Cryptanalysis of *Luffa* v2 Components*

Dmitry Khovratovich[1], María Naya-Plasencia[2], Andrea Röck[3], and Martin Schläffer[4]

[1] University of Luxembourg, Luxembourg
[2] FHNW, Windisch, Switzerland
[3] Aalto University School of Science and Technology, Finland
[4] IAIK, Graz University of Technology, Austria

Abstract. We develop a number of techniques for the cryptanalysis of the SHA-3 candidate *Luffa*, and apply them to various *Luffa* components. These techniques include a new variant of the rebound approach taking into account the specifics of *Luffa*. The main improvements include the construction of good truncated differential paths, the search for differences using multiple inbound phases and a fast final solution search via linear systems. Using these techniques, we are able to construct non-trivial semi-free-start collisions for 7 (out of 8 rounds) of *Luffa*-256 with a complexity of 2^{104} in time and 2^{102} in memory. This is the first analysis of a *Luffa* component other that the permutation of *Luffa* v1. Additionally, we provide new and more efficient distinguishers also for the full permutation of *Luffa* v2. For this permutation distinguisher, we use a new model which applies first a short test on all samples and then a longer test on a smaller subset of the inputs. We demonstrate that a set of right pairs for the given differential path can be found significantly faster than for a random permutation.
Keywords: hash functions, SHA-3 competition, Luffa, cryptanalysis, rebound attack, semi-free-start collision, distinguisher

1 Introduction

The hash function *Luffa* [5] is a Round 2 candidate of the NIST SHA-3 competition, and follows the wide-pipe design using a sponge-based mode of operation. *Luffa* shows its originality in the design of the compression function, which is based on the parallel call of 3, 4, or 5 permutations (depending on the output size). A similar design approach was used in the hash function LANE [8], but with different input and output transformations.

* The work described in this paper has been supported in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, by the French Agence Nationale de la Recherche under Contract ANR-06-SETI-013-RAPIDE, by the PRP "Security & Trust" grant of the University of Luxembourg, by the Academy of Finland under project 122736, and during the tenure of an ERCIM Alain Bensoussan Fellowship Programme.

In this paper we present several results on various components of *Luffa*. First, we analyze the *Luffa* mode of operation and derive sufficient conditions for a differential path with low-weight input and output differences in the permutations. Then we proceed with the analysis of the *Luffa* permutations and construct a 7-round truncated differential path with using a meet-in-the-middle approach. We are able to exploit the rotational symmetry of *Luffa* to increase the number of differential paths and construct solutions for these paths with an advanced rebound attack [10].

Since the number of active S-boxes in the differential path is too large for a straightforward application of the rebound attack, we need to solve this issue with several refinements of the attack. We use *multiple inbound phases* and a *parallel matching technique* to find all possible differential paths for the truncated path first. We need a parallel matching technique to match large lists of differences through the S-box layer in the inbound phase. A gradual matching as in the attacks on AES is not possible for the not block-wise operating linear transformation of *Luffa*.

Using this advanced rebound attack we get a semi-free-start collision for 7 (out of 8) rounds of *Luffa*-256 v2 in Section 3, which can be extended to an 8-round semi-free-start distinguisher (see Section 4). We have also defined a new type of distinguisher which can be used to distinguish the full permutation of *Luffa* v2 in Section 5.

Note that the supporting document of *Luffa* provides a semi-free-start collision for *Luffa*-512 with complexity 2^{204} due to the properties of the message injection and using the generalized birthday problem [11]. For *Luffa*-256 this attack does not have a complexity lower than the birthday bound. Other previous results include the existence of a non-trivial differential path for 8 rounds of the internal permutation with probability 2^{-224}, and different variations of high-order differential distinguishers [1,4].

Table 1. Results. Note that the meaning and setting of "distinguisher" varies depending on the attack.

building block	security parameter	time	memory	technique
Distinguishers				
permutation v1	full (8)	2^{224}	-	differential [7]
compression v1	6	2^{84}	-	higher order diff. [4]
compression v1	7	2^{216}	-	higher order diff. [4]
permutation v1	full (8)	2^{82}	-	algebraic zero-sum [1]
permutation v2	full (8)	2^{116}	-	two-tier differential Sect. 5
compression *Luffa*-256 v2	full (8)	2^{104}	2^{102}	advanced rebound attack Sect. 4
Semi-free-start collision				
compression *Luffa*-256 v2	7	2^{104}	2^{102}	advanced rebound attack Sect. 3

2 Description of *Luffa*

In this section we briefly describe the SHA-3 candidate *Luffa* [5]. For a more detailed description we refer to the submission document.

