Preimage Attack on Parallel FFT-Hashing

Donghoon Chang

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

Abstract. Parallel FFT-Hashing 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 Parallel FFT-Hashing with complexity $2^{t+64} + 2^{128-t}$ and memory 2^t which is less than the generic complexity 2^{128}. When $t = 32$, we can find a preimage with complexity 2^{97} and memory 2^{32}. 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 Parallel FFT-Hashing has some weaknesses when preimage attack is considered. To the best of our knowledge, this is the first attack on Parallel FFT-Hashing.

Keywords : Cryptographic Hash Function, Preimage Attack, parallel FFT-Hashing.

1 Introduction.

Nowadays, novel constructions of cryptographic hash functions are required as well as better understanding of their design principles. This is so, since the MD4-style hash functions family was broken. This paper investigates the parallel FFT-Hashing function, suggested by Schnorr and Vaudenay in 1993 [7] (improving and correcting previously broken designs [5, 6, 2, 9, 8]). The parallel FFT-Hashing function uses a simple component 'multi-permutation' repeatedly. Therefore, its algorithm can be implemented in low power memory device environment (typical modern devices such as RFID and sensors are memory limited). Thus, it becomes an attractive alternative, since so far it had no known weaknesses. So we want to know whether the structure of the parallel FFT-Hashing function can be a candidate to be used for designing a new hash function because the structure of parallel FFT-Hashing function is different from MD4-style hash functions. Especially we want to know its security in the point of view of the preimage resistance. Unlike MD4-style hash function, The parallel FFT-Hashing function has a round function which is invertible. But, unlike MD4-style hash function, the internal size of the parallel FFT-Hashing function is twice of the output size.

So, it is easy to think that the parallel FFT-Hashing function may be secure against the preimage attack. The hash function seems to be even secure against time-memory trade-off attack.

However, this paper shows that we can find a preimage with complexity $2^{t+64} + 2^{128-t}$ and memory 2^t which is less than the cost of its exhaustive search complexity (2^{128}). This type of attack demonstrates some weaknesses in the structure of the design. We note that we exploit the properties of the multi-permutation components in our attack, i.e., we capitalize on exactly the property that helps preventing collision attacks.

General Meet-in-the-Middle Attack Our attack method is different from the general meet-in-the-middle attack. For this, we explain the general meet-in-the-middle attack on Parallel FFT-Hashing. Given a hash output o, we want to find its preimage. Parallel FFT-Hashing can be described like Fig. 1. The size of the internal state is 256 bits and the output size is 128 bits. f and g can be inverted with the complexity 1. We choose randomly $x_{i+1} \sim x_t$ and compute the corresponding value in $*$ in Fig. 1 and store them in table. Like this, we get 2^t cases. Similarly, from $x_1 \sim x_i$ we compute the corresponding value s in $*$ in Fig. 1. 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 $x_1 \sim x_i$ 2^{256-t} times. Therefore, we can get a preimage with the complexity $2^t + 2^{256-t}$ and the momory size 2^t. On the other hand, this paper shows that we can find its preimage with complexity $2^{t+64} + 2^{128-t}$ and memory 2^t.

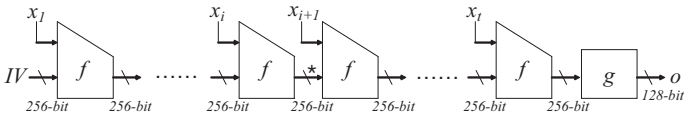

Fig. 1. Parallel FFT Hashing. f and g are invertible.

