Preimage Attack on the Parallel FFT-Hashing Function*

Donghoon Chang[1], Moti Yung[2], Jaechul Sung[3], Seokhie Hong[1], and Sangjin Lee[1]

[1] Center for Information Security Technologies(CIST), Korea University, Korea
{dhchang,hsh,sangjin}@cist.korea.ac.kr
[2] RSA Laboratories and Department of Computer Science, Columbia University, New York, USA
moti@cs.columbia.edu
[3] Department of Mathematics, University of Seoul, Korea
jcsung@uos.ac.kr

Abstract. The parallel FFT-Hashing function was designed by C. P. Schnorr and S. Vaudenay in 1993. The function is a simple and light weight hash algorithm with 128-bit digest. Its basic component is a multi-permutation which helps in proving its resistance to collision attacks.

In this work we show a preimage attack on the parallel FFT-Hashing function using $2^{t+64} + 2^{128-t}$ time complexity and 2^t memory, which is less than the generic complexity 2^{128}. Specifically, when $t = 32$, we can find a preimage using 2^{97} time and 2^{32} memory. Our method can be described as "disseminative-meet-in-the-middle-attack". we actually use the properties of multi-permutation (helpful against collision attack) to our advantage in the attack. Overall, this type of attack (beating the generic one) demonstrates that the structure of the parallel FFT-Hashing function has some weaknesses when preimage attack is considered (and relevant). To the best of our knowledge, this is the first attack on the parallel FFT-Hashing function.

Keywords: Cryptographic Hash Function, Preimage Attack, the Parallel FFT-Hashing function.

1 Introduction

Nowadays, motivated by the breaking of the MD4-style hash functions family, novel constructions of cryptographic hash functions are required as are better understanding of their design principles.

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement) (IITA-2006-(C1090-0603-0025)). Part of this work was done while the first author visited Columbia University.

J. Pieprzyk, H. Ghodosi, and E. Dawson (Eds.): ACISP 2007, LNCS 4586, pp. 59–67, 2007.

The parallel FFT-Hashing function is an example of a potential novel construction. This paper investigates this function, suggested by Schnorr and Vaudenay in 1993 [4] (improving and correcting previously broken designs [2,3,1,6,5]). The parallel FFT-Hashing function uses a simple component called 'multi-permutation' repeatedly. The designers proved that the parallel FFT-Hashing function is collision resistant when the black box multi-permutations are given by the oracle. On the other hand, the designers did not say anything about other security notions such as preimage resistance and second preimage resistance. Unlike MD4-style hash function whose compression function is not invertible, the parallel FFT-Hashing function has a step function which is invertible. For the parallel FFT-Hashing function (the MD4-style hash function), each message string is applied only to one step function (one compression function). Also the internal size of the parallel FFT-Hashing function is twice the output size. Thus, one may think that the parallel FFT-Hashing function can be secure against preimage attacks. Further, the FFT-Hashing function seems to be even secure against time-memory trade-off attacks.

In this paper, however, we give an attack that finds a preimage with complexity $2^{t+64} + 2^{128-t}$ and memory 2^t, which is less than the cost of its (generic) exhaustive search complexity (2^{128}). This attack, therefore, demonstrates some weaknesses in the structure of the design, at least when considered in settings where protection against preimage finding is crucial. We note that our attack exploits the properties of the multi-permutation components, i.e., we capitalize on exactly the property that helps preventing collision attacks in finding the preimage.

General Meet-in-the-Middle Attack. Our attack method is different from the general meet-in-the-middle attack. To show this, we explain the general meet-in-the-middle attack on the parallel FFT-Hashing function. Given a hash output o, we want to find its preimage. The parallel FFT-Hashing function can be described as in Fig. 1. The size of the internal state is 256 bits and the output size is 128 bits. f (corresponding to a step function of the parallel FFT-Hashing function) and g (corresponding to the last s steps which is the constant related to the collision resistance property) can be inverted with complexity 1.

