Preimage Analysis of the Maelstrom-0 Hash Function

Riham AlTawy and Amr M. Youssef

Concordia Institute for Information Systems Engineering,
Concordia University, Montréal, Québec, Canada

Abstract. Maelstrom-0 is the second member of a family of AES-based hash functions whose designs are pioneered by Paulo Baretto and Vincent Rijmen. According to its designers, the function is designed to be an evolutionary lightweight alternative to the ISO standard Whirlpool. In this paper, we study the preimage resistance of the Maelstrom-0 hash function using its proposed 3CM chaining construction. More precisely, we apply a meet-in-the-middle preimage attack on the compression function and combine it with a guess and determine approach which allows us to obtain a 6-round pseudo preimage for a given compression function output with time complexity of 2^{496} and memory complexity of 2^{112}. Then, we propose a four stage attack in which we adopt another meet-in-the-middle attack and a 2-block multicollision approach to defeat the two additional checksum chains and turn the pseudo preimage attack on the compression function into a preimage attack on the hash function. Using our approach, preimages of the 6-round reduced Maelstrom-0 hash function are generated with time complexity of 2^{505} and memory complexity of 2^{112}.

Keywords: Cryptanalysis, Hash functions, Meet in the middle, Preimage attack, Maelstrom-0, 3CM.

1 Introduction

The attacks of Wang *et al.* [28, 27] which broke a large cluster of widely used hash functions have proven to be most effective against Add-Rotate-Xor (ARX) based hash functions. The success of such attacks on ARX constructions is attributed to the possibility of finding differential trails that propagate for a significant number of rounds with acceptable probabilities. Moreover, considerable improvement in the attack complexity can be achieved using message modification techniques [28] which take advantage of the independence of consecutive message words which may span over a relatively large number of rounds. On the other hand, the Advanced Encryption Standard (AES) wide trail strategy [7] continues to show solid resistance to standard differential attacks. This fact has made AES-based hash functions a favorable direction when considering new designs. Indeed, at the same time when most of the standardized ARX-based hash functions were failing to resist the techniques introduced by Wang *et al.*, the already existing ISO standard Whirlpool [23] was not affected by these attacks. This conceptual shift in hash function designs was clearly evident among the SHA-3 competition proposals [22] (e.g., the SHA-3 finalists Grøstl [12] and JH [29], and LANE [16]). Additionally, Whirlwind [6] and Streebog [20], the new Russian hash standard which is officially known as GOST R 34.11-2012, are also among the recently proposed AES-based hash functions.

Maelstrom-0 is an AES-based hash function that adopts a modified chaining scheme called 3CM [8]. The function is proposed by Filho, Barreto, and Rijmen as an evolutionary lighter alternative to its predecessor Whirlpool. Maelstrom-0 is considered the second member of a family of hash functions which is preceded by Whirlpool and followed by Whirlwind. The design of Maelstrom-0 is heavily inspired by Whirlpool but adopts a simpler key schedule and takes into account the recent development in hash function cryptanalysis. Particularly, the designers consider those attacks where the cryptanalytic techniques which are applicable on the compression function can be easily mapped to the hash function due to the simplicity of the Merkle-Damgård construction used by Whirlpool. In addition to adopting a simpler key schedule which makes Maelstrom-0 more robust and significantly faster than Whirlpool, the designers employ the Davis-Mayer compression mode which is the only mode among the twelve secure constructions that naturally allows the compression function to accept a message block size different from the chaining value size, thus allowing faster hashing rate [8]. Also, all the remaining eleven constructions XOR the message and the chaining value block, thus forcing either truncation or padding to cope with the different sizes, and it is unclear to what extent truncation or padding might adversely affect the security analysis.

The most important feature in the design of Maelstrom-0 is the proposal of a new chaining construction called 3CM which is based on the 3C/3C+ family [13]. This construction computes two checksums from the generated intermediate chaining values, concatenates them, and as a finalization step processes the result as a message block in the last compression function call. This finalization step aims to thwart some generic attacks on the MD construction used in Whirlpool such as long second preimage and herding attacks, and also inhibits length extension attacks. According to the designers of Maelstrom-0, the proposed finalization step mitigates the applicability of extending attacks on the compression function to the hash function. Unfortunately, this is not the case in our attack where we employ a 4-stage approach that uses a modified technique which defeats the 3CM chaining construction [9–11] and combines it with another meet-in-the-middle (MitM) attack to extend a pseudo preimage attack on the compression function to a preimage attack on the hash function.

Literature related to the cryptanalysis of Maelstrom-0 include the analysis of the collision resistance of its compression function by Kölbl and Mendel [18] where the weak properties of the key schedule were used to produce semi free-start collision for the 6 and 7 round reduced compression function and semi free-start near collision for the 8 and 10-rounds compression function. Finally, Mendel *et al.* used the rebound attack to show how employing a message block whose size is double that of the chaining state is used to present a free start collisison on the 8.5 reduced round compression function [21].

