Rebound Attack on the Full LANE Compression Function *

Krystian Matusiewicz[1], María Naya-Plasencia[2], Ivica Nikolić[3], Yu Sasaki[4], and Martin Schläffer[5]

[1] Department of Mathematics, Technical University of Denmark, Denmark
[2] INRIA project-team SECRET, France
[3] University of Luxembourg, Luxembourg
[4] NTT Corporation, Japan
[5] IAIK, Graz University of Technology, Austria
k.matusiewicz@mat.dtu.dk,maria.naya_plasencia@inria.fr,
ivica.nikolic@uni.lu,sasaki.yu@lab.ntt.co.jp,
martin.schlaeffer@iaik.tugraz.at

Abstract. In this work, we apply the rebound attack to the AES based SHA-3 candidate LANE. The hash function LANE uses a permutation based compression function, consisting of a linear message expansion and 6 parallel lanes. In the rebound attack on LANE, we apply several new techniques to construct a collision for the full compression function of LANE-256 and LANE-512. Using a relatively sparse truncated differential path, we are able to solve for a valid message expansion and colliding lanes independently. Additionally, we are able to apply the inbound phase more than once by exploiting the degrees of freedom in the parallel AES states. This allows us to construct semi-free-start collisions for full LANE-256 with 2^{96} compression function evaluations and 2^{88} memory, and for full LANE-512 with 2^{224} compression function evaluations and 2^{128} memory.
Keywords: SHA-3, LANE, hash function, cryptanalysis, rebound attack, semi-free-start collision

1 Introduction

In the last few years the cryptanalysis of hash functions has become an important topic within the cryptographic community. The attacks on the MD4 family of hash functions (MD5, SHA-1) have especially weakened the confidence in the security of this design strategy [13,14]. Many new and interesting hash function designs have been proposed as part of the NIST SHA-3 competition [11]. The large number of submissions and different design strategies require different and improved cryptanalytic techniques as well.

* This work was supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The authors would like to thank Janusz Szmidt and Florian Mendel for useful discussions.

M. Matsui (Ed.): ASIACRYPT 2009, LNCS 5912, pp. 106–125, 2009.
The original publication is available at http://www.springerlink.com/content/q1867gk784v2mp8q/

At FSE 2009, Mendel *et al.* published the rebound attack [9] - a new technique for analysis of hash functions which has been applied first to reduced versions of the Whirlpool [2] and `Grøstl` [4] compression functions. Recently, the rebound attack on Whirlpool has been extended in [8], which in some parts is similar to our attack. The main idea of the rebound attack is to use the available degrees of freedom in the internal state to efficiently fulfill the low probability parts in the middle of a differential trail. The straight-forward application of the rebound attack to AES based constructions allows a quick and thorough analysis of these hash functions.

In this work, we improve the rebound attack and apply it to the SHA-3 candidate LANE. The hash function LANE [5] uses an iterative construction based on the Merkle-Damgård design principle [3,10] and has been first analyzed in [15]. The permutation based compression function consists of a linear message expansion and 6 parallel lanes. The permutations of each lane are based on the round transformations of the AES. In the rebound attack on LANE, we first search for differences and values, according to a specific truncated differential path. This truncated differential path is constructed such that a collision and a valid expanded message can be found with a relatively high probability. By using the degrees of freedom in the chaining values, we are able to construct a semi-free-start collision for the full versions of LANE-256 with 2^{96} compression function evaluations and memory of 2^{88}, and for LANE-512 with 2^{224} compression function evaluations and memory of 2^{128}. Although these collisions on the compression function do not imply an attack on the hash functions, they violate the reduction proofs of Merkle and Damgård, and Andreeva [1].

2 Description of LANE

The cryptographic hash function LANE [5] is one of the submissions to the NIST SHA-3 competition [11]. It is an iterated hash function that supports four digest sizes (224, 256, 384 and 512 bits) and the use of a salt. Since LANE-224 and LANE-256 are rather similar except for truncation, we write LANE-256 whenever we refer to both of them. The same holds for LANE-384 and LANE-512.

The hashing of a message proceeds as follows. First, the initial chaining value H_{-1}, of size 256 bits for LANE-256, and 512 bits for LANE-512, is set to an initial value that depends on the digest size n and the optional salt value S. At the same time, the message is padded and split into message blocks M_i of length 512 bits for LANE-256, and 1024 bits for LANE-512. Then, a compression function f is applied iteratively to process message blocks one by one as $H_i = f(H_{i-1}, M_i, C_i)$, where C_i is a counter that indicates the number of message bits processed so far. Finally, after all the message blocks are processed, the final digest is derived from the last chaining value, the message length and the salt by an additional call to the compression function.

2.1 The Compression Function

The compression function of LANE-256 transforms 256 bits (512 in the case of LANE-512) of the chaining value and 512 bits (resp. 1024 bits) of the message block into a new chaining value of 256 bits (512 bits). It uses a 64-bit counter value C_i. For the detailed structure of the compression function we refer to the specification of LANE [5]. First, the chaining value and the message block are processed by a message expansion that produces an expanded state with doubled size. Then, this expanded state is processed in two layers. The first layer is composed of six permutation lanes $P_0,\ldots,P_5$ in parallel, and the second layer of two parallel lanes Q_0, Q_1.

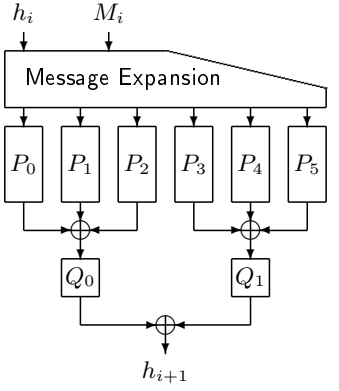

Fig. 1. The compression function of LANE.

function ROUND(r,X)
  $X \leftarrow$ SubBytes(X)
  $X \leftarrow$ ShiftRows(X)
  $X \leftarrow$ MixColumns(X)
  $X \leftarrow$ AddConstant(r, X)
  $X \leftarrow$ AddCounter(r, X)
  $X \leftarrow$ SwapColumns(X)
end function

Fig. 2. Pseudocode for the round transformation used in the LANE permutations.

2.2 The Message Expansion

The message expansion of LANE takes a message block M_i and a chaining value H_{i-1} and produces the input to six permutations $P_0,\ldots,P_5$. In LANE-256, the 512-bit message block M_i is split into four 128-bit blocks m_0, m_1, m_2, m_3 and the 256-bit chaining value H_{i-1} is split into two 128-bit words h_0, h_1 as follows $m_0||m_1||m_2||m_3 \leftarrow M_i$, $h_0||h_1 \leftarrow H_{i-1}$. Then, six more 128-bit words $a_0, a_1, b_0, b_1, c_0, c_1$ are computed

$$\begin{aligned} a_0 &= h_0 \oplus m_0 \oplus m_1 \oplus m_2 \oplus m_3\,, \quad a_1 = h_1 \oplus m_0 \oplus m_2\ , \\ b_0 &= h_0 \oplus h_1 \oplus m_0 \oplus m_2 \oplus m_3\,, \quad b_1 = h_0 \oplus m_1 \oplus m_2\ , \\ c_0 &= h_0 \oplus h_1 \oplus m_0 \oplus m_1 \oplus m_2\,, \quad c_1 = h_0 \oplus m_0 \oplus m_3\ . \end{aligned} \tag{1}$$

Each of these 128-bit values, as in AES, can be seen as 4×4 matrix of bytes. In the following, we will use the notion $x[i,j]$ when we refer to the byte of the matrix x with row index i and column index j, starting from 0.