2.1 The Iteration

The hash function *Luffa* is a variant of a sponge function and consists of a linear message injection MI, w 256-bit permutations Q_j and a finalization function C''. The chaining value at instant i is represented by $(H_0^{(i)}, \ldots, H_{w-1}^{(i)})$. The size of each message block $M^{(i)}$, each value $H_j^{(i)}$ and starting variable V_j is 256 bits. In the iteration of the hash function *Luffa*, a padded t-block message M is hashed as follows:

$$\begin{aligned}
(H_0^{(0)}, \ldots, H_{w-1}^{(0)}) &= (V_0, \ldots, V_{w-1}) \\
(X_0, \ldots, X_{w-1}) &= MI(H_0^{(i-1)}, \ldots, H_{w-1}^{(i-1)}, M^{(i)}) \quad \text{for } 1 \le i \le t \\
(H_0^{(i)}, \ldots, H_{w-1}^{(i)}) &= (Q_0(X_0), \ldots, Q_{w-1}(X_{w-1})) \quad \text{for } 1 \le i \le t \\
hash &= C''(H_0^{(t)}, \ldots, H_{w-1}^{(t)}).
\end{aligned}$$

The parameter w depends on the hash output size and is specified to be $w = 3$ for *Luffa*-224 and *Luffa*-256, $w = 4$ for *Luffa*-384, and $w = 5$ for *Luffa*-512. Fig. 1 shows the iteration of the hash function *Luffa*-256 with $w = 3$. In the following, we describe the permutations Q_j and the message injection MI of *Luffa* in more detail.

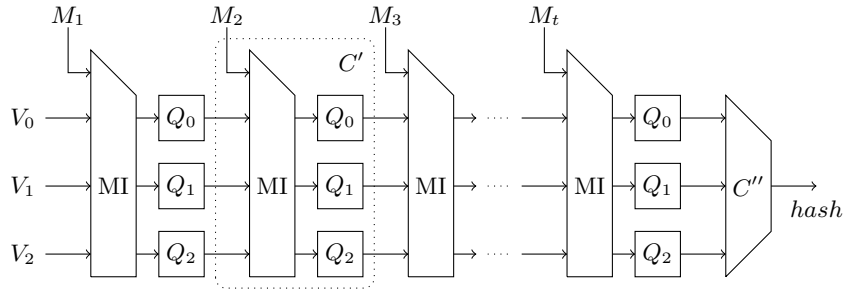

Fig. 1. The iteration of the hash function *Luffa*-256 ($w = 3$) with message injection MI, permutations Q_j and finalization function C''.

2.2 The Permutations

The non-linear 256-bit permutations Q_j update a state of 8 32-bit words $a_0, a_1, \ldots, a_7$. Initially, an InputTweak is applied to the input of the permutations.

Furthermore, each permutation consists of 3 round transformations SubCrumb, MixWord, and AddConstant which are repeated for 8 rounds. The permutations Q_j differ only in the InputTweak and AddConstant transformation. In the following, we give a detailed description of the round transformations. We organize the state in 4×2 32-bit words with the LSB of each word at the right hand side (see Fig. 2). Furthermore, we call the 4 words a_0, a_1, a_2, a_3 left words (or left side) and the 4 words a_4, a_5, a_6, a_7 right words (or right side), and we call each 4-bit column of this state a nibble. In the following, we describe a specific difference in a nibble either by its bit pattern, e.g. 1011 (with the LSB on the right), or by its hexadecimal value, e.g. $0xB$, depending on which is more convenient for the understanding.

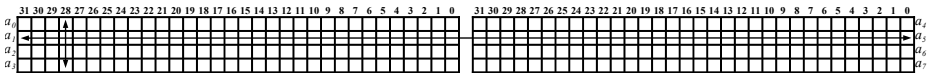

Fig. 2. The 256-bit state of each *Luffa* permutation Q_i is organized in 8 32-bit words. In this representation, SubCrumb is applied vertically to 4-bit columns (nibbles) and MixWord horizontally to 64-bit rows of the state.

In permutation Q_j, the InputTweak rotates the 4 right words a_4, a_5, a_6, a_7 to the left by j positions. This tweak has no influence on the four left words. In the non-linear SubCrumb layer, the same 4-bit S-box is applied to each nibble of the state. Hence, 64 independent S-boxes are applied to the columns of the state (see Fig. 2). Note that the wiring is different for the left and right side, which is equivalent to applying two different S-boxes S and S' on each side.