In this work, we investigate the security of Maelstrom-0 and its compression function, assessing their resistance to the MitM preimage attacks. Employing the partial matching and initial structure concepts [24], we present a pseudo preimage attack on the 6-round reduced compression function. In the presented attack, we employ a guess and determine approach [26] to guess parts of the state. This approach helps in maintaining partial state knowledge for an extra round when all state knowledge is lost due to the wide trail effect. The proposed 6-round execution separation maximizes the overall probability of the attack by balancing the chosen number of starting values and the guess size. Finally, we propose a four stage approach which combines a 2-block multicollision attack [9, 10] with a second MitM attack to bypass the effect of the 3CM checksum used in the finalization step. Our approach is successfully used to generate preimages of the 6-round reduced Maelstrom-0 hash function using the presented pseudo preimage attack on the last compression function. Up to our knowledge, our analysis is the first to consider the hash function and not only the compression function of Maelstrom-0.

The rest of the paper is organized as follows. In the next section, a brief overview of the related work regarding MitM preimage attacks and the used approaches is provided. The description of the Maelstrom-0 hash function along with the notation used throughout the paper are given in Section 3. Afterwards, in Sections 4, we provide detailed description of the pseudo preimage attack on the compression function. In Section 5, we show how preimages of the hash function are generated using our four stage approach and the attack presented in Section 4. Finally, the paper is concluded in Section 6.

2 Related Work

A pseudo preimage attack on a given a compression function CF that processes a chaining value h and a message block m is defined as follows: Given x, one must find h and m such that $CF(h, m) = x$. The ability to generate a pseudo preimage for the compression function has always been regarded as a certificational weakness as its local effect on the overall hash function is not important . However, as we are going to show in Section 5, when a pseudo preimage attack on the compression function is combined with other attacks, it can be used to build a preimage for the whole hash function.

The MitM preimage attack was first proposed by Aoki and Sasaki [5]. The main concept of the proposed MitM attacks is to separate the attacked rounds at a starting point into two independent executions that proceed in opposite directions (forward and backward chunks). The two executions must remain independent until the point where matching takes place. To maintain the independence constraint, each execution must depend on a different set of inputs, e.g., if only the forward chunk is influenced by a change in a given input, then this input is known as a forward neutral input. Consequently, all of its possible values can be used to produce different outputs of the forward execution at the matching point. Accordingly, all neutral inputs for each execution direction attribute to the number of independent starting values for each execution. Hence, the output of the forward and the backward executions can be independently calculated and stored at the matching point. Similar to all MitM attacks, the matching point is where the outputs of the two separated chunks meet to find a solution, from both the forward and backward directions, that satisfies both executions. While for block ciphers, having a matching point is achieved by employing both the encryption and decryption oracles, for hash function, this is accomplished by adopting the cut and splice technique [5] which utilizes the employed mode of operation. In other words, given the compression function output, this technique chains the input and output states through the feedforward as we can consider the first and last states as consecutive rounds. Subsequently, the overall attacked rounds behave in a cyclic manner and one can find a common matching point between the forward and backward executions and consequently can also select any starting point.

Ever since their inception, significant improvements on MitM preimage attacks have been proposed. Such improvements include the initial structure approach [25, 24] which allows the starting point to span over few successive transformations where bytes in the states are allowed to belong to both the forward and backward chunks. Additionally, the partial matching technique [5] enables only parts of the state to be matched at the matching point which extends the matching point further by not restricting full state knowledge at the matching point. Once a partial match is found, the starting values of both executions are selected and used to evaluate the remaining undetermined parts of the state at the matching point to check for a full state match. Figure 1 illustrates the MitM preimage attack approaches for a compression function operating in the Davis-Mayer mode. The red and blue arrows denote the forward and backward executions on the message state, respectively. S_0 is the first state initialized by h and S_i is the last attacked state.

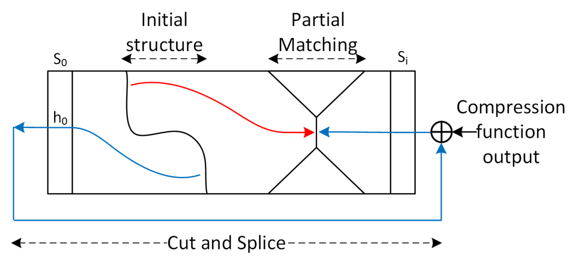

Fig. 1. MitM preimage attack techniques used on a Davis-Mayer compression function.

