' = X'_8 + X_{24}$, so we can rewrite the first equation as:

$$Y \oplus X_0^{\lll 7} = Y' \oplus X'^{\lll 7}_0. \tag{3}$$

Finally, we combine the three equations together:

$$Y \oplus (A + Y) = X_0^{\lll 7} \oplus (B + X_0)^{\lll 7}. \tag{4}$$

We need a trick to solve this equation quickly: one can check that $x \oplus (x+k)$ is always equal to `0xffffffff` when $x = \overline{k}/2$ and when the least significant bit of k is equal to one. Thus, we wait for A and B values so that their least significant bit is equal to one and we set $Y = \overline{A}/2$ and $X_0 = \overline{B}^{\lll 25}/2$. The two sides of the equations are therefore equal to `0xffffffff` and we can finally deduce a solution that verifies the three first equations of our system.

3.2 A collision for `CubeHash-1/36`

Using the differential paths Δ_1 and Δ_2 and the solving technique previously explained, we computed within a few minutes on an average PC (Processor Intel Core 2 Duo 2.0 GHz, with 2 GB of RAM) a collision for `CubeHash`-1/36 with 512 bits of output.

The collision attack uses four message inputs. The first message block is used without any difference in order to randomize the values of the internal state just after the initialisation (randomization of the equations systems). Then, the second message pair verifies the differential path Δ_1 and the third message pair the differential path Δ_2. Finally, the fourth and last message pair will erase the remaining differences (`0xa0800000`) in the internal state, and thus leads to an internal collision. Obviously, by adding some other message words without

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

difference, one will maintain colliding pairs of internal states until the end of the computation of the hash function.

The values of the message words to insert and the final hash value are given in Table 4 in the Appendix A.

3.3 Extensions to other versions

One can easily extend this practical attack against `CubeHash`-1/36 to theoretical attacks against stronger versions. First, it is obvious that when $b \geq 36$, our attack remains valid. When $r = 2$, one has to verify equations from system 1 in the first internal permutation call, and equations from system 2 in the second internal permutation call of the iteration function.

We managed to generate input blocks verifying directly five equations among the eight of the two systems. Thus, one can assume that the three other equations will be verified with probability $2^{-32\times 3}$. Therefore, one can find a collision for `CubeHash`-2/36 with a computation complexity of about 2^{96} function calls.

4 Linear differential paths

In this section, we will try to find interesting differential paths by using a $\mathcal{F}^2$-linear version of the scheme. More precisely, in the internal permutation F, we replace all the modular additions by bitwise XOR operations on 32-bit words. Overall, this simplification makes sense since non-linear components are quite few in `CubeHash`. This technique is very frequently used in order to study hash functions, for example in the case of `SHA-1` [4]. Dai [6, 7] also used this simplification to compute a collision for `CubeHash`-2/12 and `CubeHash`-1/45.

4.1 The differential path

In this section, we are interested in attacking `CubeHash`-2/4. Said in other words, after a 2-round iteration, the attacker hash complete control over X_0 (and only X_0). The linear differential path we use is very simple, almost the simplest someone can think of. We first insert a one-bit perturbation in X_0 and apply the first 2-round iteration. We then erase all the differences present in X_0 and we apply the second 2-round iteration. Finally, we get a collision on the internal state by erasing a very last difference in X_0.

More precisely, we first insert a message pair with a one-bit difference located at bit position y. Then, the next message pair will correct three differences located at bit position $y + 4$, $y + 14$ and $y + 22$ (the positions are to be taken modulo 32). Finally, the last message pair will correct the last difference located at bit position $y + 4$. The differential path is described in Figure 3.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