2 Parallel FFT-Hashing

In this section, we describe Parallel FFT-Hashing [7]. The size of each word is 16 bits. Here $+$ is the addition modulo 2^{16} on $E \cong \{0, 1, \cdots, 2^{16} - 1\}$, $*$ is the multiplication in $E \cong \mathbb{Z}^*_{2^{16}+1}$. L is the one-bit circular left shift on $\{0,1\}^4$ (such that $L(i) = 2i$ for $i \leqslant 7$ and $L(i) = 1 + 2(i - 8)$ for $i > 7$) and R is the one-bit circular right shift on E. Further, $c = 0000000011111111$ and $s = 5$ 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):=(ox ef01, ox2345, ox6789, oxabcd), (c_4, c_5, c_6, c_7):=(oxdcba, ox9876, ox5432, ox10fe), c_{8+i}:=$\overline{c_i}$ for i=0,...,7 where c_i is the bitwise logical negation of $\overline{c_i}$.

$\underline{\mathsf{PaFFTHashing}(M) = o_0||o_1||\cdots||o_7}$
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.
     $e_{L(i)} := e_{L(i)} * m_{3j+(i \bmod 3)}$ 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** $e_i := e_i * c_i$
3. **Output** $h_4(M) := o_0||o_1||\cdots||o_7$ for which $o_i = e_{L(2i)} * e_{L(2i+1)}$.

Fig. 2. Parallel FFT-hashing.

.

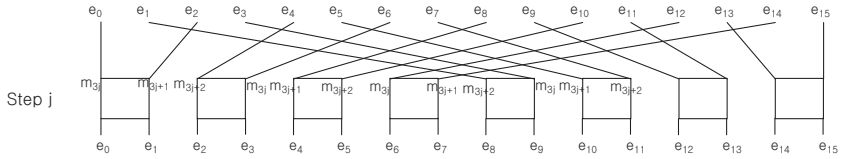

Fig. 3. Step j of Parallel FFT-Hashing.

3 Attack Strategy

In this section, we describe the strategy of our preimage attack on Parallel FFT-Hashing. 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 $w_0||w_1||\cdots||w_6||w_7$ which will be used for the preimage attack on Parallel FFT-Hashing. In second phase, we show how to find a message $m_0||m_1||\cdots||m_{23}$ which keeps the last 4 words of output of step 7 as a 4-word constant $w_4||w_5||w_6||w_7$ with complexity 1. In the third phase, given hash output $o_0||o_1||\cdots||o_7$, we show how to find a message $m_{24}||m_{25}||\cdots||m_{47}$ which makes the first 4 words of the input of step 8 and the last 4 words of input of step 8 '$w_0||w_1||w_2||w_3$' and '$w_4||w_5||w_6||w_7$' with complexity 1. In the forth 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".

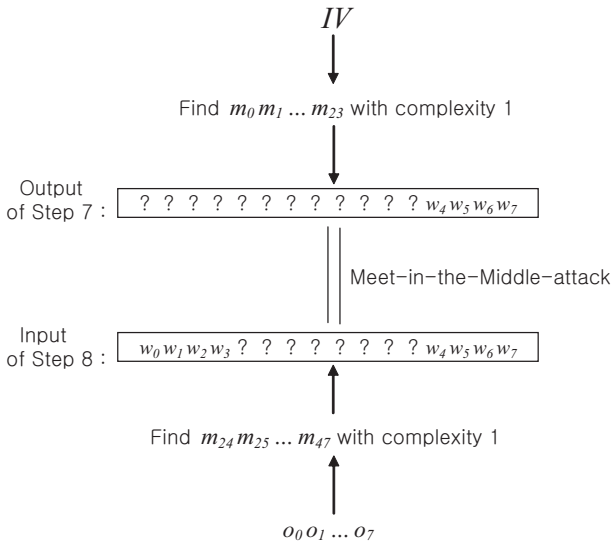

Fig. 4. Preimage Attack Strategy

4 Preimage Attack on Parallel FFT-Hashing

In this section, we show how to get a preimage for a given hash output $o_0||o_1||\cdots||o_7$. The preimage is $m_0||m_1||\cdots||m_{42}$. So when the preimage is padded, the padded preimage is $m_0||m_1||\cdots||m_{47}$ for which each m_i is 16 bit long. The last four words $w_{44}||w_{45}||w_{46}||w_{47}$ indicate the message length. We let m_{43} be '1000000000000000'.