We choose randomly $x_{i+1} \sim x_l$ and compute the corresponding value r in Fig. 1 and store them in table. This is repeated to get 2^t cases. Similarly, from $x_1 \sim x_i$ we compute the corresponding value s in Fig. 1. If s is stored in the table (i.e., we meet in the middle), we get a preimage of o. According to the

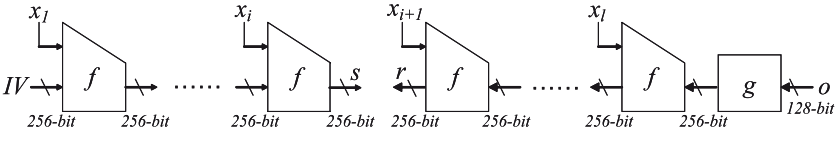

Fig. 1. The Structure of the parallel FFT-Hashing function. f and g are invertible.

birthday paradox, in order to get one preimage we have to compute s from random $x_1 \sim x_i$ 2^{256-t} times. Therefore, we can get a preimage with complexity $2^t + 2^{256-t}$ and memory size 2^t. On the other hand, this paper's attack shows that we can find a preimage with complexity $2^{t+64} + 2^{128-t}$ and memory 2^t.

2 The Parallel FFT-Hashing Function

In this section, we describe the parallel FFT-Hashing function [4]. The size of each word is 16 bits. Here + is the addition modulo 2^{16} and $a \odot b = (a'b' \ mod \ 2^{16} + 1) \ mod \ 2^{16}$ where for $a \neq 0$ and $b \neq 0$ $a' = a$ and $b' = b$ and for a=0 and b=0 $a' = 2^{16}$ and $b' = 2^{16}$. L means the one-bit circular left shift on 4-bit strings and R^j is the j-bit circular right shift on 16-bit strings. Further, $c = 0000000011111111$ is a 16-bit constant and $s = 5$ is the constant related to the number of steps in Fig. 2, which guarantees the collision resistance. In our attack, we can find a preimage for any s (even for big s). The initial value is $(c_0, c_1, \cdots, c_{15})$ which is 16 words. (c_0, c_1, c_2, c_3):=(0xef01, 0x2345, 0x6789, 0xabcd), (c_4, c_5, c_6, c_7):=(0xdcba, 0x9876, 0x5432, 0x10fe), c_{8+i}:=$\overline{c_i}$ for i=0,...,7 where $\overline{c_i}$ is the bitwise logical negation of c_i. Each step of the parallel FFT-Hashing is depicted in Fig. 3.

PaFFTHashing$(M) = o_0||o_1||\cdots||o_7$
M is the padded message for which $M = m_0||m_1||\cdots||m_{n-1} \in E^n$
1. **For** $i = 0, \ldots, 15$ **Do** $e_i := c_i$ ($c_0||\cdots||c_{15}$ is the initial value.)
2. **For** $j = 0, \ldots, \lceil n/3 \rceil$+s-2 **Do** (: Step j)
   2.1 **For** $i = 0, \ldots, 11$ **Do**
   If $m_{3j+(i \bmod 3)}$ is defined,
   $e_{L(i)} := e_{L(i)} + m_{3j+(i \bmod 3)}$ for even i.
   $e_{L(i)} := e_{L(i)} \odot m_{3j+(i \bmod 3)}$ for odd i.
   2.2 **For** $i = 0, \ldots, 7$ **Do** in parallel
   $e_{2i} := e_{L(2i)} \oplus e_{L(2i+1)}$, $e_{2i+1} := e_{L(2i)} \oplus (e_{L(2i+1)} \wedge c) \oplus R^{2i+1}(e_{L(2i+1)})$
   2.3 **For** $i = 0, \ldots, 15$ **Do** $e_i := e_i \odot c_i$
3. **Output** $h_4(M) := o_0||o_1||\cdots||o_7$ for which $o_i = e_{L(2i)} \odot e_{L(2i+1)}$.

Fig. 2. The parallel FFT-hashing function

3 Attack Strategy and Several Properties

In this section, we describe the strategy of our preimage attack on the parallel FFT-Hashing function. See Fig. 4. Our target is to find a padded preimage $m_0||m_1||\cdots||m_{47}$ when a hash output $o_0||o_1||\cdots||o_7$ is given. This strategy consists of 4 phases.