it.	round	step	active bits	nb. cond.
1	1	M	M_1^0	
1	1		X_0^0	
1	1	1	$x_0^0,\ x_{16}^0$	1
1	1	2	$x_0^7,\ x_{16}^0$	
1	1	3	$x_8^7,\ x_{16}^0$	
1	1	4	$x_0^0,\ x_8^7,\ x_{16}^0$	
1	1	5	$x_0^0,\ x_8^7,\ x_{18}^0$	
1	1	6	$x_0^0,\ x_8^7,\ x_{16}^0,\ x_{18}^0,\ x_{24}^7$	3
1	1	7	$x_0^{11},\ x_8^{18},\ x_{16}^0,\ x_{18}^0,\ x_{24}^7$	
1	1	8	$x_4^{11},\ x_{12}^{18},\ x_{16}^0,\ x_{18}^0,\ x_{24}^7$	
1	1	9	$x_0^0,\ x_2^0,\ x_4^{11},\ x_8^7,\ x_{12}^{18},\ x_{16}^0,\ x_{18}^0,\ x_{24}^7$	
1	1	10	$x_0^0,\ x_2^0,\ x_4^{11},\ x_8^7,\ x_{12}^{18},\ x_{17}^0,\ x_{19}^0,\ x_{25}^7$	
1	2	1	$x_0^0,\ x_2^0,\ x_4^{11},\ x_8^7,\ x_{12}^{18},\ x_{16}^0,\ x_{17}^0,\ x_{18}^0,\ x_{19}^0,\ x_{20}^{11},\ x_{24}^7,\ x_{25}^7,\ x_{28}^{18}$	8
1	2	2	$x_0^7,\ x_2^7,\ x_4^{18},\ x_8^{14},\ x_{12}^{25},\ x_{16}^0,\ x_{17}^0,\ x_{18}^0,\ x_{19}^0,\ x_{20}^{11},\ x_{24}^7,\ x_{25}^7,\ x_{28}^{18}$	
1	2	3	$x_0^{14},\ x_4^{25},\ x_8^7,\ x_{10}^7,\ x_{12}^{18},\ x_{16}^0,\ x_{17}^0,\ x_{18}^0,\ x_{19}^0,\ x_{20}^{11},\ x_{24}^7,\ x_{25}^7,\ x_{28}^{18}$	
1	2	4	$x_0^0,\ x_0^{14},\ x_1^0,\ x_2^0,\ x_3^0,\ x_4^{11},\ x_4^{25},\ x_9^7,\ x_{10}^7,\ x_{16}^0,\ x_{17}^0,\ x_{18}^0,\ x_{19}^0,\ x_{20}^{11},\ x_{24}^7,\ x_{25}^7,\ x_{28}^{18}$	
1	2	5	$x_0^0,\ x_0^{14},\ x_1^0,\ x_2^0,\ x_3^0,\ x_4^{11},\ x_4^{25},\ x_9^7,\ x_{10}^7,\ x_{16}^0,\ x_{17}^0,\ x_{18}^0,\ x_{19}^0,\ x_{22}^{11},\ x_{26}^7,\ x_{27}^7,\ x_{30}^{18}$	
1	2	6	$x_0^0,\ x_0^{14},\ x_1^0,\ x_2^0,\ x_3^0,\ x_4^{11},\ x_4^{25},\ x_9^7,\ x_{10}^7,\ x_{16}^{14},\ x_{20}^{11},\ x_{20}^{25},\ x_{22}^{11},\ x_{25}^7,\ x_{27}^7,\ x_{30}^{18}$	12
1	2	7	$x_0^{11},\ x_0^{25},\ x_1^{11},\ x_2^{11},\ x_3^{11},\ x_4^{22},\ x_4^4,\ x_9^{18},\ x_{10}^{18},\ x_{16}^{14},\ x_{20}^{11},\ x_{20}^{25},\ x_{22}^{11},\ x_{25}^7,\ x_{27}^7,\ x_{30}^{18}$	
1	2	8	$x_0^4,\ x_0^{22},\ x_4^{11},\ x_4^{25},\ x_5^{11},\ x_6^{11},\ x_7^{11},\ x_{13}^{18},\ x_{14}^{18},\ x_{16}^{14},\ x_{20}^{11},\ x_{20}^{25},\ x_{22}^{11},\ x_{25}^7,\ x_{27}^7,\ x_{30}^{18}$	