Our attack idea is a disseminative-meet-in-the-middle attack in the location of output of Step 7. see Fig. 5 and Fig. 6 for tags denoting locations in the cipher process that we will use throughout.

First Phase (Choice of a constant $w_0||w_1||\cdots||w_6||w_7$): (0) $\sim$ (3) [the last 4 entries into step 0 layer in the figure below] are already fixed values because they are initial values. We give (4) $\sim$ (19) fixed values. Then the values of (20) $\sim$ (35) are determined (via computation) as follows : (20) is determined by (0) and (1), (21) is determined by (2) and (3), (22) is determined by (4) and (5), (23) is determined by (20) and (21), ..., (35) is determined by (32) and (33). Then 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 as a 4-word constant $w_4||w_5||w_6||w_7$ with complexity 1) : Once m_2 is fixed, m_0 and m_1 are determined are automatically determined by the property of multi-permutation because (4) and (5) are already fixed. A permutation $B : E^2 \to E^2$, $B(a, b) = (B_1(a, b), B_2(a, b))$, is a *multi-permutation* if for every $a, b \in E$ the mappings $B_i(a, *)$, $B_i(*, b)$ for $i = 1, 2$ are permutation on E. Likewise, once m_5 is fixed, m_3 and m_4 are also determined because (6) and (7) are already fixed. Similarly, we can find $m_0 \sim m_{23}$ satisfying the values of (4) $\sim$ (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) $\sim$ (19).

Third Phase (given hash output $o_0||o_1||\cdots||o_7$, we show how to find a message $m_{24}||m_{25}||\cdots||m_{47}$ which makes the first 4 words of the input of step 8 and the last 4 words of input of step 8 '$w_0||w_1||w_2||w_3$' and '$w_4||w_5||w_6||w_7$' with complexity 1.) : Given a hash output $o_0||o_1||\cdots||o_7$, since the multi-permutation is an invertible permutation, we can invert $o_0||o_1||\cdots||o_7$ up-to the output of step 11 by giving arbitrary random value to $m_{36} \sim m_{42}$. Note that $m_{43} \sim m_{47}$ are already fixed. $w_0 \sim w_7$ is already fixed, so (40)$\sim$(45) are determined as well. Further, since we know the output of step 11, (46) is also fixed through the inverting process. m_{34} is determined by (45) and (46). Then we give arbitrary random values to m_{33} and m_{35}. Now we have the output of Step 10. m_{31} is determined by (44). Then we give arbitrary random values to m_{30} and m_{32}. m_{27} and m_{28} are determined by (40) and (42). Then (36), (38) and (39) are also determined. 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.

Forth 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 $w_0||w_1||w_2||w_3$. We store these 2^t $m_0 \sim m_{23}$ and the output of step 7