The values $a_0||a_1$, $b_0||b_1$, $c_0||c_1$, $h_0||h_1$, $m_0||m_1$, $m_2||m_3$ become inputs to the six permutations $P_0, \ldots, P_5$ described below. The message expansion for larger variants of LANE is identical but all the values are doubled in size.

2.3 The Permutations

Each permutation lane P_i operates on a state that can be seen as a double AES state (2×128-bits) in the case of LANE-256 or quadruple AES state (4×128-bits) for LANE-512. The permutation reuses the transformations SubBytes (SB), ShiftRows (SR) and MixColumns (MC) of the AES with the only exception, that due to the larger state size, they are applied twice or four times in parallel.

Additionally, there are three new round transformations introduced in LANE. AddConstant adds a different value to each column of the lane state and AddCounter adds part of the counter C_i to the state. Since our attacks do not depend on these functions, we skip their details here. The third transformation is SwapColumns (SC) - used for mixing parallel AES states. Let x_i be a column of a lane state. In LANE-256, SwapColumns swaps the two right columns of the left half-state with the two left columns of the right half-state, and in LANE-512, SwapColumns ensures that each column of an AES state gets swapped to a different AES state:

$$\begin{aligned} SC_{256}(x_0||x_1||\ldots||x_7) &= x_0||x_1||x_4||x_5||x_2||x_3||x_6||x_7 \\ SC_{512}(x_0||x_1||\ldots||x_{15}) &= x_0||x_4||x_8||x_{12}||x_1||x_5||x_9||x_{13}|| \\ &\quad x_2||x_6||x_{10}||x_{14}||x_3||x_7||x_{11}||x_{15} \ . \end{aligned}$$

The complete round transformation consists of the sequential application of all these transformations in the given order. The last round omits AddConstant and AddCounter. Each of the permutations P_j consists of six rounds in the case of LANE-256 and eight rounds for LANE-512.

The permutations Q_0 and Q_1 are irrelevant to our attack because we will get collisions before these permutations. An interested reader can find a detailed description of Q_0 and Q_1 in [5].

3 The Rebound Attack on LANE

In this section first we give a short overview of the rebound attack in general and then, describe the different phases of the rebound attack on LANE in detail.

3.1 The Rebound Attack

The rebound attack was published by Mendel *et al.* in [9] and is a new tool for the cryptanalysis of hash functions. The rebound attack uses truncated differences [6] and is related to the attack by Peyrin [12] on the hash function Grindahl [7]. The main idea of the rebound attack is to use the available degrees of freedom in the internal state to fulfill the low probability parts in the middle of a differential path. It consists of an inbound and subsequent outbound phase.

The inbound phase is an efficient meet-in-the-middle phase, which exploits the available degrees of freedom in the middle of a differential path. In the mostly probabilistic outbound phase, the matches of the inbound phase are computed backwards and forwards to obtain an attack on the hash or compression function. Usually, the inbound phase is repeated many times to generate enough starting points for the outbound phase. In the following, we describe the inbound and outbound phase of the rebound attack on LANE.

3.2 Outline of the Rebound Attack on LANE

Due to the message expansion of LANE, at least 4 lanes are active in a differential attack. We will launch a semi-free-start collision attack, and therefore we assume the differences in (h_0, h_1) to be zero. Hence, lane P_3 is not active and we choose P_1 and thus, (b_0, b_1) to be not active as well. The active lanes in our attack on LANE are P_0, P_2, P_4 and P_5. The corresponding truncated differential path for the P-lanes of LANE-256 is shown in Fig. 4. This path is very similar to the truncated differential path for LANE-256 shown in the LANE specification [Fig. 4.2, page 33], but turned upside-down. The truncated differential path used in the attack on LANE-512 is the same as in the LANE specification [Fig. 4.3, page 34] and shown in Fig. 5. The main idea of these paths is to use differences in only one of the parallel AES states for the inbound phases. This allows us to use the freedom in the other states to satisfy the outpound phases. Since we search for a collision after the P-lanes, we do not need to consider the Q-lanes.

The main idea of the attack on LANE is that we can apply more than one efficient inbound phase by using the degrees of freedom and the relatively slow diffusion due to the 2 (or 4) parallel AES states of LANE-256 (or LANE-512). The positions of the active bytes of two consecutive inbound phases are chosen such that when merging them, the number of the common active bytes of these phases is as small as possible. Since we can find many independent solutions for these inbound phases, we store them in some lists to be merged. In the outbound phase of the attack we merge the results of the inbound phases and further, merge the results of all active P-lanes. Note that the merging of two lists can be done efficiently. In each merging step, a number of conditions need to be fulfilled for the elements of the new list. We merge the lists in a clever order, such that we find one colliding pair for the compression function at the end.

In more detail, we first filter the results of each inbound phase for those solutions, which can connect both inbound phases (see Fig. 4). Then, we merge the resulting lists of two lanes such that we get a collision after the P-lanes, and parts of the message expansion are fulfilled. Finally, we filter the results of the left P-lanes (P_0, P_2) and the right P-lanes (P_4, P_5), such that the conditions on the whole message expansion are fulfilled. In the attack, we try to keep the size of the intermediate results at a reasonable size. We need to ensure, that the complexity of generating the lists is below $2^{n/2}$, but still get enough solutions in each phase to continue with the attack.

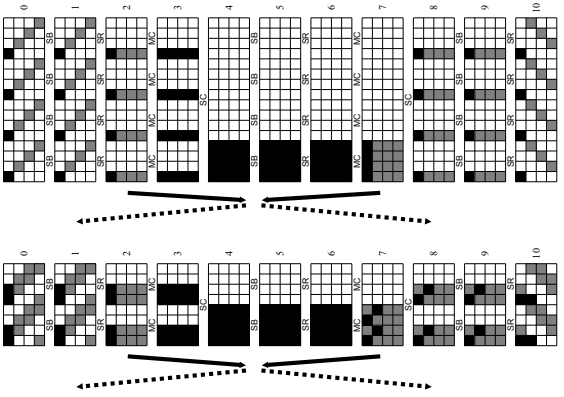

Fig. 3. The inbound phase for Lane-256 (left) and Lane-512 (right). Black bytes are active, gray bytes fixed by solutions of the inbound phase.