In the first phase, we choose a constant $w_0||w_1||\cdots||w_6||w_7$. In the second phase, we show how to find a message $m_0||m_1||\cdots||m_{23}$ such that the last 4 words of output of step 7.5 are $w_4||w_5||w_6||w_7$ with complexity 1 (time complexity 1 means the time required to simulate 7.5 steps in this case, and the time of computing the entire function once, in general).

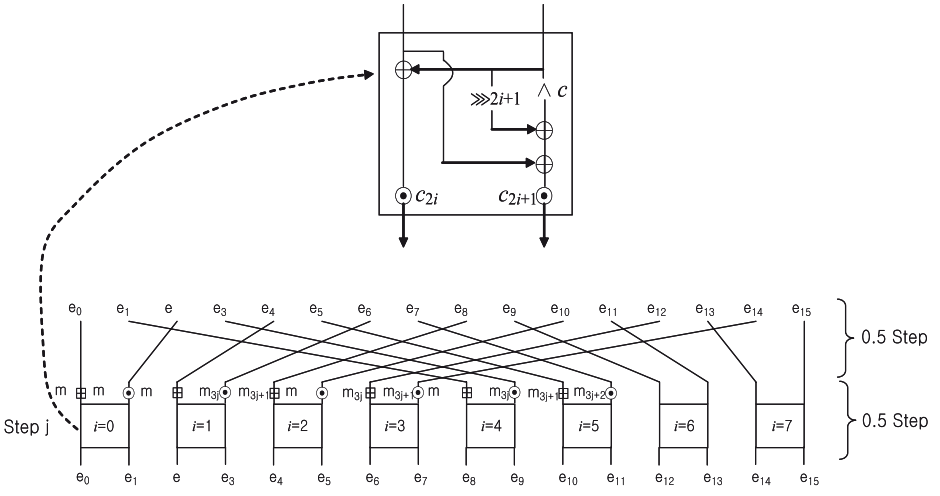

Fig. 3. Step j of the parallel FFT-Hashing function. Each box indicates the invertible multi-permutation which is explained in property 2 in Section 3.

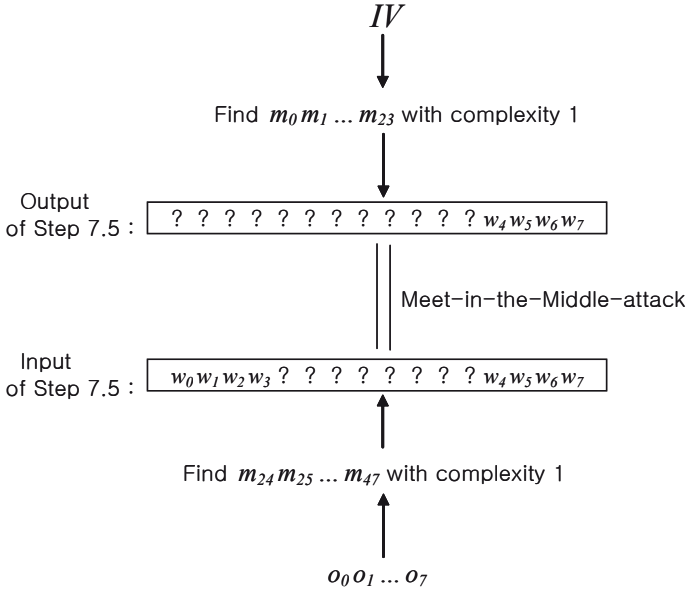

Fig. 4. Preimage Attack Strategy

In the third phase, given hash output $o_0||o_1||\cdots||o_7$, we show how to find a message $m_{24}||m_{25}||\cdots||m_{47}$ such that the first 4 words of the input of step 7.5 and the last 4 words of the input of step 7.5 are $w_0||w_1||w_2||w_3$ and $w_4||w_5||w_6||w_7$ with complexity 1, respectively (time complexity 1 means more precisely here the

time required to simulate 7.5+s steps). In the fourth phase, we find a preimage with the meet-in-the-middle-attack method on the results of phases 2 and 3. We can call this type of meet-in-the-middle-attack "disseminative-meet-in-the-middle-attack" (i.e., partial values are first disseminated through the function structure and the rest is completed employing man-in-the-middle).