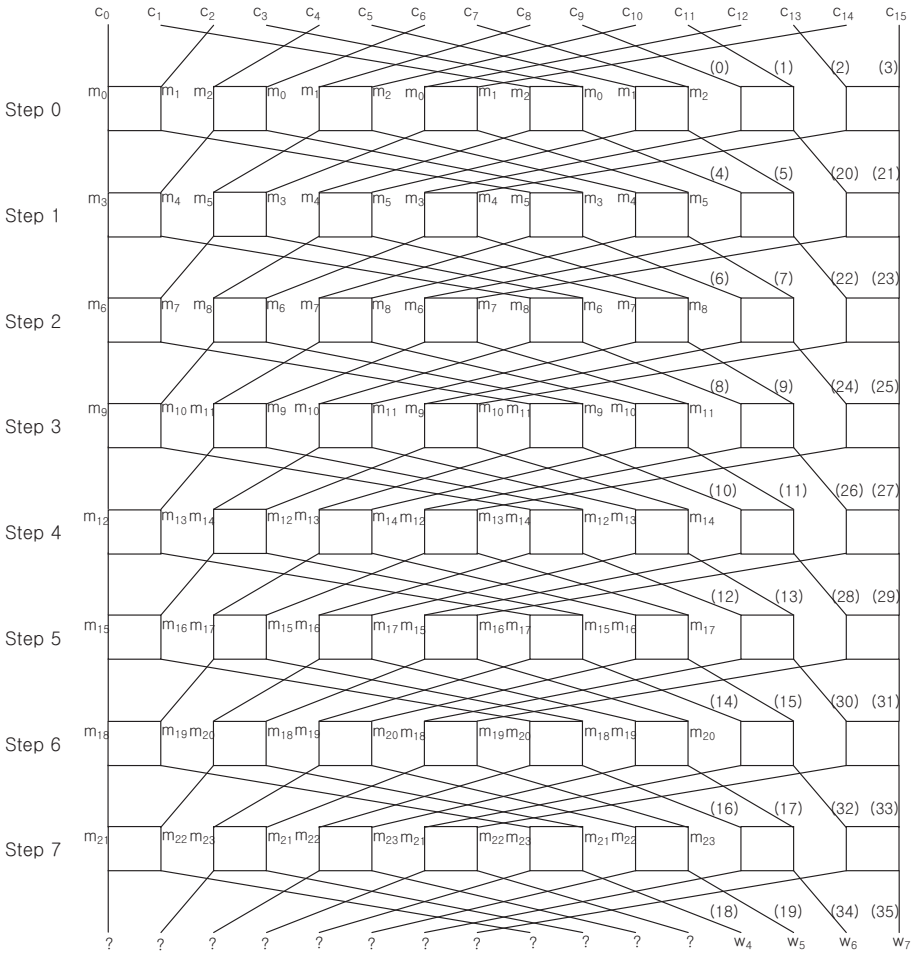

Fig. 5. First Part : Eight Steps.