3.3 The Inbound Phase

In the rebound attack on Lane, we first apply the inbound phase for a number of times. Therefore, we will explain this phase and the corresponding probabilities in detail here. In the inbound phase, we search for differences and values conforming to the truncated differential path for Lane-256 or Lane-512 shown in Fig. 3, with active bytes marked by black bytes. We only describe the application of one inbound phase here. In the example of Fig. 3, we have 16 active S-boxes between state #4 and state #5. It follows from the MDS property of MixColumns, that this path has at least one active byte in each of the 4 corresponding columns prior to the first, and after the second MixColumns transformation (state #2 and state #7). Note that the active bytes in state #2 and state #7 can also be at any position marked by gray bytes.

In the inbound phase, we first choose random differences for the 4 active bytes after the second MixColumns transformation (state #7). These differences are linearly propagated backward to 16 active bytes at the output of the previous SubBytes layer (state #5). Next, we take random differences for the 4 active bytes prior to the first MixColumns transformation (state #2) and linearly propagate forward to 16 active bytes at the input of SubBytes (state #4). Then, we need to find a match for the input and output differences of all 16 active S-boxes. For a single S-box, the probability that a random S-box differential exists is about one half, which can be verified easily by computing the differential distribution table of the AES S-box (see [9] for more details).

For each matching S-box, we get at least two (in some cases 4) possible byte values such that the S-box differential holds. Hence, we get at least 2^{16} possible values for one full AES state, such that the differential path for the chosen differences in state #2 and state #7 holds. In other words, after trying 2^{16} non-zero differences of state #2 and state #7, we get at least 2^{16} solutions for the truncated differential path between state #2 and state #7. Hence, the

average complexity to find one solution for the inbound phase (differences and values) is about 1. Note that this holds for both, Lane-256 and Lane-512.

3.4 The Outbound Phase

After we have found differences and values for each inbound phase of the active lanes, we need to connect these results and propagate them outwards in the outbound phase. In backward direction, we need to match the message expansion at the input of each lane. In forward direction, we need to match the differences of two P-lanes on each side to get a collision. We describe the conditions for these two parts according to our truncated differential path in the following.

The Message Expansion. After the inbound phases, we get values and differences at the input and output of the 4 active lanes P_0, P_2, P_4 and P_5. Since we have zero differences in (h_0, h_1) and (b_0, b_1), we get using the message expansion for lane P_1 (see Equation (1)):

$$\Delta b_0 = 0 = \Delta m_0 \oplus \Delta m_2 \oplus \Delta m_3\,, \quad \Delta b_1 = 0 = \Delta m_1 \oplus \Delta m_2$$

Hence, we get the following relation for the message differences in m_0, m_1, m_2, and m_3:

$$\Delta m_1 = \Delta m_2 = \Delta m_0 \oplus \Delta m_3 \tag{2}$$

Using (1) we get for the differences in the expanded message words (a_0, a_1) and (c_0, c_1):

$$\Delta a_0 = \Delta m_1\,,\ \Delta a_1 = \Delta m_3\,,\ \Delta c_0 = \Delta m_0\,,\ \Delta c_1 = \Delta m_2 \tag{3}$$

and thus, the following relations between a_0, a_1, c_0, and c_1:

$$\Delta a_0 = \Delta c_1 = \Delta a_1 \oplus \Delta c_0 \tag{4}$$

Beside the differences, we also need to match the values of the message expansion. Since we aim for a semi-free-start collision, we can freely choose the chaining value (h_0, h_1) such that the conditions on (a_0, a_1) are satisfied:

$$h_0 = a_0 \oplus m_0 \oplus m_1 \oplus m_2 \oplus m_3\,, \quad h_1 = a_1 \oplus m_0 \oplus m_2$$

That means we have conditions on the input (c_0, c_1) left, which we need to match with the message words m_0, m_1, m_2 and m_3. Since we can vary lanes P_0,P_2 and P_4,P_5 independently in the following attacks, we can satisfy these conditions by merging the results of both sides. Using the equations of the message expansion, we get for (c_0, c_1) using the values of (a_0, a_1):

$$c_0 = a_0 \oplus a_1 \oplus m_0 \oplus m_2 \oplus m_3\,, \quad c_1 = a_0 \oplus m_1 \oplus m_2$$

We can rearrange these equations in order to have all terms corresponding to P_0,P_2 on the left side and all terms of P_4,P_5 on the right side:

$$m_0 \oplus m_2 \oplus m_3 = c_0 \oplus a_0 \oplus a_1\,, \quad m_1 \oplus m_2 = c_1 \oplus a_0 \tag{5}$$

For merging the two sides, we will compute, store and compare the following values of each list:

$$v_1 = c_0 \oplus a_0 \oplus a_1\,, \quad v_2 = c_1 \oplus a_0\,, \qquad v_3 = m_0 \oplus m_2 \oplus m_3\,, \quad v_4 = m_1 \oplus m_2$$

Colliding P-Lanes. In the forward direction, we need to find a collision for the differences in P_0 and P_2, such that $\Delta P_0 \oplus \Delta P_2 = 0$ and for the differences in P_4 and P_5, such that $\Delta P_4 \oplus \Delta P_5 = 0$. Note that we can swap the order of the last MixColumns with the XOR operation of the P-lanes since both transformations are linear. Hence, we only need to match the differences after the last SubBytes layer in each of the two active lanes. The blue bytes in Fig. 4 of LANE-256, or the red, blue and yellow bytes in Fig. 5 of LANE-512 are independent of the inbound phase. Hence, we can use the freedom in these bytes to find a collision after the P-lanes.

4 Semi-Free-Start Collision for LANE-256

In the rebound attack on LANE-256, we construct a semi-free-start collision for the full compression function using 2^{96} compression function evaluations and memory requirements of 2^{88}. We will use the 6-round truncated differential path given in Fig. 4 which is very similar to the one shown in the LANE specification [Fig. 4.2, page 33]. We search for a collision after the P-lanes of LANE and use the same truncated differential path in the 4 active lanes P_0, P_2, P_4 and P_5. Since we do not consider differences in h_0 and h_1, but we fix their values, the result will be a semi-free-start collision. The attack on LANE-256 consists basically of the following parts:

1. **First Inbound Phase:** Apply the inbound phase at the beginning of the truncated differential path (state #2 to state #7) for each lane P_0, P_2, P_4, P_5 independently.
2. **Second Inbound Phase:** Apply the inbound phase in the middle of each lane again (state #10 to state #15).
3. **Merge Inbound Phases:** Merge the results of the two inbound phases (state #7 to state #10).
4. **Merge Lanes:** Merge the two neighboring lanes P_0,P_2 and P_4,P_5 and satisfy according differences of the message expansion.
5. **Message Expansion:** Merge the two sides (P_0, P_2) and (P_4, P_5) and satisfy the remaining conditions on the message expansion (differences and values).
6. **Find Collisions:** Choose remaining free values (neutral bytes) to find a collision for each side (P_0, P_2) and (P_4, P_5) independently.
7. **Message Expansion:** Merge the two sides (P_0, P_2) and (P_4, P_5) and satisfy the conditions on the message expansion of the remaining bytes.

