Practical attacks on the Maelstrom-0 compression function

Stefan Kölbl, Florian Mendel

Graz University of Technology, A-8010 Graz, Austria
stefan.koelbl@student.tugraz.at

Abstract. In this paper we present attacks on the compression function of Maelstrom-0. It is based on the Whirlpool hash function standardized by ISO and was designed to be a faster and more robust enhancement. We analyze the compression function and use differential cryptanalysis to construct collisions for reduced variants of the Maelstrom-0 compression function. The attacks presented in this paper are of practical complexity and show significant weaknesses in the construction compared to its predecessor. The methods used are based on recent results in the analysis of AES-based hash functions.

Keywords: hash functions, cryptanalysis, collisions, near-collisions

1 Introduction

Cryptographic hash functions are a fundamental part of modern cryptography. They are used in many practical applications e.g., verification of message integrity, message authentication or secure storage of passwords. Typically a hash function is used as a digital fingerprint of the information that needs authentication.

A cryptographic hash function takes as input a string of arbitrary finite length and produce a fixed sized output. Usually the input domain is larger than the output domain, therefore this functions are many-to-one. As a result the existence of collisions is unavoidable.

Hash functions have to be both fast and secure. The security can be discussed by the following properties:

- Preimage Resistance: For a given output y it should be computationally infeasible to find an input x' such that $y = f(x')$.
- Second Preimage Resistance: For given $x, y = f(x)$ find $x' \neq x$ such that $f(x') = y$.
- Collision Resistance: Find two distinct inputs x, x' such that $f(x) = f(x')$.

A hash function with n-bit output is secure if finding a (second) preimage takes at least 2^n and finding a collision $2^{n/2}$ (birthday attack) queries [1].

The most commonly used hash functions at the moment are SHA-1, SHA-256 and SHA-512 certified by NIST. They are part of several standards and

based on MD4 and MD5. In the last few years cryptanalysis made a huge leap forward and weaknesses have been found for these functions. There are practical collisions for MD4 [2], MD5 [3] and SHA-0 [4]. The computational effort to construct collisions for SHA-1 is still impracticable, but the security bound is much lower than expected [5] and attacks on reduced rounds are possible [6]. Therefore, there is a strong interest in new hash functions.

Maelstrom-0 is based on the Whirlpool hash function which has been adopted in the ISO 10118-3:2004 standard [7]. Maelstrom-0 is designed to be a faster and more robust enhancement of Whirlpool but we will show that the new lightweight key schedule significantly weakens this hash function and allows us to construct collisions for reduced rounds of the Maelstrom-0 compression function with practical complexity. In detail, we show how to construct a collision for 6 out of 10 rounds and give a colliding message pair. Furthermore, we show attacks for 8 and 10 rounds in a weaker attack scenario (near-collision and free-start near-collision) and a theoretical collision attack on 7 rounds.

The paper is structured as follows: First there will be a description of the Maelstrom-0 hash function followed by an overview of the attack in Section 2. We continue with a detailed section on how to construct a differential path for 6-rounds and how to obtain the colliding message pair in Section 3. Afterwards possible extensions on more rounds are discussed followed by the conclusion.

2 Description of Maelstrom-0

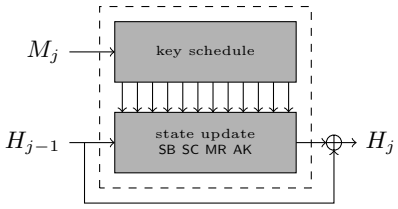

Fig. 1. Maelstrom-0 compression function

Maelstrom-0 is an iterative hash function designed by Filho, Barreto and Rijmen [8]. Maelstrom-0 processes 1024-bit message blocks and produces a 512-bit hash value. It uses the Davies-Meyer construction (see Figure 1) and 3CM chaining mode which is based on 3C [9]. If we have a message $m = M_1||M_2||\ldots||M_k$ we can compute the hash value h in the following way

$$H_0 = IV \tag{1}$$

$$H_i = E(H_{i-1}, M_i) \oplus H_{i-1}, \forall i : 0 < i \leq k \tag{2}$$

$$h = E(H_k, s_k||t_k) \tag{3}$$

where s_k is the output of the second and t_k the output of the third chain. The second chain is the XOR accumulation of all the indermediate compression function outputs and in the third chain a LFSR is involved in the accumulation process.