In MixWord, a linear mixing function is applied to two 32-bit words (a_k and a_{k+4} for $k = 0, \ldots, 3$) of the state. Hence, 4 independent linear functions are applied to each row of the state (see Fig. 2). We give here an alternative description of MixWord which is more suitable for our analysis. A detailed description of MixWord can be found in the specification of *Luffa* [6]. We denote by a_k^i and a_{k+4}^i the i-th bit of the words a_k and a_{k+4} with $k = 0, \ldots, 3$. Then, the output words b_k^i and b_{k+4}^i are computed as follows:

$$\begin{aligned} b_k^i &= a_k^i \oplus a_k^{i+18} \oplus a_k^{i+20} \oplus a_k^{i+22} \oplus a_k^{i+30} \oplus a_{k+4}^i \oplus a_{k+4}^{i+18} \oplus a_{k+4}^{i+22} \\ b_{k+4}^i &= a_k^{i+17} \oplus a_k^{i+29} \oplus a_k^{i+31} \oplus a_{k+4}^{i+17} \oplus a_{k+4}^{i+31}, \end{aligned}$$

with $0 \leq i \leq 31$, $k = 0, \ldots, 3$ and all indices modulo 32.

Note that each bit of the left output words b_k^i depends on 8 bits of the input and each bit of the right output words b_{k+4}^i depends on 5 bits of the input. For the backward direction the opposite holds: Each bit of the left input words a_k^i depends on 5 bits of the output and each bit of the right input words a_{k+4}^i depends on 8 bits of the output. In addition, we show in Fig. 3 the propagation of a 1-bit difference in forward and backward direction. Let us consider for example the case $k = 0$ in the figure. We have a difference in all the bits of b_0 and b_4

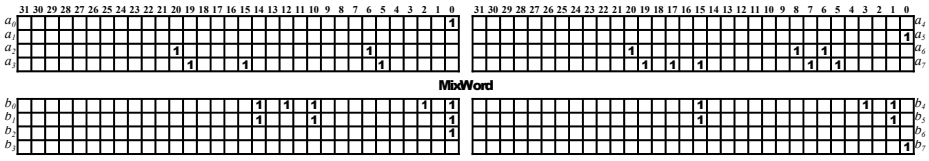

Fig. 3. Propagation of a single bit at the LSB of the left ($k = 0$) and right ($k = 1$) word in forward, and left ($k = 2$) and right ($k = 3$) word in backward direction. Empty bits are zero. Note that the 64-bit rows are independent in each MixWord transformation.

which depend on a_0^0, e.g.

$$b_0^0 = \underline{a_0^0} \oplus a_0^{18} \oplus a_0^{20} \oplus a_0^{22} \oplus a_0^{30} \oplus a_4^i \oplus a_4^{18} \oplus a_4^{22}.$$

2.3 The Message Injection

The message injection in *Luffa*-256 can be described by an operation in a ring of polynomials:

$$R = Z_2[x]/(x^8 + x^4 + x^3 + x + 1),$$

which is applied to those 8 bits of the state with equal bit position i in the 32-bit words a_k^i for $k = 0, \ldots, 7$. Hence, each bit of the message is mixed into two nibbles of the state. In the message injection, the chaining values H_j and the message block M are used to get the new intermediate chaining values X_i as follows:

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} x+1 & x & x & 1 \\ x & x+1 & x & x \\ x & x & x+1 & x^2 \end{pmatrix} \cdot \begin{pmatrix} H_0 \\ H_1 \\ H_2 \\ M \end{pmatrix}.$$

3 Semi-free-start Collision on *Luffa*-256 for 7 Rounds

In this section, we present a rebound technique to search for semi-free-start collisions in *Luffa*. With semi-free-start collision, we denote a collision on the internal state with no differences in the chaining value. Contrary to free-start collisions (with arbitrary differences in the chaining value), semi-free-start collisions are not trivial to find in sponge-like constructions. In our attack, we can find two distinct two-block messages M_1, M_2 and M_1^*, M_2^* such that

$$MI(M_2, P(MI(M_1, CV))) = MI(M_2^*, P(MI(M_1^*, CV))).$$

for some chaining value CV. In the following, we show that the complexity to find such a semi-free-start collision for 7 (out of 8) rounds of *Luffa*-256 is about 2^{104} in time and 2^{102} in memory.

3.1 Outline of the Attack

To search for semi-free-start collisions in reduced *Luffa*-256 we use an adapted and refined rebound attack. Since we do not allow differences in the chaining values, all 3 permutations need to be active. In the attack, we first search for truncated differential paths in Q_j which result in a semi-free-start collision for 7 rounds. Contrary to AES based designs, this step is already a non-trivial task. In the truncated differential path, we only consider active and non-active S-boxes (or nibbles) of the state. Hence, we will represent the truncated differential path using a line of 2x32 nibbles. We construct a path which has only one active S-boxes at the input and the end of the path. Furthermore, we also try to keep the number of active S-boxes in the middle rounds low. We use an improved rebound attack where we first filter for all possible differential paths and then solve for the values of the state to get corresponding input pairs for all three permutations. While filtering for differential paths, we also need to ensure that the input and output differences can be injected and erased by the message difference.