The MitM preimage attack was applied on MD4 [5, 14], MD5 [5], HAS-160 [15], and all functions of the SHA family [4, 3, 14]. The attack exploits the weak key schedules of ARX-based functions where some of the expanded message blocks are used independently in each round. Thus, one can determine which message blocks affect each execution for the MitM attack. Afterwards, the MitM preimage attack was adapted on the AES block cipher in hashing modes [24]. The attack was then applied to Whirlpool and a 5-round pseudo preimage attack on the compression function was used for a second preimage attack on the whole hash function in the same work. In the sequel, Wu *et al.* [30] improved the time complexity of the 5-round attack on the Whirlpool compression function. Moreover, they applied the MitM pseudo preimage attack on Grøstl and adapted the attack to produce pseudo preimages of the reduced hash function. Afterwards, a pseudo preimage attack on the 6-round Whirlpool compression function and a memoryless preimage attack on the reduced hash function were proposed in [26]. Finally, AlTawy and Youssef employed MitM pseudo preimages of the compression function of Streebog to generate preimages of the reduced hash function [1], the complexity of their attack was later improved in [19]. They also presented a second preimage analysis of Whirlwind [2].

3 Specifications of Maelstrom-0

Maelstrom-0 is an AES-based iterative hash function designed by Filho, Barreto and Rijmen [8]. Its compression function processes 1024-bit message blocks and a 512-bit chaining value. As depicted in Figure 2, the message M is padded by 1 followed by zeros to make the length of the last block 768. Then the remaining 265 bits are used for the binary representation of the message length $|M|$. Hence the padded message has the form $M = m_1||m_2||\cdots||m_k$, where the last 256-bits of m_k denote $|M|$. The compression function is iterated in the 3CM chaining

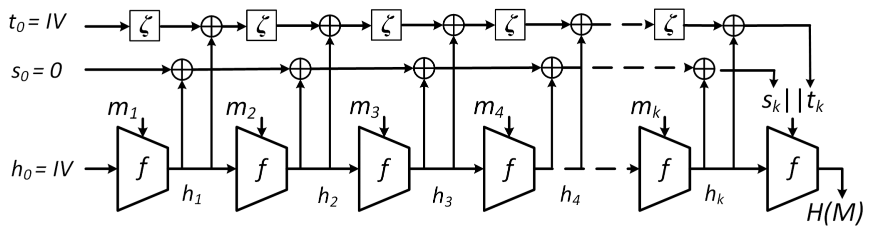

Fig. 2. The Maelstrom-0 hash function.

mode which is based on 3C/3C+ family [13]. Given that h_i denotes the internal state value after processing the message block m_i, i.e., $h_i = f(m_i, h_{i-1})$ with $h_0 = IV$, this chaining mode generalizes the Merkle-Damgård construction by maintaining three chains h_i, s_i, t_i instead of only h_i. The extra two chains are transformed into an additional message block $m_{k+1} = s_k||t_k$. The second chain s_i is a simple XOR accumulation of all intermediate compression function outputs, recursively defined as $s_0 = 0$, $s_i = h_i \oplus s_{i-1}$. The third chain is recursively defined as $t_0 = IV$, $t_i = h_i \oplus \zeta(t_{i-1})$ where an LFSR is employed by ζ to update t_{i-1} by left shifting it by one byte followed by a one byte XOR. More precisely, we compute the hash value h_i in the following way:

$$\begin{aligned} h_0 &= IV, \\ h_i &= f(h_{i-1}, m_i), \text{ for } i = 1, 2, ..., k, \\ H(M) &= f(h_k, s_k||t_k). \end{aligned}$$

The compression function, f, employs a block cipher, E and uses the Davis-Mayer mode of operation. The internal cipher is based on the one used in Whirlpool where it only differs in the key schedule. The round function which operates on 8×8 byte state is initially loaded with the input chaining value. As depicted in Figure 3, the state is updated through 10 rounds and one key addition at the beginning. One round of the state update function consists of the application of the following four transformations:

- The nonlinear layer γ: A transformation that consists of parallel application of a nonlinear Sbox on each byte using an 8-bit Sbox. The used Sbox is the same as the one used in Whirlpool.
- The cyclical permutation π: This layer cyclically shifts each column of its argument independently, so that column j is shifted downwards by j positions, $j = 0, 1, \cdots, 7$.
- The linear diffusion layer θ: A MixRow operation where each row is multiplied by an 8×8 MDS matrix over F_{2^8}. The values of the matrix are chosen such that the branch number of MixRow is 9. Therefore the total number of active bytes at both the input and output is at least 9.
- The key addition σ: A linear transformation where the state is XORed with a round key state.

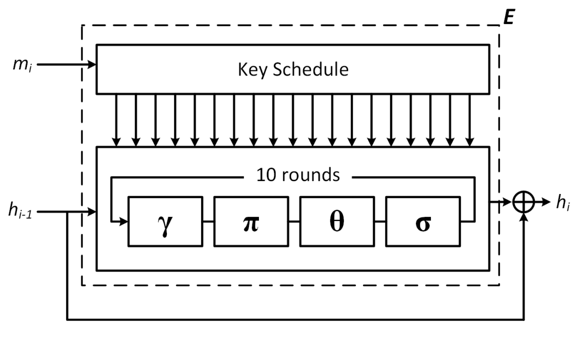

Fig. 3. The Maelstrom-0 compression function.