2.1 Block Cipher *E*

The block cipher E is based on the one used in Whirlpool. The only difference between the block cipher used for Maelstrom-0 is that it uses a different key schedule. The state update operates on 8×8 states of 64 bytes therefore the 512-bit input is bytewise transformed. The state is updated through 10 identical rounds and one key addition at the beginning. One round consists of the application of the four transformations SubBytes, ShiftColumns, MixRows and AddRoundKey similar to AES. We only give a very brief description and a more detailed one can be found in [8].

SubBytes (SB)

The SubBytes step applies a nonlinear S-box on each byte using an 8-bit S-box. The S-box is the same as in Whirlpool. For the definition of the S-box we refer to [7].

ShiftColumns (SC)

The ShiftColumns step cyclically shifts each column $j = 0, \ldots, 7$ by j steps downwards.

MixRows (MR)

MixRows is a linear mapping based on a MDS code multiplying each row by a 8×8 matrix over $\mathbb{F}_{2^8}$. The values of the matrix are choosen such that the branch number of MixRows is 9. Therefore the sum of active bytes (byte with difference) at input and output is always at least 9.

AddRoundKey (AK)

The key addition uses bitwise xor to add the round key.

Key Schedule (KS)

The key schedule takes as input the 1024-bit message block $M = (v_0, \ldots, v_{1023})$ to generate the round keys $K^0, ..., K^{10}$. The message block M is mapped to a column vector $(K^{-2} = v_0, \ldots, v_{511}, K^{-1} = v_{512}, \ldots, v_{1023})$ and in each step two new round keys are computed in the following way

$$\begin{pmatrix} K^{2i} \\ K^{2i+1} \end{pmatrix} = \alpha \begin{pmatrix} K^{2i-2} \\ K^{2i-1} \end{pmatrix} + C_i \qquad \forall i = 0, \ldots, 5 \tag{4}$$

where

$$\alpha = \begin{pmatrix} 1 & 1 \\ x^8 & x^8 + 1 \end{pmatrix} \tag{5}$$

and C_i is some round dependent constant. For the actual values we refer to [8]. There are only 11 keys needed so the last key is dropped. Multiplication is done over $\mathbb{F}_{2^{512}}[x]/p(x)$ with $p(x) = x^{512} + x^8 + x^5 + x^2 + 1$. The 512-bit round keys are transformed to a 8×8 state. For the actual round keys Maelstrom-0 uses a key extraction function which applies the SubBytes and the MixRows step to row 3 and 7 to obtain the round key. This is the only nonlinear transformation used in the key schedule. Note that inverse key schedule looks almost the same we only need to determine the inverse matrix of α which is given by

$$\alpha^{-1} = \begin{pmatrix} x^8 + 1 & 1 \\ x^8 & 1 \end{pmatrix} \tag{6}$$

3 Outline of the Attack

For our attack on Maelstrom-0 we use differential cryptanalysis. Differential cryptanalysis is a very general tool for cryptanalysis and observes how the difference between a pair of inputs affect the resulting output difference [10]. It was originally devised in the analysis of block ciphers but is also used for stream ciphers and hash functions. Usual differential cryptanalysis is a chosen plaintext attack. The basic method considers a pair of messages (M, M') and the xor difference $\Delta M = M \oplus M'$.

For our analysis we use *truncated differentials* [11]. This means we do not consider the full difference between two inputs, we only determine for single bytes whether there is a difference or not.