3.2 Matching the Message Injection

In the message injection, a difference in nibble i of M might affect two nibbles of the input of each permutations Q_j. Due to the InputTweak, these two nibbles are, at the left side the nibble at position i, and on the right side the nibble at position $i + j$. To get a sparse truncated differential path we aim for only one active S-box (one active nibble) in the first and the last round.

In the first message injection, the difference in all chaining values H_j is zero. Therefore, we get for the intermediate chaining variables X_j:

$$\begin{pmatrix} \Delta X_0 \\ \Delta X_1 \\ \Delta X_2 \end{pmatrix} = \begin{pmatrix} x+1 & x & x & 1 \\ x & x+1 & x & x \\ x & x & x+1 & x^2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ \Delta M \end{pmatrix} = \begin{pmatrix} \Delta M \\ x \cdot \Delta M \\ x^2 \cdot \Delta M \end{pmatrix}$$

In order to have only one active S-box at the input of each permutation, we require that $\Delta M, x \cdot \Delta M, x^2 \cdot \Delta M$ are polynomials either of degree < 4 or divisible by x^4. The most simple conditions which do not spread to the other nibble are

$$\Delta M = ax + b \text{ or } \Delta M = ax^5 + bx^4.$$

The first solution corresponds to a difference in a nibble on the left side and the second solution to a difference on the right side. The possible differences in these nibbles are 0001, 0010 or 0011.

In the second message injection, we need to erase all differences of the internal state. Hence, we get the following system of equations:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} x+1 & x & x & 1 \\ x & x+1 & x & x \\ x & x & x+1 & x^2 \end{pmatrix} \cdot \begin{pmatrix} \Delta H_0 \\ \Delta H_1 \\ \Delta H_2 \\ \Delta M \end{pmatrix}$$

Again, we consider only differences with only one active S-box, prior to the last (linear) MixWord transformation. This condition filters out most possible differences (we omit long calculations), and the only non-trivial solutions we have found are

$$\begin{pmatrix} \Delta H_0 \\ \Delta H_1 \\ \Delta H_2 \\ \Delta M \end{pmatrix} = \left\{ \begin{pmatrix} x^3+x^2+1 \\ x^3 \\ x \\ x+1 \end{pmatrix}, \begin{pmatrix} x^7+x^6+x^4 \\ x^7 \\ x^5 \\ x^5+x^4 \end{pmatrix} \right\}.$$

The first solution corresponds to a difference in the left side and the second one to a difference in the right side. Thus, for each permutation we have one specific output difference for the single active S-box in the last round, for which we are sure that we can erase it by a difference in the message. Note that the output difference of the permutation is the same in each active nibble. The specific differences in the nibbles of H_0, H_1, H_2, and M are 1101, 1000, 0010, and 0011, respectively.

To summarize, we search for differential paths with only a single active S-box in the first and the last round. For an active S-box in a right nibble at position i of Q_0, only the two least significant bits of the input difference should be active. The possible differences are then 0001, 0010 or 0011. For permutation Q_j, the active S-box will be at position $i + j$ modulo 32 on the right side, due to the InputTweak and should have the same difference as in Q_0 but rotated j positions to the MSB. We get the differences 0010, 0100 or 0110 for Q_1, and 0100, 1000 or 1100 for Q_2. Note that we fix the output difference of the active S-box to only one possible difference depending on the permutation. The difference at the output of the single active S-box in round 7 has to be 1101 for Q_0, 1000 for Q_1 and 0010 for Q_2. Note that for these input and output differences of the three permutations, the differences of the injected message block have been computed deterministically.

3.3 Constructing Truncated Differential Paths

For the semi-free-start collision on 7 rounds of *Luffa*-256, we use many truncated differential paths for each permutation to get enough solutions for an attack. All paths have the same numbers of active S-boxes in each round which are given as follows:

$$1 - 5 - 27 - 52 - 26 - 5 - 1$$

We have one active S-box after the SubCrumb of round 7, and we have already determined which difference we must have in it's output for each lane. This one active nibble leads to differences in 8 nibbles after the MixWord. The differences will be the same in each nibble and can be eliminated by the method described in Section 3.2. In the following analysis we will omit this last step, since it has no influence on the differential path.