We want to describe useful three properties which help us to find a preimage of the parallel FFT-Hashing function.

Property 1: In each step, the last two words of the output e_{13}, e_{15} depend only on the input words $e_9, e_{11}, e_{13}, e_{15}$. See Fig. 3.

A permutation $B : E^2 \rightarrow E^2$, $B(a,b) = (B_1(a,b), B_2(a,b))$, is a *multi-permutation* if for every $a, b \in E$ the mappings $B_i(a,*)$, $B_i(*,b)$ for $i = 1, 2$ are permutations on E.

Property 2: Each box of Fig. 3 is an invertible multi-permutation [5] ($E = \{0,1\}^{16}$). For example, for any b and i, if $B_i(*,b)$ is fixed, then $*$ and $B_{i+1 \ mod \ 2}(*,b)$ are determined automatically. And for any a, $a \odot *$ and $* \odot a$ are invertible permutations on $\{0,1\}^{16}$.

Property 3: For any $a, b, c, d, a', b', c', d', t$ and all cases of Fig. 5, if we choose the value of m, m' and m^* are determined automatically by property 2 and then the undefined values are also determined.

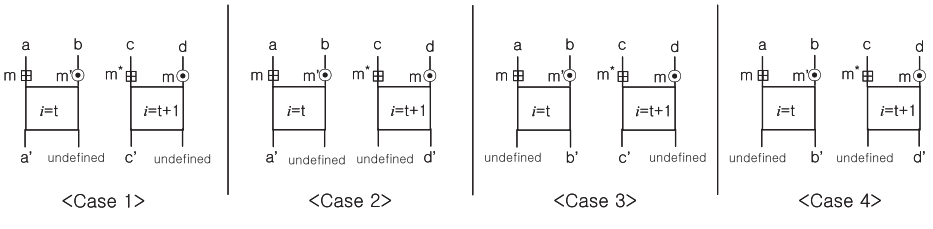

Fig. 5. Four Cases of Property 3

4 Preimage Attack on the Parallel FFT-Hashing Function

In this section, we show how to get a preimage for a given hash output $o_0||o_1||\cdots||o_7$. The original preimage is $m_0||m_1||\cdots||m_{42}$. After the preimage is padded, the padded preimage is $m_0||m_1||\cdots||m_{47}$ where the last four words $w_{44}||w_{45}||w_{46}||w_{47}$ indicate the message length and m_{43} is '1000000000000000'. Our attack idea is a disseminative-meet-in-the-middle attack in the location of output of Step 7.5.

First Phase (Choice of a constant $w_0||w_1||\cdots||w_6||w_7$) See Fig. 6. We can describe the relations among (0)$\sim$(35) like table 1. In table 1, $a \rightarrow b$ means that the value of b is determined by the value of a. (0) $\sim$ (3) [the last 4 entries into step 0 layer in Fig. 6] are already fixed values because they are initial values. So,

Table 1. Relations among (0)∼(35) in Fig 6

(0),(1),(2),(3)	→	(20),(21)	(20),(21) → (23)
(4),(5)	→	(22)	(22),(23) → (25)
(6),(7)	→	(24)	(24),(25) → (27)
(8),(9)	→	(26)	(26),(27) → (29)
(10),(11)	→	(28)	(28),(29) → (31)
(12),(13)	→	(30)	(30),(31) → (33)
(14),(15)	→	(32)	(32),(33) → (35)
(16),(17)	→	(34)	

if we choose values of (4) ∼ (17), then the values of (20) ∼ (35) are determined (via computation) by property 1 as we describe in table 1. And we choose the values of (18) and (19). Finally, we let $w_4||w_5||w_6||w_7$ be (18)||(19)||(34)||(35) and let $w_0||w_1||w_2||w_3$ be any fixed value.