The structure of Maelstrom-0 allows us to predict how differences propagate through the key schedule and the round transformations to find a *good* differential path. For our goal to mount a collision attack we are looking for an input pair (M, M') with output difference zero. The attack can be divided into two individual steps. The first part is to find a differential path which holds with high probability. We use differences in the key input (message input) such that the resulting pattern can be fulfilled with a high probability. The second step is to find a message following this differential path.

Similiar attacks have been applied to Whirlpool in [12] but they did not use any difference in the key input due to the strong key schedule used in Whirlpool. Our attack is similiar to the attacks on the AES hash mode in [13] using local collisions to cancel out differences, but we use other techniques for finding the confirming message pair [13]. In the following section we define our notation and analyze the differential properties of the round transformations and the key schedule, before we describe the attack in detail.

Notation

We denote the state after round k after the transformation $R = \{SB, SC, MR, AK\}$ by $R^k_{i,j}$ where i, j are the row and column indices. Indices are used modulo 8.

3.1 Differential properties of the round transformations

SB - SubBytes. For SubBytes we consider pairs of input/output differences $\Delta a, \Delta b \in \{0,1\}^8$. Counting over all 2^{16} possible differentials the number of solutions for

$$SB(X) \oplus SB(X \oplus \Delta a) = \Delta b \tag{7}$$

can only be $0, 2, 4, 6, 8, 256$. We are interested if a given input difference can propagate to a given output difference. Counting over all possible inputs the probability for Δa to propage to Δb through SubBytes is equal to 0.395 respectively there are about 101 valid transitions on average from Δa to another difference. This can be computed by creating a difference distribution table (DDT) of size 256×256 for all possible values.

SC - ShiftColumns. This steps moves differences to different rows but the values are not changed. The 8 bytes of a full active row are moved to 8 different rows.

MR - MixRows. MixRows is a linear step, hence xor differences propagate in a deterministic way. For truncated differences we only got the position of the difference and the propagation through MixRows is probabilistic. Since the branch number of MixRows is 9, one active byte will propagate to 8 active bytes with probability 1. The probability for a transition from a to b active bytes with $1 \leq a, b \leq 8$ and $a + b \geq 9$ is in general $2^{(b-8)\cdot 8}$.

AK - AddRoundKey. AddRoundKey uses simple xor to add the round key hence difference propagate deterministic through this operation.

3.2 Differential properties of the key schedule

The key schedule uses two 512-bit keys K^{-2}, K^{-1} to compute the next two round keys. Apart from the key extraction function the key schedule is linear. First we can simplify the key schedule and ignore the addition of the round constant and look at the two new keys separately.

$$K^0 = K^{-2} + K^{-1}$$
$$K^1 = x^8 K^{-2} + (x^8 + 1) K^{-1}$$

The difference propagation to K^0 is trivial because it simply xors the two input keys. For the second key we have to look how multiplication over $\mathbb{F}_{2^{512}}[x]/p(x)$ influences the differences. Multiplication by x^8 equals shifting bytewise and adding the irreducible polynomial $p(x)$ depending on the values of the first byte ($K_{0,0}$) of each round key. If we avoid any differences in the first byte the differences will just be shifted bytewise (see Figure 2). If we have a difference in the first byte it will only affect the last two bytes due to the structure of $p(x)$.

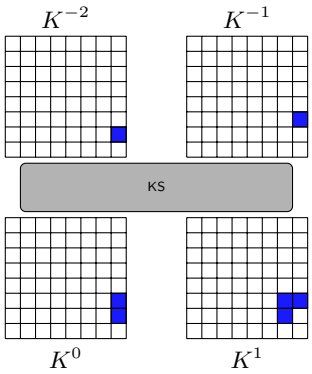

Fig. 2. Example difference propagation through the keyschedule with differences in $\mathrm{K}_{6,7}^{-2}$ and $K_{5,7}^{-1}$. Colored bytes denote differences

Key extraction function - ψ

The key extraction function applies SubBytes and MixRows to row 3 and 7 and copies all other rows. Note that ψ is only applied on the round keys and does not influence the state of the key schedule. For our attack we avoid difference in this rows so we can ignore it for our further analysis.