To get a truncated differential path with only 5 active S-boxes in the second and second last round, the single active S-box at the input has to be in the right side and the single active S-box at the output in the left side. Note that the position of the single active S-box in round 7 is identical in all permutations. However, the position of the active S-box in round 1 is rotated by j positions for permutation Q_j due to the InputTweak. However, for each permutation, we are able to find an equivalent truncated differential path with the same number of active S-boxes (see Appendix A). Therefore, in the following we only consider the path of the first permutation Q_0.

We have constructed the truncated differential path for Q_0 by first propagating forward and backward without constraints from a single active S-box in round 1 and round 7 (see Fig. 4). For each single active S-box, we have tried all 32 positions to get a good (sparse) truncated differential path for our attack. We have found a path with the following number of active nibbles after each SubCrumb and MixWord transformation in forward direction:

$$1 - \mathsf{SC} - 1 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 27 - \mathsf{SC} - 27 - \mathsf{MW} - 58$$

and in backward direction:

$$57 - \mathsf{MW} - 26 - \mathsf{SC} - 26 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 1 - \mathsf{SC} - 1$$

Since the 57 and 58 active nibbles have only 52 common active nibbles, we simply impose some constraints on the connecting MixWord transformation. This results in a $(58 - 52) \times 4 = 24$ bit condition for the forward part, and a $(57 - 52) \times 4 = 20$ bit condition for the backward part. These conditions are marked by red nibbles in Fig. 4 and are required to be zero. In this truncated differential path, the active S-box at the input is located in the right word in bit 0, and at the output in the left word in bit 14. Furthermore, in the *Luffa* permutations all XOR differential paths are rotation symmetric within 32-bit words. Hence, we actually get 32 truncated differential paths with active input the ith nibble of the right part and active output the nibble $i + 14$ modulo 32 of the left part for Q_0. In general, for Q_j we get for an active input at nibble $i + j$ of the right part, the same active output at nibble $i + 14$ of the left part (see Appendix A) due to the InputTweak.

3.4 Rebound Attack on *Luffa*

To find pairs conforming to the given truncated differential paths, we use the rebound attack [10]. Due to the different structure of *Luffa* compared to AES based hash functions, several improvements are needed. Since the truncated differential path consists of three rather active states in the middle a standard inbound phase cannot be used. Previous rebound attacks dealt with relatively short inbound phases to exploit the propagation of differences with probability one. Since this is not possible for longer inbound phases, a different strategy is needed to reduce the overall computational complexity of the attack.

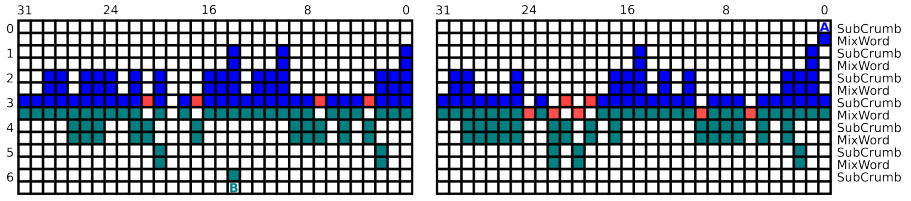

Fig. 4. An example for a truncated differential path followed in the first permutation for building the semi-free-start collisions. The difference in **A** can be one of 0001, 0010 or 0011, the difference in **B** has to be 1101 (see Section 3.2).

The main idea is to first generate all possible differential paths which conform to the truncated paths. Note that we do not compute values or generate conditions in this step. We use 5 short inbound phases to filter for differential paths from both sides with a final filtering step in the middle (see Fig. 5). In each inbound phase, we first compute all possible differential paths independently and merge the results of two adjacent inbound phases. We have carefully estimated the number of possible differential paths. In each step, this number is significantly lower than 2^{128}. Note that we apply the inbound phase at every S-box layer and also need to merge the resulting solutions through the S-box layer. This is not trivial for very large lists and explained in detail in Section 3.6. Also note that in AES-based primitives, differences between input and output of S-boxes can be filtered gradually due to the column-wise operation of MixColumns. Since this is not the case for the MixWord transformation of *Luffa*, more complicated filtering steps are needed.

After determining all possible differential paths, the next step is to solve for conforming input pairs to the permutation. We start by computing values of SubCrumb in the middle rounds 2, 3, and 4 simultaneously using a linear approach, similar as in the linearized match-in-the-middle step of [9]. Also this step has to be adapted since some S-boxes are not active and therefore, do not behave linearly. For each solution of these three rounds, all values of the state are determined and we probabilistically compute the resulting pairs outwards in the outbound phase. Finally, those pairs which match the message injection will give a semi-free-start collision for *Luffa*-256.