Second Phase (find a message $m_0||m_1||\cdots||m_{23}$ which keeps the last 4 words of output of step 7.5 as a 4-word constant $w_4||w_5||w_6||w_7$ with complexity 1) See Fig. 6. We can describe the relations among $m_0 \sim m_{23}$ as the following table 2 : Once m_2 is fixed, m_0 and m_1 are determined by property 3 because (4) and (5) are already fixed. Likewise, once m_5 is fixed, m_3 and m_4 are also determined by property 3 because (6) and (7) are already fixed. Similarly, we can find $m_0 \sim m_{23}$ satisfying the values of (4) ∼ (19). Since we can assign m_{3i+2} random values for $0 \leqslant i \leqslant 7$, we know that there are 2^{128} $m_0 \sim m_{23}$ satisfying the values of (4) ∼ (19).

Table 2. Relations among $m_0 \sim m_{23}$ in Fig 6

m_2	→	m_0,m_1
m_5	→	m_3,m_4
m_8	→	m_6,m_7
m_{11}	→	m_9,m_{10}
m_{14}	→	m_{12},m_{13}
m_{17}	→	m_{15},m_{16}
m_{20}	→	m_{18},m_{19}
m_{23}	→	m_{21},m_{22}

Third Phase (given the hash output $o_0||o_1||\cdots||o_7$, we show how to find a message $m_{24}||m_{25}||\cdots||m_{47}$ which makes the first 4 words of the input of step 7.5 and the last 4 words of the input of step 7.5 '$w_0||w_1||w_2||w_3$' and '$w_4||w_5||w_6||w_7$' with complexity 1.) See Fig. 7. Given a hash output $o_0||o_1||\cdots||o_7$, by property 2, we can invert $o_0||o_1||\cdots||o_7$ up-to the output of step 11 by giving arbitrary random value to $m_{36} \sim m_{42}$. As described in the first paragraph of Section 4, $m_{43} \sim m_{47}$ are already fixed. And $w_0 \sim w_7$ are already fixed in the first phase, so (40)∼(45) are determined as well. Further, since we know the output of step