1	2	9	$x_0^4,\ x_0^{14},\ x_0^{22},\ x_5^{11},\ x_7^{11},\ x_9^7,\ x_{11}^7,\ x_{13}^{18},\ x_{16}^{14},\ x_{20}^{11},\ x_{20}^{25},\ x_{22}^{11},\ x_{25}^7,\ x_{27}^7,\ x_{30}^{18}$	
1	2	10	$x_0^4,\ x_0^{14},\ x_0^{22},\ x_5^{11},\ x_7^{11},\ x_9^7,\ x_{11}^7,\ x_{13}^{18},\ x_{17}^{14},\ x_{21}^{11},\ x_{21}^{25},\ x_{23}^{11},\ x_{24}^7,\ x_{26}^7,\ x_{31}^{18}$	
2	1	M	$M_2^4,\ M_2^{14},\ M_2^{22}$	
2	1		$x_5^{11},\ x_7^{11},\ x_9^7,\ x_{11}^7,\ x_{13}^{18},\ x_{17}^{14},\ x_{21}^{11},\ x_{21}^{25},\ x_{23}^{11},\ x_{24}^7,\ x_{26}^7,\ x_{31}^{18}$	
2	1	1	$x_5^{11},\ x_7^{11},\ x_9^7,\ x_{11}^7,\ x_{13}^{18},\ x_{17}^{14},\ x_{21}^{25},\ x_{24}^7,\ x_{25}^7,\ x_{26}^7,\ x_{27}^7,\ x_{29}^{18},\ x_{31}^{18}$	10
2	1	2	$x_5^{18},\ x_7^{18},\ x_9^{14},\ x_{11}^{14},\ x_{13}^{25},\ x_{17}^{14},\ x_{21}^{25},\ x_{24}^7,\ x_{25}^7,\ x_{26}^7,\ x_{27}^7,\ x_{29}^{18},\ x_{31}^{18}$	
2	1	3	$x_1^{14},\ x_3^{14},\ x_5^{25},\ x_{13}^{18},\ x_{15}^{18},\ x_{17}^{14},\ x_{21}^{25},\ x_{24}^7,\ x_{25}^7,\ x_{26}^7,\ x_{27}^7,\ x_{29}^{18},\ x_{31}^{18}$	
2	1	4	$x_3^{14},\ x_8^7,\ x_9^7,\ x_{10}^7,\ x_{11}^7,\ x_{17}^{14},\ x_{21}^{25},\ x_{24}^7,\ x_{25}^7,\ x_{26}^7,\ x_{27}^7,\ x_{29}^{18},\ x_{31}^{18}$	
2	1	5	$x_3^{14},\ x_8^7,\ x_9^7,\ x_{10}^7,\ x_{11}^7,\ x_{19}^{14},\ x_{23}^{25},\ x_{24}^7,\ x_{25}^7,\ x_{26}^7,\ x_{27}^7,\ x_{29}^{18},\ x_{31}^{18}$	
2	1	6	$x_3^{14},\ x_8^7,\ x_9^7,\ x_{10}^7,\ x_{11}^7,\ x_{23}^{25},\ x_{29}^{18},\ x_{31}^{18}$	8
2	1	7	$x_3^{25},\ x_8^{18},\ x_9^{18},\ x_{10}^{18},\ x_{11}^{18},\ x_{23}^{25},\ x_{29}^{18},\ x_{31}^{18}$	
2	1	8	$x_7^{25},\ x_{12}^{18},\ x_{13}^{18},\ x_{14}^{18},\ x_{15}^{18},\ x_{23}^{25},\ x_{29}^{18},\ x_{31}^{18}$	
2	1	9	$x_{12}^{18},\ x_{14}^{18},\ x_{23}^{25},\ x_{29}^{18},\ x_{31}^{18}$	
2	1	10	$x_{12}^{18},\ x_{14}^{18},\ x_{22}^{25},\ x_{28}^{18},\ x_{30}^{18}$	
2	2	1	$x_{12}^{18},\ x_{14}^{18},\ x_{22}^{25}$	3
2	2	2	$x_{12}^{25},\ x_{14}^{25},\ x_{22}^{25}$	
2	2	3	$x_4^{25},\ x_6^{25},\ x_{22}^{25}$	
2	2	4	$x_4^{25},\ x_{22}^{25}$	
2	2	5	$x_4^{25},\ x_{20}^{25}$	
2	2	6	x_4^{25}	1
2	2	7	x_4^4	
2	2	8	x_0^4	
2	2	9	x_0^4	
2	2	10	x_0^4	
3	1	M	M_3^4	