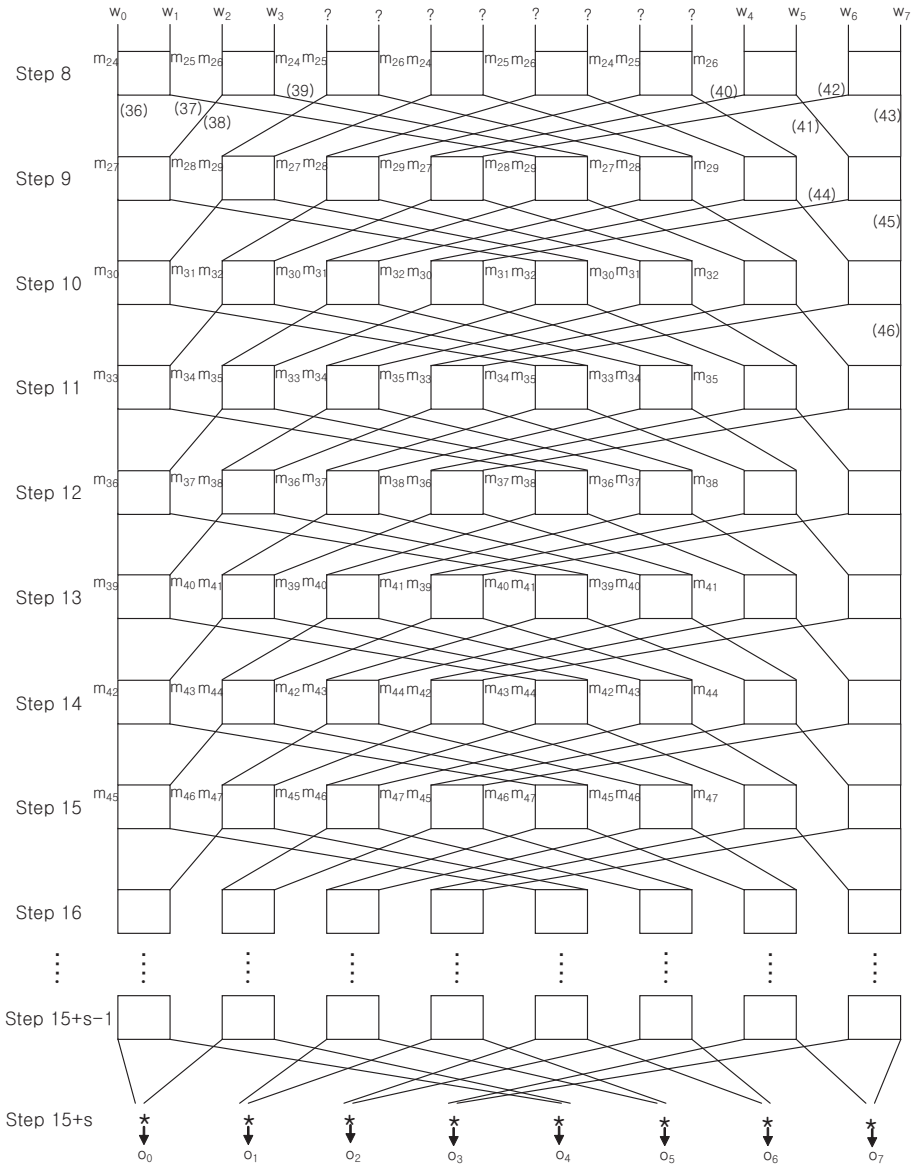

Fig. 6. Second Part.

for each $m_0 \sim m_{23}$. And 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 complexity $2^{t+64} + 2^{128-t}$ and memory 2^t. This concludes our attack.

Note that our attack does not depend on the value of s, which means that the security analysis of collision resistance of Parallel FFT-Hashing [7, 8] can not guarantee the security against the preimage attack. Further, for the same reason our attack can be used in case of any word size (in this paper, we only consider 16-bit word size).

5 Conclusion

In this paper, we described a preimage attack on Parallel FFT-Hashing which is the first attack on this design. For example we can find a preimage with time complexity 2^{97} and memory 2^{32}. This attack demonstrates that components (like multi-permutations) that assure against collision attacks may as well be exploited for preimage attacks.

Acknowledgement

Thank Prof. Moti Yung for encouraging the author to analyze Parallel FFT-Hashing and naming "disseminative-meet-in-the-middle-attack". Thank Prof. Jeachul Sung for pointing out typos in this paper.

References

1. R. Anderson and E. Biham, *Tiger: A Fast New Hash Function*, FSE'96, LNCS 1039, Springer-Verlag, pp. 89-97, 1996.
2. T. Baritaud, H. Gilbert and M. Girault, *FFT Hashing is not Collision-free*, Eurocrypt'92, LNCS 658, Springer-Verlag, pp. 35-44, 1992.
3. P. S. L. M. Barreto and V. Rijmen, *FFT The Whirlpool Hashing Function*, First open NESSIE Workshop, Leuven, Belgium, 13–14 November 2000.
4. Second Hash Workshop held by NIST, Aug. 2006. You can download all papers and presentations from http://www.csrc.nist.gov/pki/HashWorkshop.
5. C.P. Schnorr, *FFT-Hashing : An Efficient Cryptographic Hash Function*, Presented at the rump session of the Crypto'91.
6. C.P. Schnorr, *FFT-Hash II, efficient hashing*, Eurocrypt'92, LNCS 658, Springer-Verlag, pp. 45-54, 1992.
7. C.P. Schnorr and S. Vaudenay, *Parallel FFT-Hashing*, FSE'93, LNCS 809, Springer-Verlag, pp. 149-156, 1994.
8. C.P. Schnorr and S. Vaudenay, *Black Box Cryptanalysis of Hash Networks based on Mulitipermutations*, Eurocrypt'94, LNCS 950, Springer-Verlag, pp. 47-57, 1995.
9. S. Vaudenay, *FFT-Hash II is not yet Collision-free*, Crypto'92, LNCS 740, Springer-Verlag, pp. 587-593, 1993.