3.3 Constructing the differential path

Constructing the path is rather simple due to the structure of the round transformations. Using truncated differentials we can construct a good path by hand.

For the 6-round differential path (see Figure 3) we start with a difference in $\mathrm{K}_{6,7}^{1}$. We can ignore K^{-2} and K^{-1} because we can apply the inverse key schedule to $(\mathrm{K}^0, \mathrm{K}^1)$ to obtain the initial key. The difference introduced by K^1 will end up in a full active row in AK^2 due to the properties of MixRows. After SC^3 there will be an active byte in every row and therefore we will have a full active state in AK^3. We keep this full active state for AK^4 and use the properties of MixRows to obtain the pattern in AK^5. Note that in row 6 we want the 8 active bytes to propagate to 4 active bytes so that they are canceled out with the differences in K^5. We choose this pattern to obtain a single active row after applying ShiftColumns so that we can use the last MixRows to cancel out the differences in K^6. The number of active bytes for the rounds are:

$$0 - 1 - 9 - 64 - 64 - 8 - 0 \tag{8}$$

In the next step, we determine the xor differences of the differential path. For this we use the same approach that has been used in the rebound attack on Whirlpool [14]. So far we only considered truncated differentials, now we use xor differences.

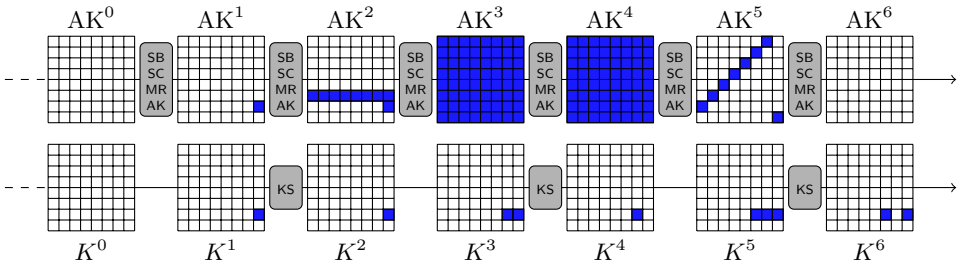

Fig. 3. 6 round differential path for Maelstrom-0

Backward direction

1. Start with an arbitrary difference in $\mathrm{K}^6_{6,5}, \mathrm{K}^6_{6,7}$. We want to cancel this difference out so we also fix the difference in $\mathrm{MR}^5_{6,5}, \mathrm{MR}^5_{6,7}$.
2. The next step is to compute the MixRows step backwards to obtain the differences in SC^5 respectively SB^5. To compute backwards the SubBytes step we choose an arbitrary possible input difference to the given output difference using the difference distribution table (see Section 3.1).
3. Now we repeat this steps to propagate the differences backward through $\mathrm{MR}^4, \mathrm{SC}^4$. The differences after SB^4 are now fixed and in the following steps we show how we can propagate the difference from the start such that they match this fixed difference.