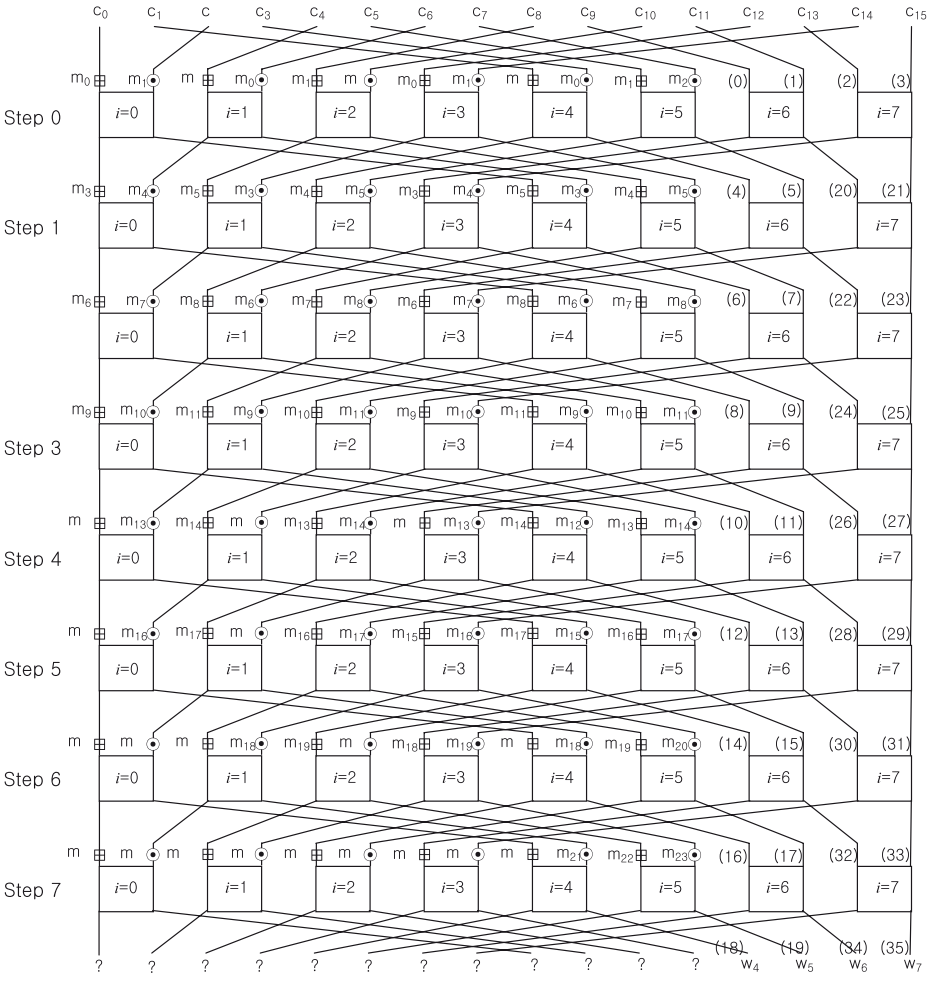

Fig. 6. The First and Second Phases

11, (46) is also fixed through the inverting process. Then m_{34} is determined by property 2 because (45) and (46) are already fixed. At this point we give arbitrary random values to m_{33} and m_{35}. Now we have the output of Step 10. Then m_{31} is determined by (44), at which point we give arbitrary random values to m_{30} and m_{32}. Then m_{27} and m_{28} are determined by (40) and (42). Then (36), (38) and (39) are also determined, while m_{26} and m_{24} are also determined by (38) and (39). Then, employing the property of multi-permutation, m_{25} is determined by (36). Then (37) is automatically determined, so m_{29} is also determined by (37). Therefore, we can get $m_{24} \sim m_{47}$ satisfying $w_0 \sim w_7$ with complexity 1. Since we can assign m_{30}, m_{32}, m_{33} and $m_{35} \sim m_{42}$ random values, we know that there are 2^{176} $m_{24} \sim m_{47}$ cases.