The key schedule takes as input the 1024-bit message block and generates the 512-bit round keys, $K_0, K_1, \cdots, K_{10}$. Since the key scheduling process is not relevant to our attack, we do not give a detailed description of the round key generation function. For more details on the specification of Maelstrom-0, the reader is referred to [8].

Notation: Let X be (8×8) byte state denoting the internal state of the function. The following notation is used in our attacks:

– X_i: The message state at the beginning of round i.
– X_i^U: The message state after the U transformation at round i, where $U \in \{\gamma, \pi, \theta, \sigma\}$.
– $X_i[r, c]$: A byte at row r and column c of state X_i.
– $X_i[\text{row } r]$: Eight bytes located at row r of state X_i.
– $X_i[\text{col } c]$: Eight bytes located at column c of state X_i.

4 Pseudo Preimage Attack on the 6-Round Reduced Compression Function

In our analysis of the compression function, we are forced to adopt a pseudo preimage attack because the compression function operates in Davis-Mayer mode. Consequently, using the cut and splice technique causes updates in the first state which is initialized by the chaining value. In our attack, we start by dividing the two execution chunks around the initial structure. More precisely, we separate the six attacked rounds into a 3-round forward chunk and a 2-round backward chunk around the starting round represented by the initial structure. The proposed chunk separation is shown in Figure 4. The number of the forward and backward starting values in the initial structure amounts for the complexity of the attack. Accordingly, one must try to balance the number starting values for each chunk and the number of known bytes at the matching point at the end of each chunk. The total number of starting values in both directions should produce candidate pairs at the matching point to satisfy the matching probability.

To better explain the idea, we start by demonstrating how the initial structure is constructed. The main objective of the MitM attack separation is to maximize the number of known bytes at the start of each execution chunk. This can be achieved by selecting several bytes as neutral so that the number of corresponding output bytes of the θ and θ^{-1} transformations at the start of both chunks that are constant or relatively constant is maximized. A relatively constant byte is a byte whose value depends on the value of the neutral bytes in one execution direction but remains constant from the opposite execution perspective. As depicted in Figure 5, we want to have six constants in the lowermost row in state a, then we need to evaluate the possible values of the corresponding red row in state b such that the values of the selected six constants in state a hold. The values of the lowermost red row in state b are the possible forward

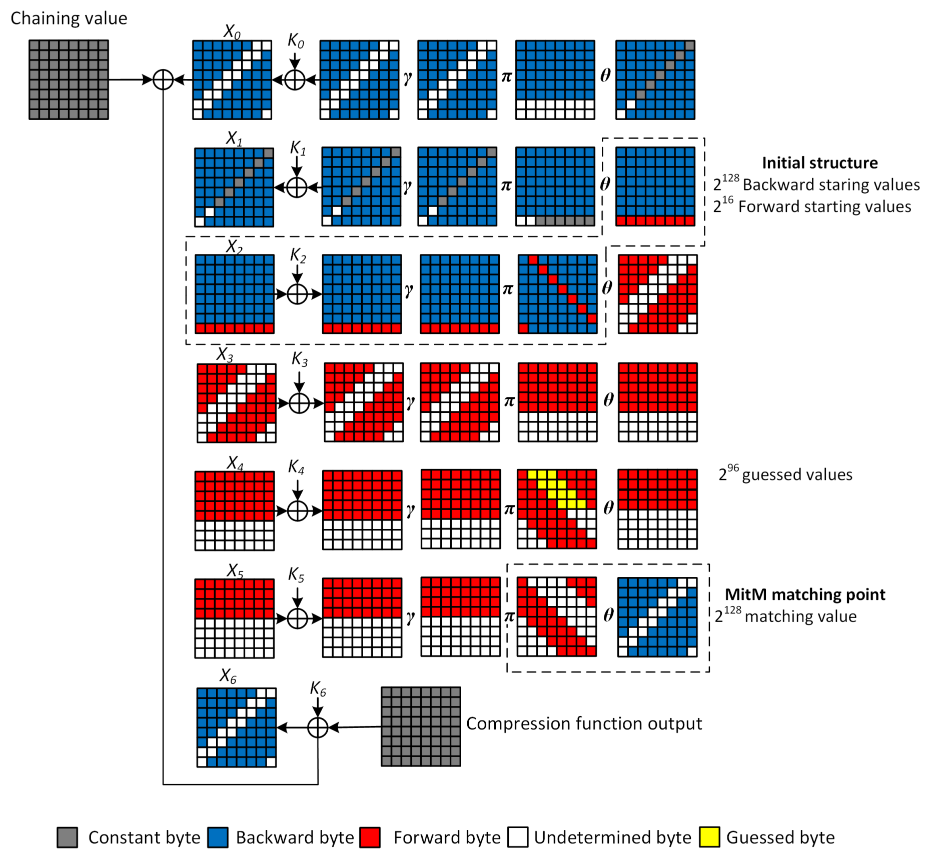

Fig. 4. Chunk separation for a 6-round MitM pseudo preimage attack the compression function.

