Preimage Attacks on CellHash and SubHash

Donghoon Chang

Center for Information Security Technologies(CIST),
Korea University, Korea
dhchang@cist.korea.ac.kr

Abstract. CellHash [3] and SubHash [4] were suggested by J. Daemen, R. Govaerts and J. Vandewalle in 1991 and 1992. SubHash is an improved version from CellHash. They have 257-bit internal state and 256-bit hash output. In this paper, we show a preimage attack on CellHash (SubHash) with the complexity 2^{129+t} and the memory $513 * 2^{128-t}$ bits for any t (with the complexity about 2^{241} and the memory $513 * 2^{17}$ bits). Even though we modify them in a famous way, we show that we can find a preimage on the modified CellHash (the modified SubHash) with the complexity 2^{194} and the memory $513 * 2^{64}$ bits (with the complexity about 2^{241} and the memory size $513 * 2^{17}$ bits). So we recommend that a simple invertible structure-repeated hash functions such as CellHash and SubHash have the size of internal state two times longer at least than the output size of hash function. For example, Parallel FFT-Hashing [6] and RadioGatún [1] are such a case.

Keywords : Hash Function, Preimage Attack.

1 Introduction.

Since MD4-style hash functions were broken [7–12], nowadays the research on new structures different from MD4-style structure is required. CellHash [3] and SubHash [4] are different structures from MD4-style hash functions. Even though they were suggested more than 15 years ago, there is no security evaluation on them. In this paper, we describe preimage attacks on them. Even though we add the feedforward process on them that an input intermediate value XOR with its output intermediate value, we show that we can find their preimages with complexity less than exhaustive search. This means that they have weak structures. So this paper's results provide the design principle of hash function to hash function designers. For example, based on this result, we recommend that a simple invertible structure-repeated hash functions such as CellHash and SubHash have the size of internal state two times longer at least than the output size of hash function. Parallel FFT-Hashing [6] and RadioGatún [1] are such a case.

2 CellHash and SubHash

CellHash and SubHash have 257-bit internal state and 256-bit hash output. They use only bit-wise operations and permutation in order to implement them efficiently in hardware. Fig. 1 and Fig. 2 show CellHash and SubHash algorithms. Message padding methods for them are as follows : In case of CellHash, the message is extended with the minimum number of 0's so that its length in bits is at least 248 and congruent to 24 modulo 32. The number of bits added in represented in a byte that is subsequently appended, most significant bit first [3]. In case of SubHash, the message is extended with a number p of 0-bits so that its length in bits is a multiple of 32 and $0 \leqslant p < 32$. Subsequently the message is extended with 1 32-bit word representing the value $2^{32} - 1 - p$, most significant bit first [4].

$\mathsf{CellHash}(M) = o_0||o_1||\cdots||o_{255}$ (256-bit hash output)
M is the padded message for which $M = m_0||m_1||\cdots||m_{n-1}$ and each m_i is 32-bit.

1. **For** $i = 0, \ldots, 256$ **Do** $h_i := c_i$ ($c_0||\cdots||c_{256}$ is the 257-bit initial value.)
2. **For** $j = 1, \ldots, n$ **Do** (: Round $j-1$)
   - 2.1 $a_0||a_1||\cdots||a_{255} = m_{j-1}||m_{j \bmod n}||\cdots||m_{j+6 \bmod n}$, each a_i is 1-bit.
   - 2.2 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 2.3 $h_0 = \overline{h_0}$
   - 2.4 $h_i = h_{i-3} \oplus h_i \oplus h_{i+3}$, $0 \leqslant i < 257$ (in parallel)
   - 2.5 $h_i = h_i \oplus a_{i-1}$, $1 \leqslant i < 257$ (in parallel)
   - 2.6 $h_i = h_{10*i}$, $0 \leqslant i < 257$ (in parallel)
3. **Output** $o_0||o_1||\cdots||o_{255} = h_t||h_{t+1}||\cdots||h_{t+255}$ where $t = 0$ or $t = 1$.

Fig. 1. CellHash Algorithm.

.

3 Preimage Attacks on CellHash and SubHash

In this section, we describe preimage attacks on CellHash and SubHash. These attacks are based on the concept of Meet-in-the-Middle Attack. Firstly, we introduce the concept of Meet-in-the-Middle Attack for finding a preimage of hash function.