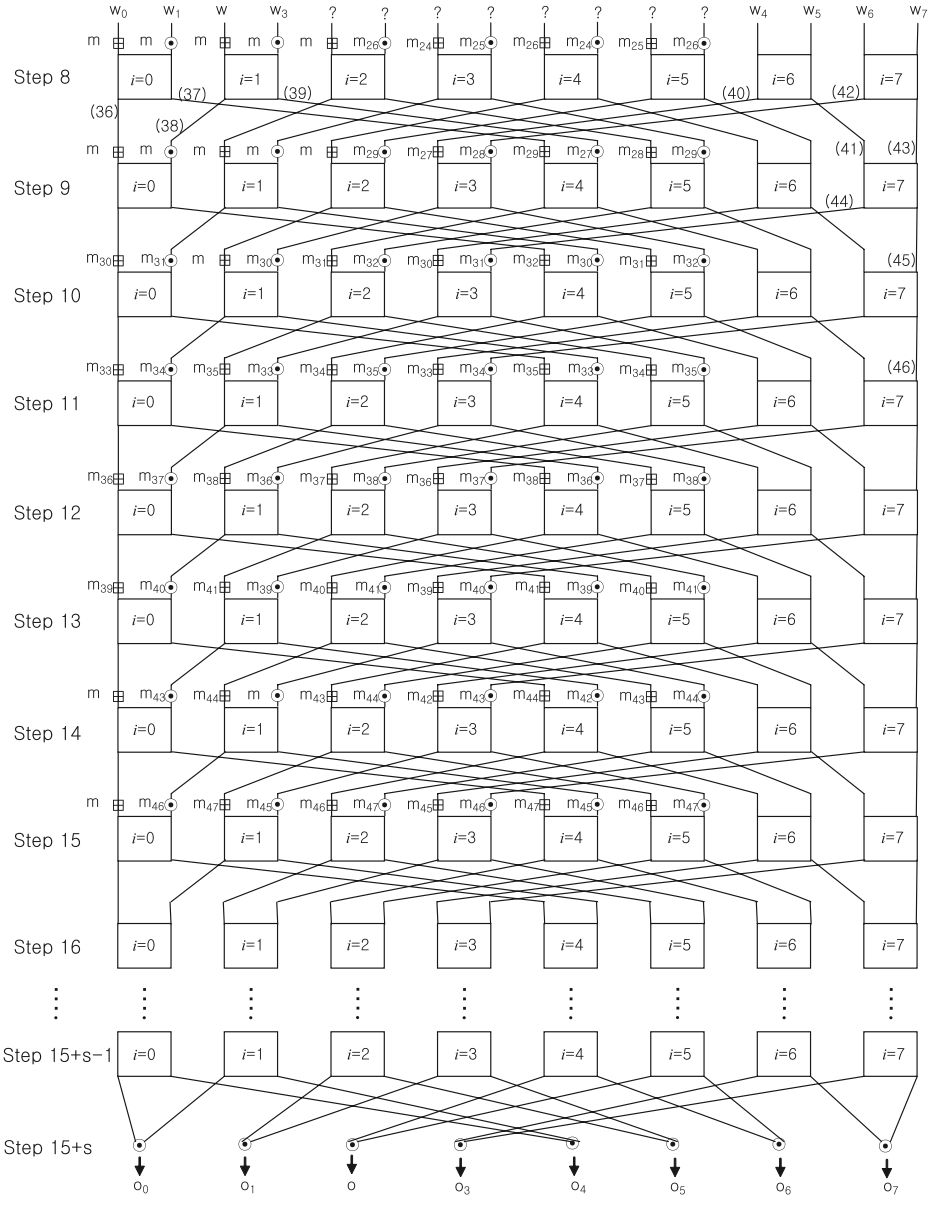

Fig. 7. The Third Phase

Fourth Phase (Meet-in-the-Middle-attack) We repeat the second phase 2^{t+64} times. Then we can get 2^t $m_0 \sim m_{23}$ which make the first 4-word of the output of step 7.5 $w_0||w_1||w_2||w_3$. We store these 2^t $m_0 \sim m_{23}$ and the output of step 7.5 for each $m_0 \sim m_{23}$. We repeat the third phase 2^{128-t} times. According to the birthday attack complexity, given a hash output $o_0||o_1||\cdots||o_7$, we can find a

padded preimage $m_0 \sim m_{47}$ with $2^{t+64} + 2^{128-t}$ time complexity and 2^t memory. This concludes our attack.

Note that our attack does not depend on the value of s which is the constant related to the number of steps guaranteeing the collision resistance property. Also our attack can be used in any word size case (in this paper, we only consider 16-bit word size).

5 Conclusion

In this paper, we described a preimage attack on the parallel FFT-Hashing function which is the first attack on this design. For example we can find a preimage with 2^{97} time complexity and 2^{32} memory which is less than the generic preimage attack complexity of 2^{128}.

References

1. Baritaud, T., Gilbert, H., Girault, M.: FFT Hashing is not Collision-free. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 35–44. Springer, Heidelberg (1993)
2. Schnorr, C.P.: FFT-Hashing: An Efficient Cryptographic Hash Function. In: Presented at the rump session of the Crypto'91
3. Schnorr, C.P.: FFT-Hash II, efficient hashing. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 45–54. Springer, Heidelberg (1993)
4. Schnorr, C.P., Vaudenay, S.: Parallel FFT-Hashing. In: Anderson, R. (ed.) Fast Software Encryption. LNCS, vol. 809, pp. 149–156. Springer, Heidelberg (1994)
5. Schnorr, C.P., Vaudenay, S.: Black Box Cryptanalysis of Hash Networks based on Multipermutations. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 47–57. Springer, Heidelberg (1995)
6. Vaudenay, S.: FFT-Hash II is not yet Collision-free. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 587–593. Springer, Heidelberg (1993)