3.5 The Inbound Phases

In this section, we filter the truncated path for all possible differential paths. In total, about $2^{68.7}$ differential paths are possible, given the constraints on the message injection. We determine these paths by applying 5 inbound phases in the 5 middle rounds. In these inbound phases, we only determine which differences are possible but do not choose actual values for the state. Note that each active S-box can have one out of $15 \sim 2^{3.9}$ differences at the input or at the output. However, only 96 out of these $15 \cdot 15$ differences are possible for the *Luffa* S-box (see the differential distribution table of this S-box). Hence, the probability of

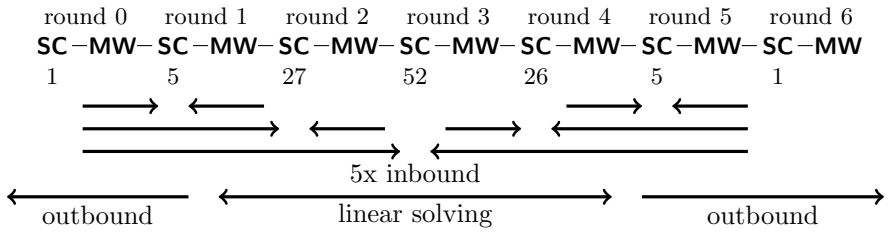

Fig. 5. A schematic view of the advanced rebound attack on *Luffa*. The attack consists of 5 small inbound phases to determine possible differential paths, a linear solving step to find the values and two outbound phases.

a differential match is about $96/225 \sim 2^{-1.23}$. Contrary to the AES S-box, we need to compute more exact probabilities to get correct results for *Luffa*.

The first inbound phase is applied to the truncated differential path $1 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 27$ in round 0 and 1. We can have $15 \sim 2^{3.9}$ non-zero differences at the input of the S-box (one active nibble in the first MixWord) and $15^5 \sim 2^{19.5}$ non-zero differences at the output (5 active nibbles in the second MixWord). Since only a fraction of $2^{-1.23 \cdot 5} = 2^{-6.15}$ differentials are possible through the 5 active S-boxes, we get in total about

$$2^{3.9} \cdot 2^{19.5} \cdot 2^{-6.15} = 2^{17.3}$$

possible differential paths for the first inbound phase. To support this estimate, we have computed the exact number of differential paths conforming to this truncated path, which is $2^{17.49}$.

Next, we continue with a second inbound phase between rounds 0-2. In the forward direction, we take all solutions of the previous inbound phase and in backward direction, we determine all possible differences in $27 - \mathsf{MW} - 52$. Note that in general, 27 active nibbles would expand to 58 active nibbles through MixWord. Therefore, we get a 24-bit condition on these differences and consider only the valid $2^{3.9 \cdot 27 - 24} = 2^{81.5}$ differences in round 3. These differences match with the previously computed ones through the 27 active S-boxes with a probability of $2^{-1.23 \cdot 27} = 2^{-33.2}$. In total, we get

$$2^{17.3} \cdot 2^{81.5} \cdot 2^{-33.2} = 2^{65.6}$$

possible differential paths for the truncated differential path $1 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 27 - \mathsf{SC} - 27 - \mathsf{MW} - 52$.

Now, we repeat the same procedure in backward direction for the second half of the path. The third inbound phase deals with rounds 4-5 and the path $26 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 1$. Similarly to the first inbound phase, we get $2^{19.5}$ differences for MixWord in round 4 and $2^{3.9}$ in round 5, which results in $2^{17.3}$ differential paths for this inbound phase. We continue with the fourth inbound phase and combine these paths with all possible differences in round

3 for $52 - \mathsf{MW} - 26$. To get 52 active nibbles instead of 57, we need a 20-bit conditions and get $2^{3.9 \cdot 26 - 20} = 2^{81.6}$ differences. We match all differences in backward and forward direction at the 26 active S-boxes and get

$$2^{17.3} \cdot 2^{81.6} \cdot 2^{-31.9} = 2^{67}$$

possible differential paths for the truncated differential path $52 - \mathsf{MW} - 26 - \mathsf{SC} - 26 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 1$.

3.6 The Final Inbound Phase: Parallel Matching

Next, we need to match all $2^{65.6}$ paths of the first three rounds with the 2^{67} paths of the last three rounds at the middle S-box layer. Since we have 52 active S-boxes, only $2^{65.6+67-1.23 \cdot 52} = 2^{68.8}$ differential paths will survive. However, the complexity of a straight forward filtering step is $2^{65.6} \cdot 2^{67} = 2^{132.6}$, which exceeds the birthday bound for collisions in *Luffa*-256. Therefore, we use a *parallel matching* technique to independently match differences in sets of 13 S-boxes. This reduces the complexity of the final inbound phase to about 2^{102} table lookups.