Meet-in-the-Middle Attack

We consider the iterated hash function construction such as Merkle-Damgård construction like Fig. 3. In Fig. 3, given a hash output o, we want to find its preimage. When the output size is 256-bit and it takes the complexity 2^{256} to

$\mathsf{SubHash}(M) = o_0||o_1||\cdots||o_{255}$ (256-bit hash output)
M is the padded message for which $M = m_0||m_1||\cdots||m_{n-1}$ and each m_i is 32-bit.

1. **For** $i = 0, \ldots, 256$ **Do** $h_i := c_i$ ($c_0||\cdots||c_{256}$ is the 257-bit initial value.)
2. **For** $i = 0, \ldots, 255$ **Do** $a_i := 0$
3. **For** $j = 0, \ldots, n-1$ **Do** (: Round j)
   - 3.1 $a_0||a_1||\cdots||a_{31} = m_j$ and $a_{32}||a_{33}||\cdots||a_{255} = a_0||a_1||\cdots||a_{223}$ (in parallel)
   - 3.2 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 3.3 $h_0 = \overline{h_0}$
   - 3.4 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 3.5 $h_i = h_i \oplus a_{i-1}$, $1 \leqslant i < 257$ (in parallel)
   - 3.6 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
4. **For** $j = 0, \ldots, 7$ **Do** (: Round $j + n$)
   - 4.1 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 4.2 $h_0 = \overline{h_0}$
   - 4.3 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 4.4 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
5. **For** $j = 0, \ldots, 15$ **Do** (: Round $j + n + 8$)
   - 5.1 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 5.2 $h_0 = \overline{h_0}$
   - 5.3 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 5.4 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
   - 5.5 $o_j||\cdots||o_{j+15} = h_{11,24,37,48,60,73,84,98,117,130,143,154,168,200,235,249}$
6. **Output** $o_0||o_1||\cdots||o_{255}$.

Fig. 2. SubHash Algorithm.

.

find a preimage of f, it also takes the complexity 2^{256} to find a preimage of the iterated hash function. When it takes the complexity 1 to find a preimage of f, we can find a preimage with using the meet-in-the-middle attack. We assume that x_1 and x_2 are independent from x_3 and x_4. Given an output o, we choose randomly x_3 and x_4 and compute the corresponding value in $*$ in Fig. 3 and store them in table. Like this, we get 2^{128-t} cases (Here, we assume that x_i has 64-bit size at least). Similarly, from x_1 and x_2 we compute the corresponding value s in $*$ in Fig. 3. If s is in the table, we can get a preimage of o. According to the birthday paradox, In order to get one preimage we have to compute s from random x_1 and x_2 2^{129+t} times. Therefore, we can get a preimage with the complexity 2^{129+t} and the momory size 2^{128-t}.

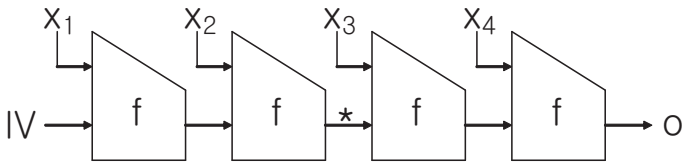

Fig. 3. Merkle-Damgård Construction in case of 4 block-message. IV is the initial value.

When it takes the complexity 2^{129} to find a preimage of f, we can find a preimage with using the meet-in-the-middle attack. We assume that x_1 and x_2 are independent from x_3 and x_4. Given an output o, we choose randomly x_3 and x_4 and compute the corresponding value in $*$ in Fig. 3 and store them in table. Like this, we get 2^{128-t} cases with the complexity 2^{257-t} (Here, we assume that x_i has 64-bit size at least). Similarly, from x_1 and x_2 we compute the corresponding value s in $*$ in Fig. 3. If s is in the table, we can get a preimage of o. According to the birthday paradox, In order to get one preimage we have to compute s from random x_1 and x_2 2^{129+t} times. Therefore, we can get a preimage with the complexity $2^{129+t} + 2^{257-t}$ and the momory size 2^{128-t}. In case of $t = 64$, we can get a preimage with the complexity 2^{194} and the momory size 2^{64}.