Forward direction

1. Start with the difference in $\mathrm{AK}^1_{6,7}$ and propagate it forward. We get a full active row in MR^1. For the next SubBytes step we choose arbitrary possible output differences using the DDT to the 9 given input differences and we end up with a full active state in MR^2 and AK^2.
2. Now we start the matching process for each row $r = 0, \ldots, 7$ individual. Select the bytes $\mathrm{AK}^2_{(8-r),i}$ for $i = 0, \ldots, 7$ and it follows that this bytes form a single row after ShiftColumns. Propagate the bytes through $\mathrm{SB}^3, \mathrm{SC}^3, \mathrm{MR}^3, \mathrm{AK}^3$ step. For the SubBytes step we use a random valid output difference.
   (a) Check for all bytes in row r of AK^3 if there is a valid transition from the input difference to the output difference in SB^4. For any given input difference we get up to 114 possible output difference. On average the probability that the differences in one byte transition is valid is 0.395 and the total probability is $\approx 2^{-10.72}$ that a full row matches. In practice we can improve this by using favorable differences. If we do not get a match for the row, we just choose another output difference in the previous step. For every byte we can choose from at least 89 possible output differences, therefore we can easily generate enough solutions to find a match.
   (b) Repeat the steps until all conditions are fulfilled. The expected costs are $8 \cdot 2^{10.72} = 2^{13.72}$

After finishing this steps the differential path is fully determined.

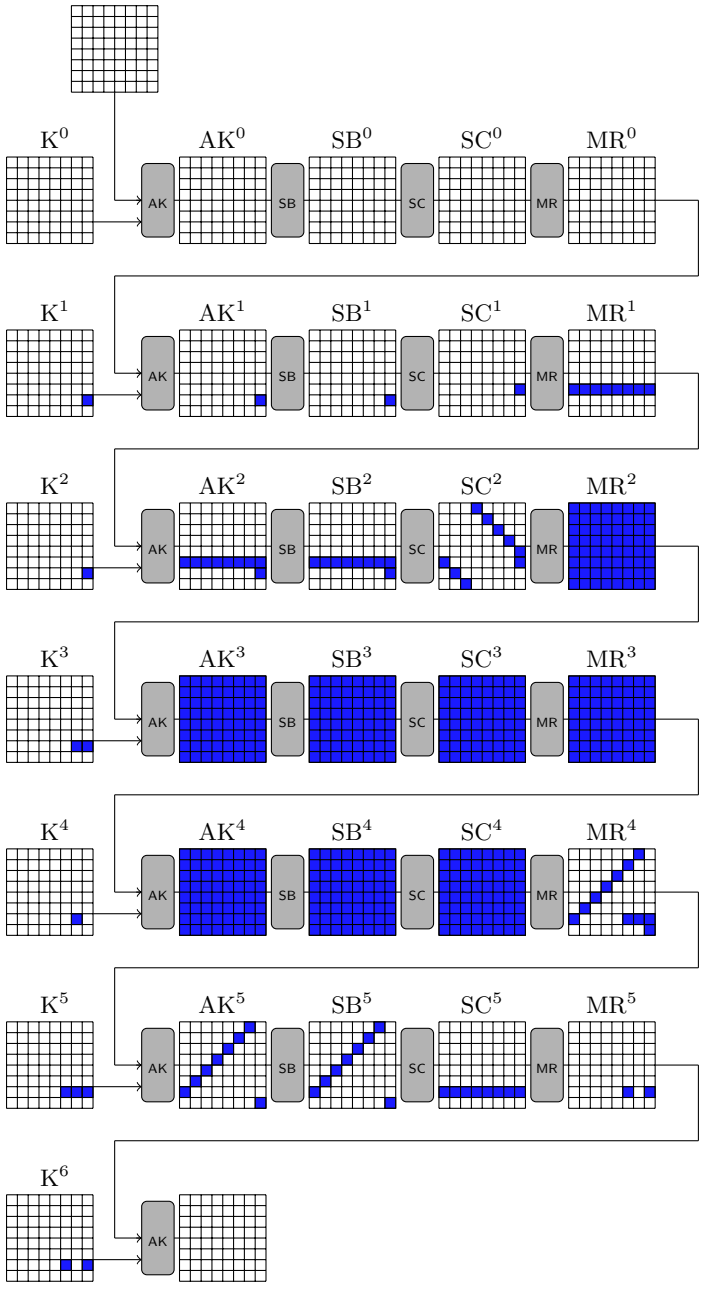

Fig. 4. Full differential path for 6 rounds using truncated differentials.