We first generate two sorted lists $\mathcal{A}$ and $\mathcal{B}$ from the differential paths found in the previous two inbound phases: List $\mathcal{A}$ contains all $2^{65.6}$ paths of round 0-2, sorted by the 52 non-zero input differences $\{x_1, \ldots, x_{52}\}$ of the active S-boxes in round 3. Similarly, list $\mathcal{B}$ contains all 2^{67} paths of round 3-5, sorted by the 52 non-zero output differences $\{y_1, \ldots, y_{52}\}$ of the same S-boxes. Next, we separate these 52 S-boxes into sets of 13 S-boxes which we match independently. Note that for each set of 13 S-boxes only $15^{13} = 2^{50.8}$ non-zero input or output differences exist. It follows that we can associate $2^{65.6-50.8} = 2^{14.8}$ elements from list $\mathcal{A}$ or $2^{67-50.8} = 2^{16.2}$ elements from list $\mathcal{B}$ to each of these $2^{50.8}$ differences.

Using the two lists $\mathcal{A}$ and $\mathcal{B}$, we generate two new and sorted lists $\mathcal{C}$ and $\mathcal{D}$ with differential matches for the first 13 and second 13 active S-boxes. List $\mathcal{C}$ contains differential matches between $\{x_1, \ldots, x_{13}\}$ and $\{y_0, \ldots, y_{13}\}$, and list $\mathcal{D}$ between $\{x_{14}, \ldots, x_{26}\}$ and $\{y_{14}, \ldots, y_{26}\}$. The resulting size of each list is $2^{50.8+50.8-13 \cdot 1.23} = 2^{85.6}$. The complexity to match two times $2^{50.8}$ differences at 13 S-boxes is about 2^{102} table lookups.

Next, we associate the differences $\{x_{14}, \ldots, x_{52}\}$ of $\mathcal{A}$ to $\mathcal{C}$, and the differences $\{y_1, \ldots, y_{13}, y_{27}, \ldots, y_{52}\}$ of $\mathcal{B}$ to $\mathcal{D}$. The resulting list $\mathcal{C}'$ contains the differences $\{y_1, \ldots, y_{13}, x_1, \ldots, x_{52}\}$ sorted by $\{y_1, \ldots, y_{13}\}$, and list $\mathcal{D}'$ contains the differences $\{x_{14}, \ldots, x_{26}, y_1, \ldots, y_{52}\}$ sorted by $\{x_{14}, \ldots, x_{26}, y_1, \ldots, y_{13}\}$. The size of the resulting new list $\mathcal{C}'$ is the size of $\mathcal{C}$ multiplied by all elements of $\mathcal{A}$ which can be associated to each of $x_1, \ldots, x_{13}$. The same holds for $\mathcal{D}'$ with elements of $\mathcal{B}$. Hence, we get $2^{85.6+14.8} = 2^{100.4}$ entries in $\mathcal{C}'$ and $2^{85.6+16.2} = 2^{101.8}$ entries in $\mathcal{D}'$.

Finally, we merge the two lists $\mathcal{C}'$ and $\mathcal{D}'$ by their overlapping differences $\{y_1, \ldots, y_{13}, x_{14}, \ldots, x_{26}\}$. Since we need to match differences in 26 active nibbles, we get a 101.4-bit condition. This results in about $2^{100.4+101.8-101.4} = 2^{100.8}$ solutions with a complexity of about 2^{102} simple table lookups (we do not need

to consider S-box differentials here anymore) and memory. Of these solutions, about $2^{-1.23 \cdot 26} = 2^{-32}$ differential paths give valid differentials also for the remaining 26 S-boxes. Hence, we get in total about $2^{100.8-32} = 2^{68.8}$ differential paths for round 0-6.

3.7 Linear Solving for Pairs

In the previous step, we have constructed all $2^{68.8}$ possible differential paths for the given truncated differential path of Section 3.3. However, to get conforming input pairs, we still need to determine the actual values. Note that we can only find values for a fraction of the differential paths. In the following, we show how to determine these paths and how to construct all values with a low average cost, by extending the linear approach published in [9].

We start by constructing solutions which satisfy the differential path in rounds 2-4 first. Since these rounds contain most active S-boxes, they are usually considered as the most expensive rounds. Note that each active S-box in *Luffa* restricts the number of possible values to either 2 or 4 elements, which can be described by an affine space: If there are two solutions, the input values can be described as $ax + b$, and the output values as $cx + d$. In this case x is a boolean variable common for input and output, and a, b, c, d are elements from Z_2^4. If there are four solutions, they are defined by two binary variables and two affine equations, one for the input and one for the output.