Easy to invert Step 2.2 of CellHash in Fig. 1 and Step 3.2, 4.1, 5.1 of SubHash Fig. 2

Except Step 2.2 in CellHash, Step 2.3-2.6 of CellHash are linear parts, which are easy to invert. Step 2.4 is invertible if the size of the intermediate value is no multiple of 9 [3]. We focus on Step 2.2. Step 2.2 is invertible if the size of the intermediate value is odd [3]. We want to find an input 257-bit $h_0||h_1||\cdots||h_{256}$ when we are given an output value of Step 2.2 (We denote the value by $b_0||b_1||\cdots||b_{256}$.).

$$
\begin{aligned}
b_0 &= h_0 \oplus (h_1 \vee \overline{h_2}) \\
b_1 &= h_1 \oplus (h_2 \vee \overline{h_3}) \\
b_2 &= h_2 \oplus (h_3 \vee \overline{h_4}) \\
&\cdot \\
&\cdot \\
b_{254} &= h_{254} \oplus (h_{255} \vee \overline{h_{256}}) \\
b_{255} &= h_{255} \oplus (h_{256} \vee \overline{h_0}) \\
b_{256} &= h_{256} \oplus (h_0 \vee \overline{h_1})
\end{aligned}
$$

From last equation, we try to compute. We guess h_0 and h_1, then h_{256} is determined. Then h_{255} is determined. Similarly, all other values are also determined. Only we check if guessed h_0 and h_1 satisfy first and second equations. Since h_0 and h_1 can be one among (0,0), (0,1), (1,0) and (1,1), it is enough to check 4 times at most. Step 3.2, 4.1, 5.1 of SubHash Fig. 2 is also computed in the same way. Therefore, we can say that it is possible to invert Step 2.2 of CellHash in Fig. 1 and Step 3.2, 4.1, 5.1 of SubHash Fig. 2 with complexity 1.

Preimage Attack on CellHash

Each 32-bit message word is applied eight times. We denote Step 2.1-2.6 by f. Then we can describe CellHash like Fig. 4. We denote j-th round 256-bit input message by $X_j = m_{j \bmod n} || m_{j+1 \bmod n} || \cdots || m_{j+7 \bmod n}$.

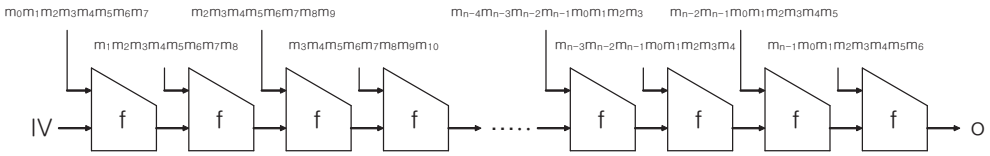

Fig. 4. Message Input Method of CellHash.

Now we try to find a preimage of any given hash output o. Our target padded message is $m_0||m_1||\cdots||m_{31}$. Among the message (corresponding to $X_0 \sim X_{31}$), we give fixed 256-bit values to X_0 and X_{16}, which means that $m_0||m_1||\cdots||m_7$ and $m_{16}||m_{17}||\cdots||m_{23}$ are fixed. Then, we can know that when we give random values to X_8 and X_{24}, the values of $X_0 \sim X_{16}$ (which depend only on X_8) are independent from the values of $X_{17} \sim X_{31}$ (which depend only on X_{16}). Therefore, we can apply the meet-in-the-middle attack in output position of Round 16. As described above, Therefore, we can get a preimage with the complexity 2^{129+t} and the momory size 2^{128-t}.