4.1 First Inbound Phase

We start the attack on LANE-256 by applying the first inbound phase to each of the 4 active lanes P_0, P_2, P_4, P_5 independently. In each lane, we start with 5 active bytes in state #2 and 8 active bytes in state #7 and choose 2^{96} random non-zero differences for these 13 bytes (note that we could choose up to 2^{104} differences). We propagate backward and forward to 16 active bytes at the input (state #4) and output (state #5) of the SubBytes layer in between. We get at least 2^{96} solutions for the inbound phase with a complexity of 2^{96} (see Sect. 3.3). For each result, only the red and black bytes in Fig. 4 are determined, i.e. the differences as well as the actual values of the bytes are found. Note that we have chosen the position of active bytes in state #0, such that at least one term of Equation (2) or (4) is zero for each byte. At this point, we can compute backwards to state #0 and independently verify the condition on one byte of the input differences:

$$P_0 : \ \Delta a_0[0,0] = \Delta a_1[0,0]\,, \qquad P_4 : \ \Delta m_0[2,3] = \Delta m_1[2,3]$$
$$P_2 : \ \Delta c_0[2,3] = \Delta c_1[2,3]\,, \qquad P_5 : \ \Delta m_2[0,0] = \Delta m_3[0,0]$$

The condition on each of these bytes is fulfilled with a probability of 2^{-8} and we store the 2^{88} valid results of each lane P_0, P_2, P_4 and P_5 in the corresponding lists L_0, L_2, L_4 and L_5. Note that we store the values and differences of state #10 (red and black bytes) in these lists, since we need to merge these bytes with the second inbound phase in the following. For an efficient merging step, the lists are stored in hash tables (or sorted) according to the bytes to be merged (diffences and values of active bytes in state #10).

4.2 Second Inbound Phase

Next, we apply the inbound phase again to match the differences at SubBytes between state #12 and state #13. We start with 2^{64} differences in the 8 active bytes of state #10 and 2^{32} differences in the 4 active bytes of state #15. Hence, we get about 2^{96} solutions for the second inbound phase with a complexity of 2^{96}. For each result, the gray and black values in Fig. 4 between state #7 and state #18 are determined. Again, this means we fix the actual values of these bytes. The results of the second inbound phase for each lane are stored in lists L'_0, L'_2, L'_4 and L'_5. A node of each lists holds the values and differences of state #10 (gray and black bytes). Again, the lists are stored in hash tables (or sorted) according to the bytes (black bytes) to be merged.

4.3 Merge Inbound Phases

The two previous inbound phases overlap in 8 active bytes (state #7 to state #10). We connect the two inbound phases by checking the conditions on the overlapping bytes of state #10. Since both values and differences need to match, we get a condition on 128 bits. We merge the 2^{88} results of the first inbound

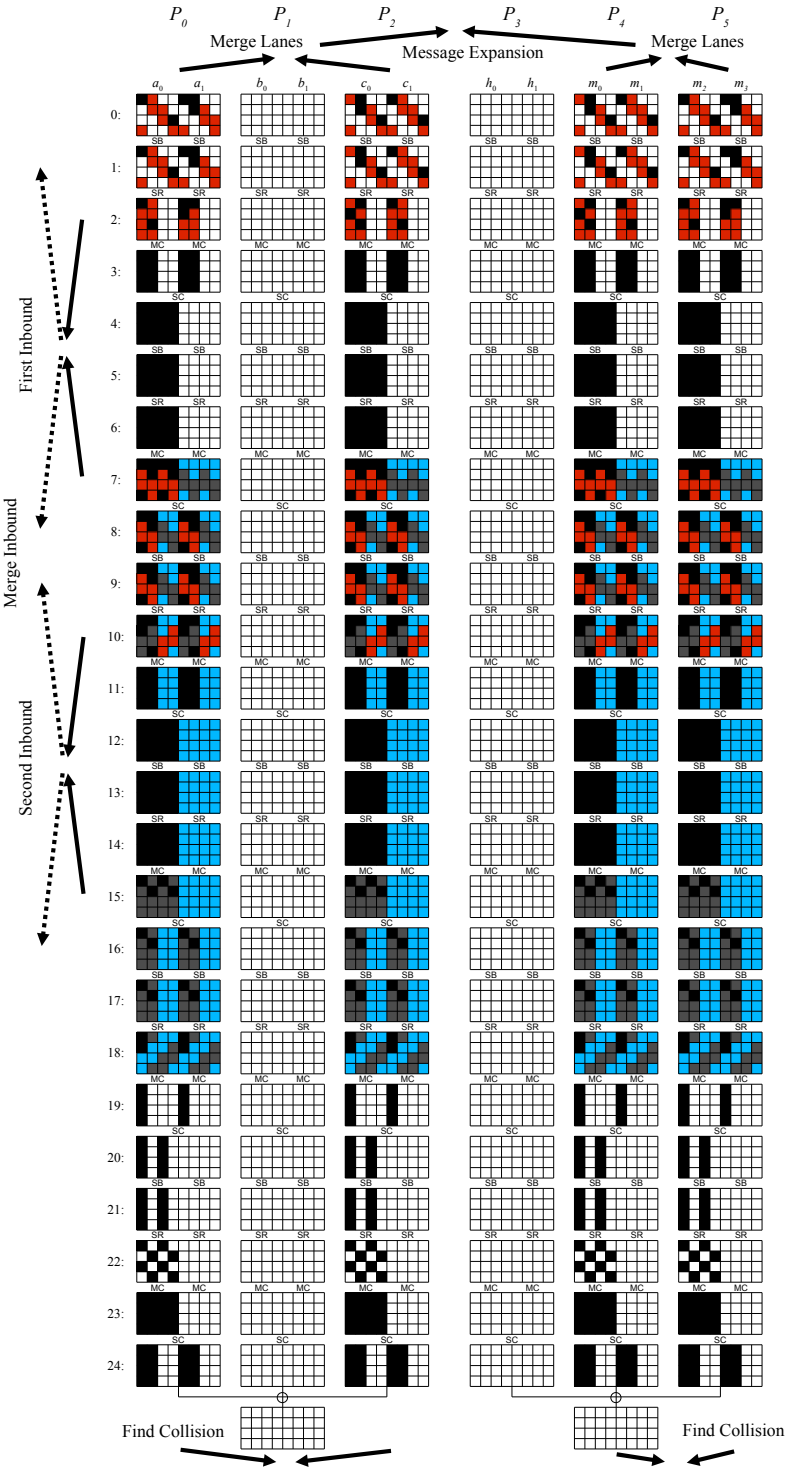

Fig. 4. The truncated differential path for 6 rounds of LANE-256. Black bytes are active, red (gray) bytes correspond to the first inbound phase, gray (dark gray) bytes to the second inbound phase and blue (light gray) bytes are used to find collisions in the P-lanes (colors in brackets correspond to grayscale printing).

phase and 2^{96} results of the second inbound phase to get $2^{88} \times 2^{96} \times 2^{-128} = 2^{56}$ differential paths for each lane. A pair connecting both inbound phases is found trivially. For each node of the first list (for example L_0), we check the overlapping bytes against the values of the second list (L_0'). Since the second list is a hash table, the effort for producing all 2^{56} valid pairs is 2^{88} hash table lookups.