starting values. For the lowermost row in state b, we randomly choose the six constant bytes in a[row 7] and then evaluate the values of red bytes in b[row 7] so that after applying θ^{-1} on b[row 7], the chosen values of the six constants hold. Since we require six constant bytes in the lowermost row in state a, we need to maintain six variable bytes in b[row 7] in order to solve a system of six equations when the other two bytes are fixed. Accordingly, for the last row in state b, we can randomly choose any two red bytes and compute the remaining six so that the output of θ^{-1} maintains the previously chosen six constant bytes at state a. To this end, the number of forward starting values is 2^{16}. Similarly, we choose 40 constant bytes in state d and for each row in state c we randomly choose two blue bytes and compute the other five such that after the θ transformation we get the predetermined five constants at each row in d. However, the value of the five shaded red bytes in each row of state d depends also on the one red byte in the rows of state c. We call these bytes relative constants because their final values cannot be determined until the forward execution starts and these values are different for each forward execution iteration. Specifically, their final values are the predetermined constants acting as offsets which are XORed with the corresponding red bytes multiplied by the MDS matrix coefficients. In the sequel, we have two free bytes for each row in c which means 2^{128} backward starting values.

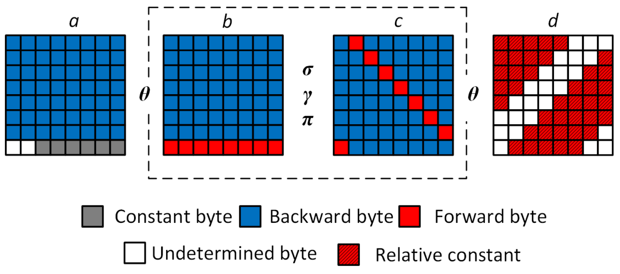

Fig. 5. Initial structure used in the attack on the 6-round compression function.

Following Figure 4, due to the wide trail strategy where one unknown byte results in a full unknown state after two rounds, we lose all state knowledge after applying θ on X_4^π. To maintain partial state knowledge in the forward direction and reach the matching point at X_5^π, we adopt a guess and determine approach [26], by which, we can probabilistically guess the undetermined bytes in some rows of the state at round 4 before the linear transformation. Thus, we maintain knowledge of some state rows after the linear transformation θ which are used for matching. One have to carefully choose the number of guessed bytes and both starting values in the initial structure to result in an acceptable number of correctly guessed matching pairs. Accordingly, we guess the twelve unknown yellow bytes in state X_4^π. As a result, we can reach state X_5^π with four determined bytes in each row where matching takes place.

As depicted in Figure 4, the forward chunk begins at X_2^θ and ends at X_5^π which is the input state to the matching point. The backward chunk starts at X_1^π and ends after the feedforward at X_5^θ which is the output state of the matching point. The red bytes denote the bytes which are affected by the forward execution only and thus can be independently calculated without the knowledge of the blue bytes. White words in the forward chunk are the ones whose values depend on the blue bytes of the backward chunk. Accordingly, their values are undetermined. Same rationale applies to the blue bytes of backward execution. Grey bytes are constants which can be either the compression function output or the chosen constants in the initial structure.

At the matching point, we partially match the available row bytes from the forward execution at X_5^π with the corresponding row bytes from the backward execution at X_5^θ through the linear θ transformation. In each row, we have four and six bytes from the forward and backward executions, respectively. Since the linear mapping is performed on bytes, we compose four byte linear equations in two unknown bytes. Then we evaluate the values of the two unknown bytes from two out of the four equations and substitute their values in the remaining two equations. With probability 2^{-16} the two remaining byte equations are satisfied. Hence, the matching probability for one state row is 2^{-16}. Thus, the partial matching probability for the whole state is $2^{8\times -16=-128}$.

For our attack, the chosen number for the forward and backward starting values, and the guessed values are 2^{16}, 2^{128}, and 2^{96}, respectively. Setting these parameters fixes the number of matching values to 2^{128}. The chosen parameters maximize the attack probability as we aim to increase the number of starting forward values and keep the number of backward and matching values as close as possible and larger than the number of guessed values. In what follows, we give a description of the attack procedure and complexity based on the above chosen parameters:

1. Randomly choose the constants in X_1^{π} and X_2^{θ} and the input message block value.
2. For each forward starting value fw_i and guessed value g_i in the 2^{16} forward starting values and the 2^{96} guessed values, compute the forward matching value fm_i at X_5^{π} and store (fw_i, g_i, fm_i) in a lookup table T.
3. For each backward starting value bw_j in the 2^{128} backward starting values, we compute the backward matching value bm_j at X_5^{θ} and check if there exists an $fm_i = bm_j$ in T. If found, then a partial match exists and the full match should be checked. If a full match exists, then we output the chaining value h_{i-1} and the message m_i, else go to step 1.