Preimage Attack on SubHash

Each 32-bit message word is applied eight times. We denote j-th round 256-bit input message by $X_j = m_j||m_{j-1}||\cdots||m_{j-7}$ where $m_{-1} = m_{-2} = \cdots = m_{-7} = 0^{32}$. Our target padded message is $m_0||m_1||\cdots||m_{23}$ ($n = 24$). We know that a given hash output are computed from last sixteen rounds, Round 32∼47. From a given hash output, We know 16 bits in 257-bit output of each of last sixteen rounds. Here, we focus on unknown 241 bits among 257-bit output of Round 32. We find the 241 bits satisfying the remaining 240 bits among 256-bit hash output through exhaustive searching. So, we can expect 2 candidates with complexity 2^{241}, which means that we can 1 candidate with complexity 2^{240}. Now, we have 257-bit output of Round 32. And wee can get the output of Round 23 by inverting the 257-bit output of Round 32. Therefore, we can use same method as the preimage attack on CellHash. Among the message (corresponding to $X_0 \sim X_{23}$), we give fixed 256-bit values to X_{15}, which means that $m_{15}||m_{14}||\cdots||m_8$ are fixed. Then, we can know that when we give random values to X_7 and X_{23}, the values of $X_0 \sim X_{15}$ (which depend only on X_7) are independent from the values of $X_{16} \sim X_{23}$ (which depend only on X_{23}). Therefore, we can apply the meet-in-the-middle attack in output position of Round 15. As described above, Therefore, we can get a preimage with the complexity 2^{129+t} and the momory size 2^{128-t} satisfying the output of Round 23. So, we can find a preimage of a given hash output with complexity $2^{240} + 2^{129+t}$ and the memory size 2^{128-t}. In case of $t = 111$, we can find a preimage of a given hash output with complexity 2^{241} and the memory size 2^{17}.

4 The Modified Versions of CellHash and SubHash

CellHash and SubHash have inverting-easy round functions. On the other hand, in order to make the inverse difficult, MD4-style hash functions use the feedforward operation that the input intermediate is added to the output intermediate in each compression function. In case of block-cipher based hash functions such as PGV construction [2, 5], XOR operation is used as the feedforward operation. Therefore it is required to check the security of CellHash and SubHash with the feedforward operation. In case of CellHash and SubHash, there is no addition operation because they use only bit-wise operations and the permutation. So, we consider their modified versions, CellHash and SubHash which use the feedforward operation with XOR. Fig. 5 and Fig. 6 show the modified versions of CellHash and SubHash.

5 Preimage Attacks on the Modified Versions of CellHash and SubHash

Inverting Problem of Round Function of CellHash and SubHash

In case of the modified CellHash, Section 2.1-2.8 of Fig. 5 is the round function. Since Step 2.4-2.8 are linear, given an output of the round function, we

$\underline{\mathsf{modCellHash}(M) = o_0||o_1||\cdots||o_{255}}$ (256-bit hash output)
$\underline{M}$ is the padded message for which $M = m_0||m_1||\cdots||m_{n-1}$ and each m_i is 32-bit.

1. **For** $i = 0, \ldots, 256$ **Do** $h_i := c_i$ ($c_0||\cdots||c_{256}$ is the 257-bit initial value.)
2. **For** $j = 1, \ldots, n$ **Do** (: Round $j-1$)
   - $\underline{2.1}$ $z_i = h_i$, $0 \leqslant i < 257$
   - 2.2 $a_0||a_1||\cdots||a_{255} = \underline{m_{j-1}}||m_{j \bmod n}||\cdots||m_{j+6 \bmod n}$, each a_i is 1-bit.
   - 2.3 $h_i = \underline{h_i} \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 2.4 $h_0 = \overline{h_0}$
   - 2.5 $h_i = h_{i-3} \oplus h_i \oplus h_{i+3}$, $0 \leqslant i < 257$ (in parallel)
   - 2.6 $h_i = h_i \oplus a_{i-1}$, $1 \leqslant i < 257$ (in parallel)
   - 2.7 $h_i = h_{10*i}$, $0 \leqslant i < 257$ (in parallel)
   - $\underline{2.8}$ $h_i = h_i \oplus z_i$, $0 \leqslant i < 257$
3. **Output** $o_0||o_1||\cdots||o_{255} = h_t||h_{t+1}||\cdots||h_{t+255}$ where $t = 0$ or $t = 1$.

Fig. 5. The Modified CellHash Algorithm.

.

can get the output of Step 2.3 which is represented by the linear combinations of $h_0 \sim h_{256}$. We want to find an input 257-bit $h_0||h_1||\cdots||h_{256}$ when we are given an output value of Step 2.3 which is represented by the linear combinations of $h_0 \sim h_{256}$. We denote the linear combination of i-th bit position by $L_i(h_0, h_1, \cdots, h_{256})$.