Note that for each pair which satisfies and connects both inbound phases, the differences and values between state #0 and state #18 (black, red and gray bytes) are determined. We compute and store the 2^{56} input values and differences of state #0 in lists L_0, L_2, L_4 and L_5. Altough we still do not know half of the state, each of these input pairs conforms to the whole truncated differential path from state #0 to state #24 with a probability of 1. In other words, we know that in state #24, there are at most the given bytes active.

4.4 Merge Lanes

Next, we continue with merging the solutions of each lane by considering the message expansion. We first combine the inputs of lane P_0 and P_2 by merging lists L_0 and L_2. When merging these lists, we need to satisfy the conditions on the differences of the message expansion. We have conditions on 5 active bytes of state #0 in lane P_0 and P_2 (see Fig. 4). Remember that we have chosen the position of these active bytes, such that at least one term of Equation (2) or (4) is zero. Hence, we only need to check if *two* corresponding byte differences are equal. Since we have already verified one byte difference (see Sect. 4.1), we have 4 byte condition left:

$$\Delta a_0[0,0] = \Delta c_1[0,0]\,, \quad \Delta a_1[0,1] = \Delta c_0[0,1] \tag{6}$$

$$\Delta a_1[1,1] = \Delta c_0[1,1]\,, \quad \Delta a_0[2,3] = \Delta c_0[2,3] \tag{7}$$

These conditions are fulfilled with a probability of 2^{-32} and by merging two lists (L_0 and L_2) of size 2^{56}, we get $2^{56} \times 2^{56} \times 2^{-32} = 2^{80}$ valid matches which we store in list L_{02}. We repeat the same for lane P_4 and P_5 by merging lists L_4 and L_5. We get 2^{80} matches for list L_{45} as well, since we need to fulfill the 32-bit conditions on the differences of the following 4 bytes:

$$\Delta m_1[0,0] = \Delta m_2[0,0]\,, \quad \Delta m_0[0,1] = \Delta m_3[0,1] \tag{8}$$

$$\Delta m_0[1,1] = \Delta m_3[1,1]\,, \quad \Delta m_0[2,3] = \Delta m_2[2,3] \tag{9}$$

Again, if we use hash tables or the previous lists are sorted according to the bytes to match, the merge operation can be performed very efficiently. Hence, the total complexity to produce the lists L_{02} and L_{45} is determined by their final size and requires an effort of around 2^{80} computations.

4.5 Message Expansion

For all entries of the lists L_{02} and L_{45}, the values in 32 bytes and differences in 10 bytes of each of (a_0, a_1, c_0, c_1) and (m_0, m_1, m_2, m_3) have been fixed (red and

black bytes in state #0 of Fig. 4). Note that the conditions on the differences of each side on its own have already been fulfilled ($P_0 \leftrightarrow P_2$ and $P_4 \leftrightarrow P_5$). Hence, if we just fulfill the conditions on the remaining differences between $P_0 \leftrightarrow P_4$, then the conditions on $P_2 \leftrightarrow P_5$ are satisfied as well. Using Equations (2)-(4), the position of active bytes in Fig. 4 and the already matched differences of Sect. 4.1 and Sect. 4.4, we only have the following 4 byte conditions left:

$$\Delta a_0[0,0] = \Delta m_1[0,0]\,, \quad \Delta a_1[0,1] = \Delta m_0[0,1]$$
$$\Delta a_1[1,1] = \Delta m_0[1,1]\,, \quad \Delta a_0[2,3] = \Delta m_0[2,3]$$

Note that we also need to fulfill the conditions on the values of the states. Remember that we can freely choose the chaining values (h_0, h_1) to satisfy the values in the first 16 bytes of the message expansion (a_0, a_1). To fulfill the conditions on the 16 bytes of (c_0, c_1) we need to satisfy Equation (5) using the corresponding values v_1, v_2, v_3 and v_4. Hence, we need to find a match for the following values and differences by merging lists L_{02} and L_{45}:

- 8 bytes of v_1 from L_{02} with v_3 from L_{45},
- 8 bytes of v_2 from L_{02} with v_4 from L_{45},
- 4 bytes of differences in L_{02} and in L_{45}.

Since we have 2^{80} elements in each list and conditions on 160 bits, we expect to find $2^{80} \times 2^{80} \times 2^{-160} = 1$ result. This result satisfies the message expansion for all lanes and is a solution for the truncated differential path of each active lane between state #0 and state #24. However, we do not get a collision at the end of the P-lanes yet, since we do not know the differences of state #24.

4.6 Find Collisions

In this phase of the attack, we search for a collision at the end of the P-lanes (P_0, P_2) and (P_4, P_5) using the remaining freedom in the second half of the state. Note that the 16-byte difference in state #24 is obtained from 8-byte difference in state #22 with the linear transforms MixColumns and SwapColumns. Hence, the collision space (the 16 bytes where the two lanes differ) has only 2^{64} distinct elements. If we take a look at Fig. 4, we get for the values in state #7:

- The black, red and gray bytes represent values which have already been determined by the previous parts of the attack.
- The blue bytes represent values not yet determined and can be used to vary the differences in state #22.

To find a collision between two lanes, we can still choose 2^{64} values for the blue bytes in state #7 of each lane and store these results in lists L_0, L_2, L_4 and L_5. Note that for these 2^{64} values, we get only 2^{32} different values for the two free bytes in the first and fifth column of state #18. Hence, we can only iterate through 2^{32} differences in state #22 for each lane. However, this is enough to find one colliding difference for each side, since $2^{32} \times 2^{32} \times 2^{-64} = 1$. By repeating this step 2^{32} times for each side, we expect $2^{64} \times 2^{64} \times 2^{-64} = 2^{64}$ results for each merged list L_{02} and L_{45}.

4.7 Message Expansion

Finally, we need to match the message expansion for the remaining 32 bytes of each side. Hence, we just repeat the same procedure as we did for the first half of state #0, except that we only need to match the values of 32 bytes but no differences. Again, we can use the remaining bytes of (h_0, h_1) to fulfill the conditions on 16 bytes of (a_0, a_1). Since, we have 2^{64} solutions in each list L_{02} and L_{45}, we expect to find $2^{64} \times 2^{64} \times 2^{-128} = 1$ colliding pair for (c_0, c_1) and thus, a collision for the full compression function of LANE-256.