The complexity of the attack is evaluated as follows: after step 2, we have $2^{16+96} = 2^{112}$ forward matching values which need 2^{112} memory for the look up table. At the end of step 3, we have 2^{128} backward matching values. Accordingly, we get $2^{112+128} = 2^{240}$ partial matching candidate pairs. Since the probability of a partial match is 2^{-128} and the probability of a correct guess is 2^{-96}, we expect $2^{240-128-96} = 2^{16}$ correctly guessed partially matching pairs. To check for a full match, we want the partially matching starting values to result in the correct values for the 48 unknown bytes in both X_5^{π} and X_5^{θ} that make the blue and red words hold. The probability that the latter condition is satisfied is $2^{48\times -8} = 2^{-384}$. Consequently, the expected number of fully matching pairs is 2^{-368} and hence we need to repeat the attack 2^{368} times to get a full match. The time complexity for one repetition is 2^{112} for the forward computation, 2^{128} for the backward computation, and 2^{16} to check that partially matching pairs fully match. The overall time complexity of the attack is $2^{368}(2^{112}+2^{128}+2^{16}) \approx 2^{496}$ and the memory complexity is 2^{112}.

5 Preimage of the Maelstrom-0 hash function

In this section, we propose a 4-stage approach by which we utilize the previously presented pseudo preimage attack on the Maelstrom compression function to produce a preimage for the whole hash function. The designers of Maelstrom-0 proposed the 3CM chaining scheme that computes two additional checksum chains specifically to inhibit the ability of extending attacks on the compression function to the hash function. The two additional checksums are computed from a combination of the XOR of the intermediate chaining values, then the two results are concatenated and processed as the input message block of the last compression function call in the hash function. At first instance, this construction seems to limit the scope of our attack to the compression function. Nevertheless, employing the 4-stage approach, a preimage of the hash function can be found when we consider a large set of messages that produce different combinations of intermediate chaining values and thus different checksums and combine it with a set of pseudo preimage attacks on the last compression function call. Hence, another MitM attack can be performed on both sets to find a message that correspond to the retrieved checksums. As depicted in Figure 6, the attack is divided into four stages:

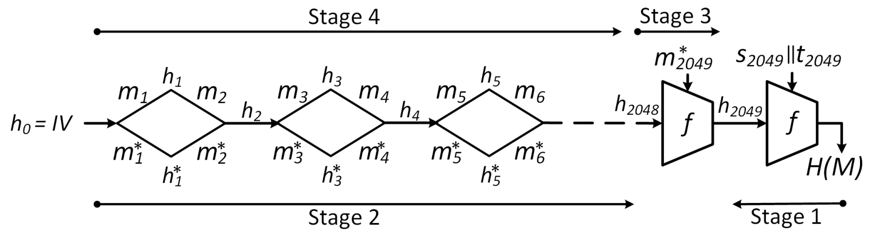

Fig. 6. A 4-stage preimage attack on the Maelstrom-0 hash function.

1. Given the hash function output $H(M)$, we produce 2^p pseudo preimages for the last compression function call. The output of this step is 2^p pairs of the last chaining value and the two checksums $(h_{2049}, s_{2049}, t_{2049})$. We store these results in a table T.
2. In this stage, we construct a set of 2^{1024} of 2-block messages such that all of them collide at h_{2048}. This structure is called a 2-block multicollision of length 1024 [10, 17]. More precisely, an n-block multicollisison of length t is a set of 2^t messages where each message consists of exactly $n \times t$ blocks and every consecutive n application of the compression function results in the same chaining value. Consequently, we have 2^t different possibilities for the intermediate chaining values and all the 2^t n-block messages lead to the same $h_{n\times t}$ value. Constructing a 2^t n-block mulitcollision using exhaustive collision search requires a time complexity of $t(2(n-1)+2^{b/2})$, where b is the chaining state size, and a memory complexity of $t(2 \cdot n)$ message to store t two messages of n-block each. In our case, we build 2^{1024} 2-block multicollision where each 2-block collision gives us two choices for the checksum of two consecutive chaining values. In other words, in the first 2-block collision,

we either choose (h_1, h_2) or (h_1^*, h_2) and thus two choices for the checksum chains. To this end, we have 2^{1024} different 2-block massages stored in $1024 \cdot 2 \cdot 2 = 2^{12}$ memory and hence 2^{1024} candidate chaining checksums.