$$
\begin{aligned}
L_0(h_0, h_1, \cdots, h_{256}) &= h_0 \oplus (h_1 \vee \overline{h_2}) \\
L_1(h_0, h_1, \cdots, h_{256}) &= h_1 \oplus (h_2 \vee \overline{h_3}) \\
L_2(h_0, h_1, \cdots, h_{256}) &= h_2 \oplus (h_3 \vee \overline{h_4}) \\
&\vdots \\
L_{254}(h_0, h_1, \cdots, h_{256}) &= h_{254} \oplus (h_{255} \vee \overline{h_{256}}) \\
L_{255}(h_0, h_1, \cdots, h_{256}) &= h_{255} \oplus (h_{256} \vee \overline{h_0}) \\
L_{256}(h_0, h_1, \cdots, h_{256}) &= h_{256} \oplus (h_0 \vee \overline{h_1})
\end{aligned}
$$

The right parts of above equations is nonlinear. We can expect that there is one solution because the number of variables is same as that of equation. But, we can not find the solution directly by Gaussian Elimination. Since each h_{2i} is related to two equations, we guess 129 values of h_{2i} for $0 \leqslant i \leqslant 128$. Then above can be represented by the linear combination, so we can find the solution by Gaussian Elimination. Therefore, it takes the complexity 2^{129} to find the solution.

$\underline{\mathsf{modSubHash}(M) = o_0||o_1||\cdots||o_{255}}$ (256-bit hash output)
M is the padded message for which $M = m_0||m_1||\cdots||m_{n-1}$ and each m_i is 32-bit.

1. **For** $i = 0, \ldots, 256$ **Do** $h_i := c_i$ ($c_0||\cdots||c_{256}$ is the 257-bit initial value.)
2. **For** $i = 0, \ldots, 255$ **Do** $a_i := 0$
3. **For** $j = 0, \ldots, n-1$ **Do** (: Round j)
   - $\underline{3.1}$ $z_i = h_i$, $0 \leqslant i < 257$
   - 3.2 $a_0||a_1||\cdots||a_{31} = m_j$ and $a_{32}||a_{33}||\cdots||a_{255} = a_0||a_1||\cdots||a_{223}$ (in parallel)
   - 3.3 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 3.4 $h_0 = \overline{h_0}$
   - 3.5 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 3.6 $h_i = h_i \oplus a_{i-1}$, $1 \leqslant i < 257$ (in parallel)
   - 3.7 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
   - $\underline{3.8}$ $h_i = h_i \oplus z_i$, $0 \leqslant i < 257$
4. **For** $j = 0, \ldots, 7$ **Do** (: Round $j + n$)
   - $\underline{4.1}$ $z_i = h_i$, $0 \leqslant i < 257$
   - 4.2 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 4.3 $h_0 = \overline{h_0}$
   - 4.4 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 4.5 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
   - $\underline{4.6}$ $h_i = h_i \oplus z_i$, $0 \leqslant i < 257$
5. **For** $j = 0, \ldots, 15$ **Do** (: Round $j + n + 8$)
   - $\underline{5.1}$ $z_i = h_i$, $0 \leqslant i < 257$
   - 5.2 $h_i = h_i \oplus (h_{i+1} \vee \overline{h_{i+2}})$, $0 \leqslant i < 257$ (in parallel)
   - 5.3 $h_0 = \overline{h_0}$
   - 5.4 $h_i = h_i \oplus h_{i+3} \oplus h_{i+8}$, $0 \leqslant i < 257$ (in parallel)
   - 5.5 $h_i = h_{12*i}$, $0 \leqslant i < 257$ (in parallel)
   - $\underline{5.6}$ $h_i = h_i \oplus z_i$, $0 \leqslant i < 257$
   - 5.7 $o_j||\cdots||o_{j+15} = h_{11,24,37,48,60,73,84,98,117,130,143,154,168,200,235,249}$
6. **Output** $o_0||o_1||\cdots||o_{255}$.

Fig. 6. The Modified SubHash Algorithm.

.

Preimage Attack on the Modified CellHash

Each 32-bit message word is applied eight times. We denote j-th round 256-bit input message by $X_j = m_{j \bmod n}||m_{j+1 \bmod n}||\cdots||m_{j+7 \bmod n}$. Now we try to find a preimage of any given hash output o. Our target padded message is $m_0||m_1||\cdots||m_{31}$. Among the message (corresponding to $X_0 \sim X_{31}$), we give fixed 256-bit values to X_0 and X_{16}, which means that $m_0||m_1||\cdots||m_7$ and $m_{16}||m_{17}||\cdots||m_{23}$ are fixed. Then, we can know that when we give random values to X_8 and X_{24}, the values of $X_0 \sim X_{16}$ (which depend only on X_8) are independent from the values of $X_{17} \sim X_{31}$ (which depend only on X_{16}). Therefore, we can apply the meet-in-the-middle attack in output position of Round 16. As described above, Therefore, we can get a preimage with the complexity 2^{194} and the momory size 2^{64}.