4.8 Complexity

Let us find the complexity of the whole attack. The first inbound phase requires 2^{96} computations and 2^{88} memory, the second inbound requires 2^{96} computations and 2^{96} memory, and the merging of the inbound phases requires 2^{88} hash table lookups and 2^{56} memory. Obviously, the second inbound phase and the merge inbound phases can be united to lower the memory requirement of these three steps. Namely, we create the lists L_0, L_2, L_4 and L_5 in the first inbound phase. Then, for each differential path of the second inbound phase, instead of storing it in a list, we immediately check if it can be merged with some differential from the lists. Only if it can be merged, we do the outbound phase and compute state #0. Hence, the first three steps of our attack require around 2^{96} computations and 2^{88} memory. The merge lanes step requires 2^{80} computations and memory. The message expansion steps require 2^{80} computations, while the find collisions steps require 2^{32} computations. Hence, the total attack complexity is around 2^{96} computations and 2^{88} memory. Note that the cost of each computation is never greater than the cost of one compression function evaluation. Therefore, the complexity to find a semi-free-start collision for all 6 rounds of LANE-256 is about 2^{96} compression function evaluations and 2^{88} memory.

5 Semi-Free-Start Collision for LANE-512

In the rebound attack on LANE-512, we construct a semi-free-start collision for the full, 8-round compression function using 2^{224} compression function evaluations and memory requirements of 2^{128}. We use the same iterative truncated differential path as shown in the specification of LANE-512 [Fig. 4.3, page 34], which is given in Fig. 5. Similar to the attack on LANE-256, we search for a collision after the P-lanes and use the same truncated differential path in the 4 active lanes P_0, P_2, P_4 and P_5. The attack on LANE-512 consists basically of the following parts:

1. **First Inbound Phase:** Apply the inbound phase at the beginning of the truncated differential path (state #2 to state #7) for each lane P_0, P_2, P_4, P_5 independently.
2. **Merge Lanes:** Merge the two neighboring lanes P_0,P_2 and P_4,P_5 and satisfy according differences of the message expansion.

3. **Message Expansion:** Merge the two sides (P_0, P_2) and (P_4, P_5) and satisfy the remaining conditions on the message expansion (differences and values).
4. **Second Inbound Phase:** Apply the inbound phase in the middle of each lane again (state #10 to state #15).
5. **Merge Inbound Phases:** Merge the results of the two inbound phases.
6. **Starting Points:** Choose random values for the brown bytes in state #7 to get enough starting points for the subsequent phases.
7. **Merge Lanes:** Merge the values of the starting points for the two neighboring lanes P_0,P_2 and P_4,P_5 and satisfy the according differences of the message expansion.
8. **Message Expansion:** Merge the two sides (P_0, P_2) and (P_4, P_5) and satisfy the remaining conditions on the message expansion (differences and values) for the starting points.
9. **Third Inbound Phase:** Apply the inbound phase at the end of each lane for a third time (state #18 to state #23).
10. **Merge Inbound Phases:** Merge the results of the three inbound phases and use the remaining freedom in between.
11. **Find Collisions:** Merge the corresponding two lanes to find a collision for each side (P_0, P_2) and (P_4, P_5) independently.
12. **Message Expansion:** Merge the two sides (P_0, P_2) and (P_4, P_5) and satisfy the conditions on the message expansion of the remaining bytes.

5.1 First Inbound Phase

We start the attack on Lane-512 by applying the first inbound phase to each of the 4 active lanes P_0, P_2, P_4, P_5 independently. In each lane, we start with 8 active bytes in state #2 and 4 active bytes in state #7 and choose 2^{84} random non-zero differences for these 12 bytes (note that we could choose up to 2^{96} differences). We propagate backward and forward to 16 active bytes at the input (state #4) and output (state #5) of the SubBytes layer in between. We get at least 2^{84} matches for the inbound phase with a complexity of 2^{84} (see Sect. 3.3). For each result, the gray and black bytes in Fig. 5 are determined. Hence, we can already verify the condition on one byte of the input differences for each lane by computing backwards to state #0:

$$
\begin{aligned}
P_0 &: \Delta a_0[2,2] = \Delta a_1[2,2]\,, & P_0 &: \Delta a_0[2,6] = \Delta a_1[2,6] \\
P_2 &: \Delta c_0[1,1] = \Delta c_1[1,1]\,, & P_2 &: \Delta c_0[1,5] = \Delta c_1[1,5] \\
P_4 &: \Delta m_0[1,1] = \Delta m_1[1,1]\,, & P_4 &: \Delta m_0[1,5] = \Delta m_1[1,5] \\
P_5 &: \Delta m_2[2,2] = \Delta m_3[2,2]\,, & P_5 &: \Delta m_2[2,6] = \Delta m_3[2,6]
\end{aligned}
$$

The conditions on each of the lanes are fulfilled with a probability of 2^{-16} and we store the 2^{68} valid matches of each lane P_0, P_2, P_4 and P_5 in the corresponding lists L_0, L_2, L_4 and L_5.

5.2 Merge Lanes

Next, we continue with merging the solutions of each lane by considering the message expansion. We first combine the results of lane P_0 and P_2 by merging lists L_0 and L_2. When merging these lists, we need to satisfy the conditions on the differences of the message expansion for the following 6 bytes:

$$\begin{aligned} \Delta a_1[0,0] &= \Delta c_0[0,0], \quad \Delta a_1[0,4] = \Delta c_0[0,4] \\ \Delta a_0[1,1] &= \Delta c_0[1,1], \quad \Delta a_0[1,5] = \Delta c_0[1,5] \\ \Delta a_0[2,2] &= \Delta c_1[2,2], \quad \Delta a_0[2,6] = \Delta c_1[2,6] \end{aligned}$$

Since this match is fulfilled with a probability of 2^{-48} and we merge two lists of size 2^{68}, we get $2^{68} \times 2^{68} \times 2^{-48} = 2^{88}$ valid matches which we store in L_{02}. We repeat the same for lane P_4 and P_5 merge lists L_4 and L_5. We get 2^{88} matches for list L_{45}, since we need to fulfill conditions on differences of 6 bytes as well:

$$\begin{aligned} \Delta m_0[0,0] &= \Delta m_3[0,0], \quad \Delta m_0[0,4] = \Delta m_3[0,4] \\ \Delta m_0[1,1] &= \Delta m_2[1,1], \quad \Delta m_0[1,5] = \Delta m_2[1,5] \\ \Delta m_1[2,2] &= \Delta m_2[2,2], \quad \Delta m_1[2,6] = \Delta m_2[2,6] \end{aligned}$$

5.3 Message Expansion

For all entries of lists L_{02} and L_{45}, the values in 32 bytes and differences in 16 bytes of each of (a_0, a_1, c_0, c_1) and (m_0, m_1, m_2, m_3) have been fixed (gray and black bytes in state #0 of Fig. 5). Since the conditions on the differences of each side on its own have already been fulfilled, we just need to match the conditions on the remaining 6-byte differences between each side (P_0, P_2) and (P_4, P_5):

$$\begin{aligned} \Delta a_1[0,0] &= \Delta m_0[0,0], \quad \Delta a_1[0,4] = \Delta m_0[0,4] \\ \Delta a_0[1,1] &= \Delta m_0[1,1], \quad \Delta a_0[1,5] = \Delta m_0[1,5] \\ \Delta a_0[2,2] &= \Delta m_1[2,2], \quad \Delta a_0[2,6] = \Delta m_1[2,6] \end{aligned}$$

Remember that we can freely choose the chaining values (h_0, h_1) to satisfy the values in the first 16 bytes of the message expansion (a_0, a_1). To fulfill the conditions on the 16 bytes of (c_0, c_1) we need to find matches for the following values and differences using lists L_{02} and L_{45}:

- 8 bytes of v_1 from L_{02} with v_3 from L_{45},
- 8 bytes of v_2 from L_{02} with v_4 from L_{45},
- 6 bytes of differences in L_{02} and in L_{45}.