3. At this stage, we try to connect the resulting chaining value, h_{2048}, from stage 2 to one of 2^p chaining values, h_{2049}, stored in T which was created in stage 1, using the freedom of choosing m_{2049}. Specifically, we randomly choose 512 bit of m^*_{2049}, then properly pad it and append the message length, and using h_{2048} generated by the multicollison, we compute h^*_{2049} and check if it exists in T. As T contains 2^p entries, it is expected to find a match after 2^{512-p} evaluations of the following compression function call:

$$h^*_{2049} = f(h_{2048}, m^*_{2049}).$$

   Once a matching h^*_{2049} value is found in T, the corresponding checksums s^*_{2049}, t^*_{2049} are retrieved. Hence the desired checksums at the output of the multicollision, s_{2048} and t_{2048} are equal to $s^*_{2049} \oplus h^*_{2049}$ and $\zeta^{-1}(t^*_{2049} \oplus h^*_{2049})$, respectively.
4. At the last stage of the attack, we try to find a message M out of the 2^{1024} 2-block messages generated in stage 2 that results in checksums equal to the ones retrieved in stage 3. For this, we form a system of 1024 equations in 1024 unknowns to select one combination from the 2^{1024} different combinations of possible chaining checksums which make the retrieved two checksums hold. Note that, the algorithm proposed in [9] which employs 2^{512} 2-block multicollision and treats the two checksums independently by solving two independent systems of 512 equations cannot work on 3CM, as the two checksums are dependent on each other. This algorithm only works on the 3C chaining construction [10, 11] because it utilizes only one checksum. Accordingly, in our solution, we adopt 1024 2-block messages to find a common solution for the two checksums simultaneously, hence, having the required freedom to satisfy two bit constraints for each bit position in the two checksums. The time complexity of this stage is about $1024^3 = 2^{30}$.

The time complexity of the attack is evaluated as follows: we need $2^p\times$ (complexity of pseudo preimage attack) in stage 1, $1024 \times 2^{256} + 2048 \approx 2^{266}$ to build the 2-block multicollision at stage 2, 2^{512-p} evaluations of one compression function call at stage 3, and finally 2^{30} for stage 4. The memory complexity for the four stages is as follows: 2^p 3-states to store the pseudo preimages in stage 1 and 2^{112} for the pseudo preimage attack, and 2^{12} for the multicollision in stage 2. Since the time complexity is highly influenced by p, so we have chosen $p = 8$ to maximize the attack probability. Accordingly, preimages for the 6-round Maelstrom-0 hash function can be produced with a time complexity of $2^{8+496} + 2^{266} + 2^{512-8} + 2^{30} \approx 2^{505}$. The memory complexity of attack is dominated by the memory requirements of the pseudo preimage attack on the compression function which is given by 2^{112}.

6 Conclusion

In this paper, we have investigated Maelstrom-0 and its compression function with respect to MitM preimage attacks. We have shown that with a carefully balanced chunk separation and the use of a guess and determine approach, pseudo preimages for the 6-round reduced compression function are generated with time complexity of 2^{496} and memory complexity of 2^{112}. Moreover, we have analyzed the employed 3CM chaining scheme which is designed specifically to inhibit the ability of extending attacks on the compression function to the hash function, and proposed a 4-stage approach to bypass its effect and turn the pseudo preimage attack on the compression function to a preimage attack on the hash function. Accordingly, 6-round hash function preimages are generated with time complexity of 2^{505} and a memory complexity of 2^{112}. It should be noted that, if one considers removing the linear transformation from the last round similar to AES, the attack could be extended to cover seven rounds.

7 Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions that helped improve the quality of the paper. This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. AlTawy, R., and Youssef, A. M. Preimage attacks on reduced-round Stribog. In *AFRICACRYPT* (2014), D. Pointcheval and D. Vergnaud, Eds., vol. 8469 of *Lecture Notes in Computer Science*, Springer, pp. 109–125.
2. AlTawy, R., and Youssef, A. M. Second preimage analysis of Whirlwind. In *Insrypt* (2014), D. Lin, M. Yung, and J. Zhou, Eds., vol. 8957 of *Lecture Notes in Computer Science*, Springer, pp. 311–328.
3. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., and Wang, L. Preimages for step-reduced SHA-2. In *ASIACRYPT* (2009), M. Matsui, Ed., vol. 5912 of *Lecture Notes in Computer Science*, Springer, pp. 578–597.
4. Aoki, K., and Sasaki, Y. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In *CRYPTO* (2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 70–89.
5. Aoki, K., and Sasaki, Y. Preimage attacks on one-block MD4, 63-step MD5 and more. In *SAC* (2009), R. M. Avanzi, L. Keliher, and F. Sica, Eds., vol. 5381 of *Lecture Notes in Computer Science*, Springer, pp. 103–119.
6. Barreto, P., Nikov, V., Nikova, S., Rijmen, V., and Tischhauser, E. Whirlwind: a new cryptographic hash function. *Designs, Codes and Cryptography 56*, 2-3 (2010), 141–162.
7. Daemen, J., and Rijmen, V. *The Design of Rijndael: AES- The Advanced Encryption Standard.* Springer, 2002.
8. Filho, D., Barreto, P., and Rijmen, V. The Maelstrom-0 hash function. In *VI Brazilian Symposium on Information and Computer Systems Security* (2006).