Fig. 3. Linear differential path for `CubeHash`-2/4 and `CubeHash`-2/3. The three first columns give in order the iteration number, the round number and the step number in the internal permutation. A step denoted M represents the active bits of the message block inserted. The fourth column provides the active bits, where X_i^j denotes the j-th bit of the internal word X_i. Finally, the number of conditions is given in the last column.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

In the linear model, a collision can be computed directly. However, as any differential path, for the real function there is a certain probability of success that a random message pairs fulfilling the input differential constraints will also fulfill the output differential constraints. This probability can often be translated to a set of sufficient conditions to verify. In our case, the conditions directly depict the expected linear behavior of the scheme. The real `CubeHash` uses two modular additions in each call of the internal permutation F, which are non-linear components because of the carry potentially created. Thus, two interesting situations can occur:

1. a perturbation at a certain bit position is added to another bit containing no difference (move).
2. a perturbation at a certain bit position is added to another bit containing a difference (correction).

Each such situation will lead to exactly one sufficient condition because we want the modular addition carry to be unaffected by the perturbation. Said in other words, when we add together two word pairs A, B and A', B', the number of conditions will be the hamming weight of $(A \oplus A') \vee (B \oplus B')$, where $\vee$ stands for the bitwise or operation. When all the conditions from situations 1 and 2 are verified through the entire differential path, we are assured that the scheme behaved completely linearly regarding the bit perturbations and we can finally get our collision.

In total, the differential path presented possesses 46 sufficient bit conditions (24 for the first iteration and 22 for the second). This means that by testing about 2^{46} different messages pairs, one has a rather good chance to find a valid candidate. However, this can be improved by carefully choosing the bit position on which the initial difference is inserted. Indeed, some conditions can be placed on bit position 31 and will therefore be always verified with probability 1. We give in Table 1 the number of bit conditions of the differential path from Figure 3, according to the bit position of the initial difference inserted.

bit position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nb. conditions	46	46	46	46	46	46	41	46	46	46	46	46	46	38	46	46

bit position	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nb. conditions	46	43	46	46	41	46	46	46	32	46	46	46	46	46	46	35

Table 1. Number of conditions for the differential path from Figure 3, according to the bit position of the first difference inserted.

Note also that valid candidate search speed-ups are very likely to exist. Indeed, the complexity cost here only takes in account the probability of the differential path. However, in hash functions cryptanalysis, the use of the available

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

degrees of freedom can drastically improve the overall complexity of the attack. For example, it is easy to force some bits of the first inserted message word in order to validate with probability 1 the very first condition of the differential path. Also, one can try to validate some conditions of the first iteration and some conditions of the second iteration independently, by playing with the message word inserted just before the second iteration.

4.2 Collision attack for CubeHash-2/4 and CubeHash-2/3

According to Table 1, using the differential path from Figure 3 on bit position 24 provides a collision attack on CubeHash-2/4 with an overall complexity of 2^{32} operations. An example of such a collision for the 512-bit output version of the scheme is given here. It required a few minutes of computation on an average PC (Processor Intel Core 2 Duo 2.0 GHz, with 2 GB of RAM). No specific search speed-up were used which leaves room for further improvements.

This colliding pair is composed of five message inputs. The two first message blocks are used without any difference in order to randomize the values of the internal state just after the initialisation. Then, the third message pair inserts the initial one-bit difference at position 24, and the fourth message pair permits to prepare the internal state for the second part of the differential path. Finally, the fifth and last message pair will erase the remaining difference in the first word of the internal state, and thus leads to an internal collision.

The values of the message words to insert and the final hash value are given in Table 5 in the Appendix B.

In the implementation of CubeHash, one can check that if 3 bytes are inserted at each iteration, they will be xored to the 3 first least significant bytes of X_0 (the first byte of a word is considered to be the least significant one). Thus, using the same differential path but by inserting the initial perturbation at bit position 0 gives us a collision attack against CubeHash-2/3 with only 2^{46} operations. Indeed, if one inserts the perturbation at bit position 0 (in the first byte of X_0), the first message correction after one iteration will occur on bit positions 4, 14 and 22 (all located in the three first bytes of X_0). Finally, after another iteration, the attacker will have to erase the difference located at bit position 4 (in the first byte of X_0) in order to get an internal collision. One can see that all the control needed by the attacker is located in the three first bytes of X_0. Thus, a collision for CubeHash-2/3 can be found with only 2^{46} operations.