Since we have 2^{88} elements in each list and conditions on 176 bits, we expect to find $2^{88} \times 2^{88} \times 2^{-176} = 1$ result. This result satisfies the message expansion for all lanes and is a solution for the truncated differential path of each active lane between state #0 and state #10.

5.4 Second Inbound Phase

Next, we apply the inbound phase again to match the differences at SubBytes between state #12 and state #13. After the first inbound phase, the values of 16 bytes in state #10 (black and gray bytes), and the difference in 16 bytes (1st AES-block) of state #12 (black bytes) have already been fixed. Hence we can start with 2^{32} possible 4-byte differences in state #15, compute backwards to state #13 and need to match the differences in the SubBytes layer. We expect to find at least 2^{32} solutions for the second inbound phase (see Sect. 3.3).

5.5 Merge Inbound Phases

The result of the second inbound phase are 2^{32} values for the 16 bytes in state #10 (green and black bytes). From the first inbound phase, we have obtained one solution for 16 bytes in state #10 (gray and black bytes) as well. In these 16 bytes, the values of the 4 active bytes (black) overlap between both inbound phases and the probability for a successful match is 2^{-32}. Among the 2^{32} results of the second inbound phase, we expect to find one solution to match the values of state #10. Once we have found a match, we can compute the values of the newly determined 12 bytes in state #7, marked by green bytes in Fig. 5.

5.6 Starting Points

In this phase of the attack, we will compute a number of starting points which we will need for the subsequent steps. For each lane, we choose random values for the 12 bytes in state #7 (marked by brown bytes in Fig. 5) and compute the corresponding 16-byte values in state #0. We repeat this step 2^{64} times and store the results in the corresponding lists L'_0, L'_2, L'_4 or L'_5.

5.7 Merge Lanes

Next, we merge lists L'_0 and L'_2 to get the list L'_{02}, consisting of 2^{128} values for the 32 newly determined bytes of (m_0, m_1, m_2, m_3) (brown bytes of state #0 in lane P_0 and P_2). Further, we merge lists L'_4 and L'_5 to get the list L'_{45} of size 2^{128} containing the 32 byte values of (a_0, a_1, c_0, c_1).

5.8 Message Expansion

Finally, we satisfy the conditions of the message expansion on (a_0, a_1) using the values of (h_0, h_1), and use the two lists L'_{02} and L'_{45} to satisfy the conditions on (c_0, c_1). Since we need to match 16 bytes of (c_0, c_1) and have 2^{128} elements in both lists, we expect $2^{128} \times 2^{128} \times 2^{-128} = 2^{128}$ matching pairs which we store in list L_s. We will use these values in a later phase of the attack.

5.9 Third Inbound Phase

Now, we extend the truncated differential path by applying a third inbound phase between state #18 and state #23 for each active lane. Note that the values in 16 bytes of state #18 (black and green bytes), and the differences in 16 bytes (1st AES-block) of state #20 (black bytes) have already been fixed due to the second inbound phase. Similar to the second inbound phase, we start with 2^{32} 4-byte differences in state #23 and compute backwards to state #21 to get a match for the SubBytes layer. Since we have 2^{32} starting differences, we expect to find 2^{32} results for the third inbound phase, with fixed values and differences for the 16 bytes in state #15 (purple and black bytes).

5.10 Merge Inbound Phases

The values of the second and the third inbound phase overlap in 4 active bytes (black) of state #18. Since we have 2^{32} results of the third inbound phase, we expect to find one solution after merging the two phases. Once we have found a match, we can compute the values of the newly determined 12 bytes in state #15, marked by purple bytes in Fig. 5. Next, we need to connect all three inbound phases. For all possible 8-byte values of state #10 marked by red bytes, we compute the 16 corresponding bytes in state #15 (2nd AES-block). If the computed values satisfy the 4 bytes in state #15 marked by purple, we store the result of each lane in the corresponding lists L_0^a, L_2^a, L_4^a and L_5^a. In total, we obtain $2^{64} \cdot 2^{-32} = 2^{32}$ entries in each list. We repeat the same for the bytes marked by blue and yellow, and generate the lists L_i^b and L_i^c for each of the active lanes with index $i \in \{0, 2, 4, 5\}$. For each lane, we merge the three lists L_i^a, L_i^b and L_i^c and store the 2^{96} results in lists L_i^*. Note that for each entry in these lists, we can determine all values and differences of the corresponding lane.

5.11 Find Collisions

In this phase of the attack, we finally search for a collision at the end of the P-lanes (P_0, P_2) and (P_4, P_5) using the elements of lists L_i^*. To find a collision at the end of the P-lanes, we need to match the 16 byte differences in state #32 of the two corresponding active lanes such that $\Delta(P_0 \oplus P_2) = 0$ and $\Delta(P_4 \oplus P_5) = 0$. Note that we can satisfy these conditions independently for each side (P_0, P_2) and (P_4, P_5). Since we need to match 128 bits and we have 2^{96} elements in each list L_i^*, we expect to find $2^{96} \cdot 2^{96} \cdot 2^{-128} = 2^{64}$ collisions for each side. We store the corresponding inputs (a_0, a_1, c_0, c_1) for the collisions between lane P_0 and P_2 in list L_{02}^* and the inputs (m_0, m_1, m_2, m_3) for the collisions between lane P_4 and P_5 in list L_{45}^*.

5.12 Message Expansion

Finally, we need to match the message expansion for the remaining 32 bytes of each side. Hence, we just repeat the same procedure as we did for the first

part of state #0, except that we only need to match the values of 32 bytes but no differences. Again, we use the values of (h_0, h_1) to satisfy the conditions on (a_0, a_1) first. Then, we match the values of the 32 bytes in (c_0, c_1). Since we only have 2^{64} entries in both of L^*_{02} and L^*_{45}, the success probability for a match is $2^{64} \cdot 2^{64} \cdot 2^{-256} = 2^{-128}$. However, we can still repeat from Sect. 5.6 using a different starting point stored in list L_s. Since we have 2^{128} elements in list L_s, we can repeat the previous steps up to 2^{128} times. Hence, we expect to find one valid match for the message expansion and thus, a collision for the full compression function of Lane-512.