9. Gauravaram, P., and Kelsey, J. Cryptanalysis of a class of cryptographic hash functions. Cryptology ePrint Archive, Report 2007/277, 2007. http://eprint.iacr.org/.
10. Gauravaram, P., and Kelsey, J. Linear-XOR and additive checksums dont protect Damgård-Merkle hashes from generic attacks. In *CT-RSA* (2008), T. Malkin, Ed., vol. 4964 of *Lecture Notes in Computer Science*, Springer, pp. 36–51.
11. Gauravaram, P., Kelsey, J., Knudsen, L. R., and Thomsen, S. On hash functions using checksums. *International Journal of Information Security 9*, 2 (2010), 137–151.
12. Gauravaram, P., Knudsen, L. R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. Grøstl a SHA-3 candidate. *NIST submission* (2008).
13. Gauravaram, P., Millan, W., Dawson, E., and Viswanathan, K. Constructing secure hash functions by enhancing Merkle-Damgård construction. In *ACISP* (2006), L. Batten and R. Safavi-Naini, Eds., vol. 4058 of *Lecture Notes in Computer Science*, Springer, pp. 407–420.
14. Guo, J., Ling, S., Rechberger, C., and Wang, H. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *ASIACRYPT* (2010), M. Abe, Ed., vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 56–75.
15. Hong, D., Koo, B., and Sasaki, Y. Improved preimage attack for 68-step HAS-160. In *ICISC* (2009), D. Lee and S. Hong, Eds., vol. 5984 of *Lecture Notes in Computer Science*, Springer, pp. 332–348.
16. Indesteege, S. The Lane hash function. Submission to NIST (2008). Avalabile at: http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf.
17. Joux, A. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO* (2004), M. Franklin, Ed., vol. 3152 of *Lecture Notes in Computer Science*, Springer, pp. 306–316.
18. Kölbl, S., and Mendel, F. Practical attacks on the Maelstrom-0 compression function. In *ACNS* (2011), J. Lopez and G. Tsudik, Eds., vol. 6715 of *Lecture Notes in Computer Science*, Springer, pp. 449–461.
19. Ma, B., Li, B., Hao, R., and Li, X. Improved cryptanalysis on reduced-round GOST and Whirlpool hash function. In *ACNS* (2014), I. Boureanu, P. Owesarski, and S. Vaudenay, Eds., vol. 8479 of *Lecture Notes in Computer Science*, Springer, pp. 289–307.
20. Matyukhin, D., Rudskoy, V., and Shishkin, V. A perspective hashing algorithm. In *RusCrypto* (2010). *(In Russian)*.
21. Mendel, F., Rechberger, C., Schlffer, M., and Thomsen, S. S. The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In *FSE* (2009), O. Dunkelman, Ed., vol. 5665 of *Lecture Notes in Computer Science*, Springer, pp. 260–276.
22. NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. In *Federal Register* (November 2007), vol. 72(212). Available at: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.
23. Rijmen, V., and Barreto, P. S. L. M. The Whirlpool hashing function. *NISSIE submission* (2000).
24. Sasaki, Y. Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In *FSE* (2011), A. Joux, Ed., vol. 6733 of *Lecture Notes in Computer Science*, Springer, pp. 378–396.

25. Sasaki, Y., and Aoki, K. Finding preimages in full MD5 faster than exhaustive search. In *EUROCRYPT* (2009), A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*, Springer, pp. 134–152.
26. Sasaki, Y., Wang, L., Wu, S., and Wu, W. Investigating fundamental security requirements on Whirlpool: Improved preimage and collision attacks. In *ASIACRYPT* (2012), X. Wang and K. Sako, Eds., vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 562–579.
27. Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full SHA-1. In *CRYPTO* (2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 17–36.
28. Wang, X., and Yu, H. How to break MD5 and other hash functions. In *EUROCRYPT* (2005), R. Cramer, Ed., vol. 3494 of *Lecture Notes in Computer Science*, Springer, pp. 19–35.
29. Wu, H. The hash function JH, 2011. Avalabile at:http://www3.ntu.edu.sg/home/wuhj/research/jh/jh-round3.pdf.
30. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., and Zou, J. (Pseudo) preimage attack on round-reduced Grøstl hash function and others. In *FSE* (2012), A. Canteaut, Ed., vol. 7549 of *Lecture Notes in Computer Science*, Springer, pp. 127–145.