4.3 Collision attack for CubeHash-4/4 and CubeHash-4/3

One can use the same linearization technique to cryptanalyze CubeHash-4/4 and CubeHash-4/3. The differential path will be a little bit more complicated than the one from Figure 3 and its probability of success will also be much lower. We give in Figure 4 the new differential path for CubeHash-4/4. The display has been simplified because of the big number of conditions. However, since we use

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

it.	round	step	active bits	nb. cond.
1	1	M	M_1^0	
1	1		x_0^0	
1	1	1-10	$x_0^0,\ x_2^0,\ x_4^{11},\ x_8^7,\ x_{12}^{18},\ x_{17}^0,\ x_{19}^0,\ x_{25}^7$	4
1	2	1-10	$x_0^4,\ x_0^{14},\ x_0^{22},\ x_5^{11},\ x_7^{11},\ x_9^7,\ x_{11}^7,\ x_{13}^{18},\ x_{17}^{14},\ x_{21}^{11},\ x_{21}^{25},\ x_{23}^{11},\ x_{24}^7,\ x_{26}^7,\ x_{31}^{18}$	20
1	3	1-10	$x_0^4,\ x_0^{14},\ x_0^{22},\ x_2^4,\ x_2^{14},\ x_2^{22},\ x_4^1,\ x_4^{15},\ x_4^{25},\ x_8^{11}$ $x_8^{21},\ x_8^{29},\ x_{12}^0,\ x_{12}^8,\ x_{12}^{18},\ x_{12}^{22},\ x_{14}^{18},\ x_{17}^4,\ x_{17}^{14},\ x_{17}^{22}$ $x_{19}^4,\ x_{19}^{14},\ x_{19}^{22},\ x_{22}^{25},\ x_{25}^{11},\ x_{25}^{21},\ x_{25}^{29},\ x_{28}^{18},\ x_{30}^{18}$	30
1	4	1-10	$x_0^4,\ x_0^8,\ x_0^{12},\ x_0^{28},\ x_5^1,\ x_5^{15},\ x_5^{25},\ x_7^1,\ x_7^{15},\ x_7^{25},\ x_9^{11},\ x_9^{21},\ x_9^{29}$ $x_{11}^{11},\ x_{11}^{21},\ x_{11}^{29},\ x_{13}^0,\ x_{13}^8,\ x_{13}^{22},\ x_{17}^4,\ x_{17}^{18},\ x_{17}^{28},\ x_{21}^1,\ x_{21}^7,\ x_{21}^{25},\ x_{21}^{29}$ $x_{23}^1,\ x_{23}^{15},\ x_{23}^{25},\ x_{24}^{11},\ x_{24}^{21},\ x_{24}^{29},\ x_{26}^{11},\ x_{26}^{21},\ x_{26}^{29},\ x_{31}^0,\ x_{31}^8,\ x_{31}^{22}$	60
2	1	M	$M_2^8,\ M_2^{12},\ M_2^{28}$	
2	1		$x_0^4,\ x_5^1,\ x_5^{15},\ x_5^{25},\ x_7^1,\ x_7^{15},\ x_7^{25},\ x_9^{11},\ x_9^{21},\ x_9^{29},\ x_{11}^{11},\ x_{11}^{21}$ $x_{11}^{29},\ x_{13}^0,\ x_{13}^8,\ x_{13}^{22},\ x_{17}^4,\ x_{17}^{18},\ x_{17}^{28},\ x_{21}^1,\ x_{21}^7,\ x_{21}^{25},\ x_{21}^{29},\ x_{23}^1$ $x_{23}^{15},\ x_{23}^{25},\ x_{24}^{11},\ x_{24}^{21},\ x_{24}^{29},\ x_{26}^{11},\ x_{26}^{21},\ x_{26}^{29},\ x_{31}^0,\ x_{31}^8,\ x_{31}^{22}$	
2	1	1-10	$x_0^4,\ x_2^4,\ x_4^{15},\ x_8^{11},\ x_{12}^0,\ x_{12}^8,\ x_{14}^0,\ x_{14}^8,\ x_{14}^{22},\ x_{17}^4,\ x_{19}^4$ $x_{22}^7,\ x_{22}^{15},\ x_{22}^{29},\ x_{25}^{11},\ x_{28}^0,\ x_{28}^8,\ x_{28}^{22},\ x_{30}^0,\ x_{30}^8,\ x_{30}^{22}$	56
2	2	1-10	$x_5^{15},\ x_7^{15},\ x_9^{11},\ x_{11}^{11},\ x_{13}^{22},\ x_{17}^{18},\ x_{21}^{15},\ x_{21}^{29},\ x_{23}^{15},\ x_{24}^{11},\ x_{26}^{11},\ x_{31}^{22}$	29
2	3	1-10	$x_{12}^{22},\ x_{14}^{22},\ x_{22}^{29},\ x_{28}^{22},\ x_{30}^{22}$	18
2	4	1-10	x_0^8	4
3	1	M	M_3^8	