3.4 Finding the message pair

To find the colliding message pair we need to find a message that follows the previous differential path. We need to determine the message $M = K^{-2}||K^{-1}$. This can be done by random trials which obviously leads to a high complexity. A more efficient method is to use the triangulation algorithm invented by Khovratovich et al. in the cryptanalysis of AES in hash mode [13]. Due to the structure of Maelstrom-0 the best result we achieved had a complexity of 2^{184} which is in fact lower than the theoretical bound but we used the following approach which turns out to be very efficient for Maelstrom-0.

1. Start at SB^4 and determine the correct values for the given differences. We need to fulfill the conditions on all 64 active bytes, but there are no restrictions yet so we can choose the right pair of values.
2. For SB^5 we got 8 conditions but we can use the according bytes in K^5 to get the correct input values to this SubBytes layer. The values for the rows affected by the key extraction function can be computed by inverting the MixRows and SubBytes step to obtain the desired value.
3. Now we need to satisfy the conditions for the fourth SubBytes layer. We can compute the values from the difference for SB^3 and apply ShiftColumns, MixRows and use K^4 to correct them and get the right input to the fourth SubBytes layer. After this step all values of K^4 and 8 bytes of K^5 are fixed.
4. Compute $\mathrm{K}^2, \mathrm{K}^3$ by applying the inverse KeySchedule. From the inverse KeySchedule it follows that K^5 is xored to K^4 in both cases. We can use the remaining 54 free bytes now to influence K^3 and therefore the values we need to satisfy the 9 byte conditions of the third SubBytes layer. By changing single bytes in K^3 we can influence the value of each active byte in SB^2. The probability that we get the right value is 2^{-8}. There are still 7 bytes that are not fixed in every row of K^5 so we can create 2^{56} possible values and are guarenteed to find a solution. Each row in K^5 only affects one respectively two active bytes in SB^2 therefore we can find the right values for each row individually. With the naive approach by trying out different values for each row we get a total complexity of $\mathcal{O}(2^{16})$.
5. With only one byte condition left in SB^1 to fulfill we can simply bruteforce the last S-box and repeat the previous steps. This results in a semi-free start collision with a complexity of 2^{24}.

A colliding message pair for 6 rounds is given in the Appendix A.

3.5 Extension to more rounds

In this section, we show how the attack can be extended to more rounds. It is possible to construct collisions for 7 rounds. This could be achieved by using a different path with another state between the two full active ones. This leads to

$$0 - 1 - 9 - 64 - 8 - 64 - 8 - 0 \tag{9}$$

active bytes. The attack can be constructed similar to the attack on 6 rounds, but we have more conditions which can not be fulfilled using the message input. A possible approach would be:

1. Fix the differences in AK^4 and propagate them backward to SB^3. Fix the differences in SB^2 and propagate them forward to AK^3. We try to find a match with the same method used in the 6 round attack here.
2. Choose a valid transition for the propagation from AK^4 to SB^4. Compute the differences forward to AK^5. Now choose a difference in AK^6 and propagate it backward to SB^5 and try to find a match.

Finding the message can now be done in the following way:

1. Determine the values at AK^3 resp. SB^3. We got 8 byte conditions at SB^4 which can be fulfilled using the corresponding column in K^4. The 64 byte conditions for SB^5 can be fulfilled using K^5. Following onward this direction we need one transition of 8 to 3 active bytes in MR^6 which costs 2^{40} and we need them to cancel out with the last 3 bytes of K^7. So the total probability for the backwards direction is 2^{64}.
2. In forward direction this looks very similar. We got one 8 to 1 transition in MR^1 which costs 2^{56} and two conditions for the bytes $K^1_{6,7}$ and $K^2_{6,7}$ to cancel out. Therefore the probability in forward direction is 2^{72}.
3. The total probability is 2^{136} in this case.