5.13 Complexity

The total complexity of the rebound attack on Lane-512 is determined by the merging step after the third inbound phase. This step has a complexity of 2^{96} compression function evaluations and is repeated 2^{128} times. The memory requirements are determined by the largest lists, which are L'_{02} and L'_{45} (or L_s) with a size of 2^{128}. Hence, the total complexity to find a semi-free-start collision for Lane-512 is about $2^{128} \cdot 2^{96} = 2^{224}$ compression function evaluations and 2^{128} in memory.

6 Conclusion

In this work, we have applied the rebound attack to the hash function Lane. In the attack we use a truncated differential path with differences concentrating mostly in one part of the lanes. Due to the relatively slow diffusion of parallel AES rounds, we are therefore able to solve parts of the lanes independently. First, we search for differences and values (for parts of the state) according to the truncated differential path and also satisfy the message expansion. Then, we choose values which can be changed such that the truncated differential path and according message expansion still holds. The freedom in these values is then used to search for a collision at the end of the lanes without violating the differential path or message expansion.

In the rebound attack on Lane, we are able to construct semi-free-start collisions for full round Lane-224 and Lane-256 with 2^{96} compression function evaluations and memory of 2^{80}, and for full round Lane-512 with complexity of 2^{224} compression function evaluations and memory of 2^{128}. Although these collisions on the compression function do not imply an attack on the hash functions, they violate the reduction proofs of Merkle and Damgård, or Andreeva in the case of Lane. However, due to the limited degrees of freedom, a collision attack on the hash function seems to be difficult for full round Lane.

References

1. Andreeva, E.: On LANE modes of Operation. Technical Report (2008), COSIC
2. Barreto, P.S.L.M., Rijmen, V.: The WHIRLPOOL Hashing Function. Submitted to NESSIE, September 2000, revised May 2003. Available online at `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`
3. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 416–427. Springer (1989)
4. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Available online at `http://www.groestl.info` (2008)
5. Indesteege, S.: The LANE hash function. Submission to NIST (2008), Available online at: `http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf`
6. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE. LNCS, vol. 1008, pp. 196–211. Springer (1994)
7. Knudsen, L.R., Rechberger, C., Thomsen, S.S.: The Grindahl Hash Functions. In: Biryukov, A. (ed.) FSE. LNCS, vol. 4593, pp. 39–57. Springer (2007)
8. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT. LNCS, Springer (2009), to appear.
9. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE. LNCS, vol. 5665, pp. 260–276. Springer (2009)
10. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 428–446. Springer (1989)
11. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register Notice (November 2007), `http://csrc.nist.gov`
12. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT. LNCS, vol. 4833, pp. 551–567. Springer (2007)
13. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO. LNCS, vol. 3621, pp. 17–36. Springer (2005)
14. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 19–35. Springer (2005)
15. Wu, S., Feng, D., Wu, W.: Cryptanalysis of the LANE Hash Function. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. LNCS, Springer (2009), to appear.

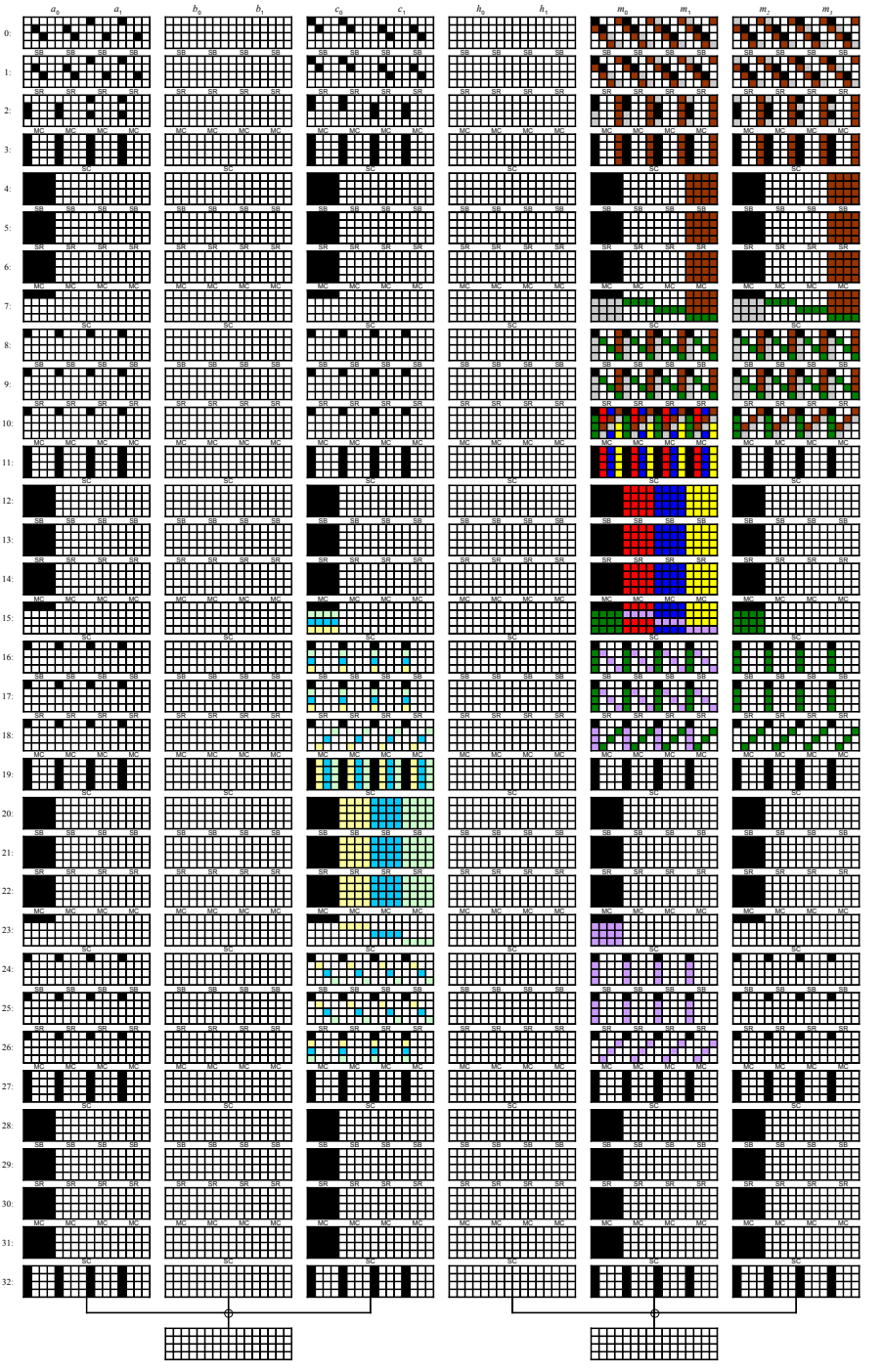

Fig. 5. The truncated differential path for 8 rounds of LANE-512. Lane P_0 shows the plain truncated differential path, lane P_2 other possible truncated differential paths and lane P_4 and P_5 are used to describe the attack.