Fig. 4. Linear differential path for `CubeHash`-4/4 and `CubeHash`-4/3. The three first columns give in order the iteration number, the round number and the step number in the internal permutation. A step denoted M represents the active bits of the message block inserted. The fourth column provides the active bits, where X_i^j denotes the j-th bit of the internal word X_i. Finally, the number of conditions is given in the last column. The display of this differential path has been simplified because of the big number of active bits.

the linear model, the entire set of sufficient conditions can be easily retrieved from the input differences.

As for the previous differential path, the number of conditions depends on the bit position of the first perturbation inserted (because the conditions on bit position 31 are automatically verified). We give in Table 2 the number of bit conditions of the differential path from Figure 4, according to the bit position of the initial difference inserted.

One can directly conclude that a collision attack requiring 2^{189} operations can be mounted on `CubeHash`-4/4 by selecting the bit position 20 for the first perturbation inserted. Concerning `CubeHash`-4/3, only the bit positions 4 to 11 can be selected to insert the first perturbation, so that the control required by the attacker only involves the three least significant bytes of the internal word X_0. Thus, by choosing the bit position 9, one gets a collision attack against `CubeHash`-4/3 with only 2^{195} operations.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

bit position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nb. conditions	221	221	198	218	221	221	212	221	221	195	207	221	221	207	221	221

bit position	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nb. conditions	210	207	221	221	189	221	221	213	202	221	221	197	221	221	216	202

Table 2. Number of conditions for the differential path from Figure 4, according to the bit position of the first difference inserted.

4.4 Others versions of `CubeHash`

We can apply similar techniques to cryptanalyze other versions of `CubeHash`. We give in Table 3 the best possible linear differential path probabilities, according to the parameters of the hash function considered and the maximum number of iterations. One can check that for some slower versions of `CubeHash` the probabilities of success of the linear differential paths are too low. However, it may be possible to use more iterations and aim to build a more complex attack composed of the concatenation of several individual linear differential paths. Also, depending on the amount of degrees of freedom available, one may be able to improve the overall complexity of the attack.

r	b	max nb. it.	probability
1	64	3	2^{3}
	32	5	2^{32}
	16	5	
	8	5	
	4	5	
	2	7	2^{221}
	1	15	2^{1225}
2	64	3	2^{32}
	32	3	
	16	3	
	8	3	
	4	3	
	2	4	2^{221}
	1	8	2^{1225}
4	64	3	2^{189}
	32	3	
	16	3	
	8	3	
	4	3	
	2	4	2^{964}
	1	9	2^{2614}
8	64	3	2^{650}
	32	3	2^{830}
	16	3	
	8	3	2^{1009}
	4	3	
	2	5	2^{2614}
	1	5	

Table 3. Best found probability of success for linear differential paths, according to the parameters of the hash function (r and b) and the maximum number of iterations.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

5 Conclusion

In this paper, we provided two different cryptanalysis approaches that led to the computation of a real collision for `CubeHash`-1/36 and `CubeHash`-2/4. The linear differential paths also give a theoretical collision attack against `CubeHash`-2/3 in 2^{46} operations, against `CubeHash`-4/4 in 2^{189} operations and against `CubeHash`-4/3 in 2^{195} operations. Those complexities have to be compared to 2^{128} and 2^{256} for an ideal hash function of 256 and 512-bit output respectively.

6 Acknowledgments

The authors would like to thank Jean-Philippe Aumasson, Shahram Khazaei, Willi Meier and María Naya-Plasencia for the very interesting discussions on `CubeHash`.