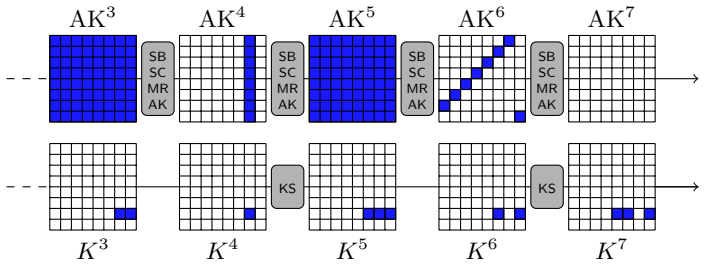

Fig. 5. 7 round differential path. The first three rounds are omitted and are the same as for the 6 round path.

By choosing a weaker attack scenario the previous collision attack on 6 rounds can be extended to more rounds. If we allow differences in the output we get near-collisions and additional differences at the input lead to free-start near-collisions.

Adding two more rounds after the 6-round attack we get near-collisions for 8 rounds with the same complexity. The key addition after round 6 leads to 3 active bytes which propagate to 3 active rows. The last key adds another active byte.

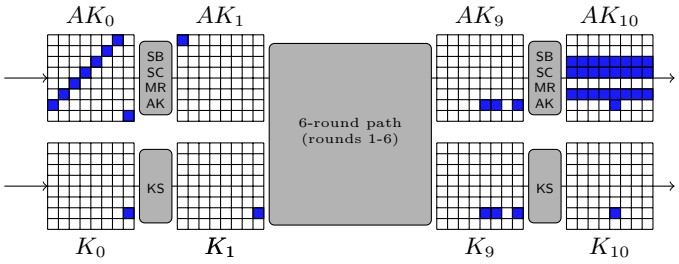

Fig. 6. Extending the differential path at the end to get a near-collisions for 8 rounds with 25 active bytes. Furthermore, adding 2 rounds at the beginning for full 10-rounds free-start near-collisions.

This gives us a near-collision with 25 active bytes (see Figure 6). Furthermore it is also possible to additionally prepend two rounds to construct a free-start near-collisions for the full 10 rounds of Maelstrom-0.

Attack	rounds	complexity	generic attack
semi-free-start collision	6	2^{24}	2^{256}
semi-free-start collision	7	2^{136}	2^{256}
semi-free-start near-collision	8	2^{24}	2^{156}
free-start near-collision	10	2^{24}	2^{124}

Table 1. Summary of attacks

4 Conclusion

In this paper, we have shown how to construct collisions for the Maelstrom-0 compression function. Maelstrom-0 was designed to be faster and more robust, however the new lightweight key schedule significantly weakens the compression function and allows efficient attacks. The linear key schedule allows to construct good differential paths and the key extraction function can be easily avoided. We can construct collisions for 6 and 7 rounds. Furthermore, the attack scenario can be extended to full Maelstrom-0, resulting in a free-start near-collision. In Table 1 we summarize our results for the Maelstrom-0 compression function.

Acknowledgments

We would like to thank the designers of Maelstrom-0 for providing a reference implementation and Matthias Schlaipfer for providing a implementation of the

rebound attack for Whirlpool which helped us to verify our results. The work in this paper has been supported in part by the Austrian Science Fund (FWF), project P21936-N23 and by the European Commission under contract ICT-2007-216646 (ECRYPT II).