Preimage Attack on the Modified SubHash

Each 32-bit message word is applied eight times. We denote j-th round 256-bit input message by $X_j = m_j||m_{j-1}||\cdots||m_{j-7}$ where $m_{-1} = m_{-2} = \cdots = m_{-7} = 0^{32}$. Our target padded message is $m_0||m_1||\cdots||m_{23}$ ($n = 24$). We know that a given hash output are computed from last sixteen rounds, Round 32$\sim$47. From a given hash output, We know 16 bits in 257-bit output of each of last sixteen rounds. Here, we focus on unknown 241 bits among 257-bit output of Round 32. We find the 241 bits satisfying the remaining 240 bits among 256-bit hash output through exhaustive searching. So, we can expect 2 candidates with complexity 2^{241}, which means that we can 1 candidate with complexity 2^{240}. Now, we have 257-bit output of Round 32. And wee can get the output of Round 23 by inverting the 257-bit output of Round 32. Therefore, we can use same method as the preimage attack on CellHash. Among the message (corresponding to $X_0 \sim X_{23}$), we give fixed 256-bit values to X_{15}, which means that $m_{15}||m_{14}||\cdots||m_8$ are fixed. Then, we can know that when we give random values to X_7 and X_{23}, the values of $X_0 \sim X_{15}$ (which depend only on X_7) are independent from the values of $X_{16} \sim X_{23}$ (which depend only on X_{23}). Therefore, we can apply the meet-in-the-middle attack in output position of Round 15. As described above, Therefore, we can get a preimage with the complexity 2^{129+t} and the momory size 2^{128-t} satisfying the output of Round 23. So, we can find a preimage of a given hash output with complexity $2^{240} + 2^{129+t} + 2^{257-t}$ and the memory size 2^{128-t}. In case of $t = 111$, we can find a preimage of a given hash output with complexity 2^{241} and the memory size 2^{17}.

References

1. G. Bertoni, J. Daemen, G. V. Assche and M. Peeters, *RadioGatún, a belt-and-mill hash function*, In Second Hash Workshop of NIST, 2006.
2. J. Black, P. Rogaway and T. Shrimpton, *Black-box analysis of the block-cipher-based hash function constructions from PGV*, Advances in Cryptology - CRYPTO'02, LNCS 2442, Springer-Verlag, pp. 320-335, 2002.

3. J. Daemen, R. Govaerts and J. Vandewalle, *A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton*, Asiacrypt'91, LNCS 739, Springer-Verlag, pp. 82-96, 1993.

4. J. Daemen, R. Govaerts and J. Vandewalle, *A Hardware Design Model for Cryptographic Algorithms*, ESORICS'92, pp. 419-434, 1992.

5. B. Preneel, R. Govaerts and J. Vandewalle, *Hash functions based on block ciphers:A synthetic approach*, Advances in Cryptology - CRYPTO'93, LNCS 773, Springer-Verlag, pp. 368-378, 1994.

6. C.P. Schnorr and S. Vaudenay, *Parallel FFT-Hashing*, FSE'93, LNCS 809, Springer-Verlag, pp. 149-156, 1994.

7. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 1–18, Springer-Verlag, 2005.

8. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 19–35, Springer-Verlag, 2005.

9. X. Wang, H. Yu and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 1–16, Springer-Verlag, 2005.

10. X. Wang, Y. L. Yin and H. Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 17–36, Springer-Verlag, 2005.

11. H. Yu, X. Wang, A. Yun and S. Park. Cryptanalysis of the Full HAVAL with 4 and 5 Passes. To appear in *FSE'2006*, Springer-Verlag, 2006.

12. H. Yu, G. Wang, G. Zhang and X. Wang. The Second-Preimage Attack on MD4. In *CANS'2005*, volume **3810** of *Lecture Notes in Computer Science*, pages 1–12, Springer-Verlag, 2005.