References

1. Jean-Philippe Aumasson. Collision for CubeHash2/120-512. NIST mailing list (local link), 2008.
2. Jean-Philippe Aumasson, Willi Meier, María Naya-Plasencia, and Thomas Peyrin. Inside the hypercube. Cryptology ePrint Archive, Report 2008/486, 2008.
3. Daniel J. Bernstein. CubeHash specification (2.b.1). Submission to NIST, 2008.
4. Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.
5. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
6. Wei Dai. Collisions for CubeHash1/45 and CubeHash2/89. Available online, 2008.
7. Wei Dai. Collision for CubeHash2/12. Available online, 2009.
8. Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.
9. National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition.
10. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002.
11. Thomas Peyrin. Cryptanalysis of Grindahl. In *ASIACRYPT*, pages 551–567, 2007.
12. Ronald L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992.
13. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
14. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In Cramer [5], pages 1–18.
15. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Shoup [13], pages 17–36.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

16. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Cramer [5], pages 19–35.
17. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In Shoup [13], pages 1–16.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

Appendix A: collision for CubeHash-1/36-512

Both message instances given in Table 4 lead to the following hash output, in hexadecimal display:

```
fb5e5578 c020296a 9d66df51 c49031ba
12c1eb92 eb404187 0b02e344 d2c9b335
5b1d6afa 8ac26a39 2fa35d96 c684bc3d
d6ecbd6e f71339e3 ba35bd72 841af694
```

Iteration 1

word i	M_i	M'_i	$M_i \oplus M'_i$
0	9d45c73a	9d45c73a	00000000
1	f6106042	f6106042	00000000
2	7f3be941	7f3be941	00000000
3	2b58d9ed	2b58d9ed	00000000
4	b7923ded	b7923ded	00000000
5	fb187e3f	fb187e3f	00000000
6	fd5d2414	fd5d2414	00000000
7	e66ffb2c	e66ffb2c	00000000
8	367126de	367126de	00000000

Iteration 2

word i	M_i	M'_i	$M_i \oplus M'_i$
0	43f69ab4	bc09654b	ffffffff
1	00000000	00000000	00000000
2	e3c37da8	1c6319ad	ffa06405
3	00000000	00000000	00000000
4	58fdd79b	58fdd79b	00000000
5	b08456a3	b08456a3	00000000
6	00000000	00000000	00000000
7	765e25fb	765e25fb	00000000
8	edaea852	bddcae41	50720613

Iteration 3

word i	M_i	M'_i	$M_i \oplus M'_i$
0	00000000	00000000	00000000
1	00000000	00000000	00000000
2	00000000	00000000	00000000
3	00000000	00000000	00000000
4	1be9de4a	1fa91472	0440ca38
5	00000000	00000000	00000000
6	8912b045	6b0ea65f	e21c161a
7	00000000	00000000	00000000
8	00000000	00000000	00000000

Iteration 4

word i	M_i	M'_i	$M_i \oplus M'_i$
0	00000000	7d23e83d	7d23e83d
1	00000000	00000000	00000000
2	00000000	07e4687b	07e4687b
3	00000000	00000000	00000000
4	00000000	00000000	00000000
5	00000000	00000000	00000000
6	00000000	00000000	00000000
7	00000000	00000000	00000000
8	00000000	8e6c1d83	8e6c1d83

Table 4. Message words to insert during four iterations in order to get a collision for the 512-bit output version of CubeHash-1/36. The values are given in hexadecimal notation.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.

Appendix B: collision for CubeHash-2/4-512

Both message instances given in Table 5 lead to the following hash output, in hexadecimal display:

```
220f6a8a 640870f4 2757873d 8f16bc80
0f5595fa a519aa37 2091d3f0 c1e86527
fe9fa656 de7d1cb7 b9c367b2 a06d6616
27aa321d d3fd2ec6 378d61d1 9a270371
```

	M	M'	$M \oplus M'$
Iteration 1	72d9dcf5	72d9dcf5	00000000
Iteration 2	b835e32f	b835e32f	00000000
Iteration 3	05a4593f	04a4593f	01000000
Iteration 4	b897ebd7	a897ab97	10004040
Iteration 5	00000000	10000000	10000000

Table 5. Message words to insert during fives iterations in order to get a collision for the 512-bit output version of CubeHash-2/4. The values are given in hexadecimal notation.

Appeared in D. Pointcheval and M. Abdalla (Eds.): ACNS 2009, LNCS 5536, pp. 354–368.