References

1. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
2. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of LNCS., Springer (2005) 1–18
3. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of LNCS., Springer (2005) 19–35
4. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In Shoup, V., ed.: CRYPTO. Volume 3621 of LNCS., Springer (2005) 1–16
5. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full sha-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
6. De Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Adams, C.M., Miri, A., Wiener, M.J., eds.: Selected Areas in Cryptography. Volume 4876 of LNCS., Springer (2007) 56–73
7. Barreto, P.S.L.M., Rijmen, V.: The Whirlpool Hashing Function. Submitted to NESSIE, September 2000, revised May 2003 (2000) Available online: http://www.larc.usp.br/ pbarreto/WhirlpoolPage.html.
8. Filho, D.G., Barreto, P.S., Rijmen, V.: The maelstrom-0 hash function. In: SBSeg 2006. (2006)
9. Gauravaram, P., Millan, W., Dawson, E., Viswanathan, K.: Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction. In Batten, L.M., Safavi-Naini, R., eds.: ACISP. Volume 4058 of Lecture Notes in Computer Science., Springer (2006) 407–420
10. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptology **4**(1) (1991) 3–72
11. Knudsen, L.R.: Truncated and higher order differentials. In Preneel, B., ed.: FSE. Volume 1008 of Lecture Notes in Computer Science., Springer (1994) 196–211
12. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full whirlpool compression function. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 126–143
13. Khovratovich, D., Biryukov, A., Nikolic, I.: Speeding up collision search for byte-oriented hash functions. In Fischlin, M., ed.: CT-RSA. Volume 5473 of Lecture Notes in Computer Science., Springer (2009) 164–181
14. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In Dunkelman, O., ed.: FSE. Volume 5665 of Lecture Notes in Computer Science., Springer (2009) 260–276

A Colliding Message Pair

Here a colliding message pair (M, M') and the chaining value are given. The message pair has been found by using the 6-round path and the difference in the

messages are $\Delta M^1_{6,7} = \Delta M^2_{6,7} = 01$. Values are given in hex notation.

Chaining Value	After 6 rounds (AK^6)
62 c4 11 cf 0e 4e dd eb	6d 85 84 15 32 bd fc 98
7e 1f 07 7c d7 84 ae 56	b6 db 17 12 ed c5 fe 73
a4 81 51 b2 1e 91 d3 fe	f5 85 8e a7 93 ea b0 87
23 08 cd 4a b8 d4 82 b9	ac 8e da b0 e1 20 82 d8
67 89 16 74 f0 e6 7d 58	15 32 a8 61 d5 3f bc 93
76 e0 fa f9 b6 8b 01 9c	ba dd 0a 2b bb 20 87 1f
83 d8 d8 36 e3 9e 54 f2	32 45 86 6a c2 41 73 df
43 0c 85 58 a0 9b 30 38	34 81 63 4e 4a 10 18 a7
M	M'
25 fe e7 fa 16 6f 30 2b	25 fe e7 fa 16 6f 30 2b
c3 03 8e d9 79 3a d6 06	c3 03 8e d9 79 3a d6 06
8e 53 d3 da 9b 41 33 e0	8e 53 d3 da 9b 41 33 e0
66 e6 da 06 5c 9b f1 f2	66 e6 da 06 5c 9b f1 f2
31 1a ff 5c a1 ac 25 cd	31 1a ff 5c a1 ac 25 cd
2f 6e 63 a9 84 0e d5 40	2f 6e 63 a9 84 0e d5 40
00 c0 d9 9f 24 ab 7c 20	00 c0 d9 9f 24 ab 7c 21
1f 2f d8 2f bc d2 04 2a	1f 2f d8 2f bc d2 04 2a
34 8c 53 c5 17 b4 87 35	34 8c 53 c5 17 b4 87 35
e1 9c 2c e8 1d fb df 80	e1 9c 2c e8 1d fb df 80
97 3d 46 0f ee 1d 5d 4b	97 3d 46 0f ee 1d 5d 4b
63 55 37 c3 de 04 88 8e	63 55 37 c3 de 04 88 8e
b8 13 92 12 2c d2 8d 8e	b8 13 92 12 2c d2 8d 8e
ef 3b fc 5a b3 44 6b 7b	ef 3b fc 5a b3 44 6b 7b
ef f6 80 42 49 9a 5d de	ef f6 80 42 49 9a 5d df
9f 1b d8 e9 88 7f c4 73	9f 1b d8 e9 88 7f c4 73