Hence, we can describe the set of all possible values in the active S-boxes of the three middle rounds using about $1.23 \cdot (26+52+27) = 129$ boolean variables. Note that this number is slightly different for each particular path, depending on the actual number of solutions for each S-box. These variables are linked by the equations of the linear MixWord transformation and the (affine) AddConstant in round 3 and 4. In total, we get at least $(26 + 27) \cdot 4 = 212$ equations which link the active S-boxes of the three middle rounds.

Unfortunately, these equations involve 12 inactive S-boxes in round 3, whose input and output values can not be described by affine spaces with common boolean variables. However, the input and output values can be described by a linear space separately. Note that there is also one inactive S-boxes which does not influence the active S-boxes of round 4. Thus, only those 4 variables describing its input are involved in the linear system. For the remaining 11 inactive S-boxes we get $11 \cdot 8 = 88$ additional variables. In total, we have to match the result over 11 S-boxes and get 212 equations in $129 + 4 + 88 = 221$ variables.

To summarize, we expect to get 2^9 solutions for each differential path with a complexity of 2^{23} simple bit operations. Most of these solutions are actually filtered out by the 44-bit filter of the 11 inactive S-boxes. In total, we therefore get $2^{68.8+9-44} = 2^{33.8}$ solutions for the three middle rounds. The complexity of this step does not exceed 2^{80} simple bit operations. Next, we compute these solutions forwards and backwards and check whether they also conform to the remaining differential path in the outbound phase.

3.8 The Outbound Phase

In the outbound phase, we simply propagate the solutions of the three middle rounds outwards and check whether the remaining path and the conditions on the message injection (see Section 3.2) are fulfilled. The probability that a solution of the previous step also fulfills the differential of the 5 active S-boxes in round 1 and 6 is $2^{-(4-1.23)\cdot 10} = 2^{-27.7}$. Therefore, we get in total about $2^{33.8} \cdot 2^{-27.7} \sim 2^6$ pairs for the whole 7-round truncated differential path:

$$1 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 27 - \mathsf{SC} - 27 - \mathsf{MW} - 52 - \mathsf{SC} - 52 - \mathsf{MW} - 26 - \mathsf{SC} - 26 - \mathsf{MW} - 5 - \mathsf{SC} - 5 - \mathsf{MW} - 1.$$

To satisfy the required differences at the input, we need to get one out of three 2-bit difference at the input of the single active S-box in round 0. At the output of the single active S-box in round 7 we need to match one specific difference. Hence, the conditions at both input and output are satisfied with a probability of $\frac{3}{15} \cdot \frac{1}{15} = 2^{-6.2}$.

Next, we repeat all previous steps for each of the three permutations and check whether we can find a message according to the input and output differences of the permutations. Note that the input difference of one permutation can be corrected by choosing an appropriate message differences. Hence, we only need to ensure that the input differences in two permutations match. The probability that these three 2-bit differences match is $\frac{1}{3^2} = 2^{-3.2}$. Hence, the probability to get the right input and output differences in all three permutations is $2^{3\cdot(6-6.2)-3.2} = 2^{-3.8}$, which is actually not enough to get a semi-free-start collision for *Luffa*-256.

However, as already noted in Section 3.3, we can rotate the differential path 32 times. Since the values cannot be rotated due to the addition of the round constant we need to repeat the search for conforming pairs from Section 3.7, but not the expensive part of finding the differences that verify the path. To get a semi-free-start collision for *Luffa*-256, we have to repeat this about $2^{3.8} = 14$ times. To summarize, the most expensive part of the attack is the computation of all possible differential paths in the final inbound phase for each of the three permutations. Hence, the complexity to find a semi-free-start collision for *Luffa*-256, or equivalently a collision on the internal state of 768 bits, is about 2^{104} in time and 2^{102} memory.

4 Building an 8-Round Distinguisher from the 7-Round Semi-free-start Collision

If we consider the previous explained procedure for building the semi-free-start collision on 7 rounds, and we look at the differential path, it is easy to see that one round later, so after the eighth rounds, if the path is followed, only 8 nibbles will be active before the last MixWord and all the other ones inactive. By considering not the bits but the linear combination of bits, we get the same result after the last MixWord. We can then build a distinguisher on the compression

function for 8 rounds, with the same complexity as before, where we will find $(64-8)\times 3$ combination of nibbles in the output without any difference, what means a collision on 672 bits with a complexity of about 2^{104}, which is much lower than the birthday paradox. If we would try to build such a collision exploiting a generic attack on the general mode, we could control one permutation with the message insertion, but this will mean colliding on 512 bits, and if we want to have some active nibbles in all the permutations, we couldn't do any better than the birthday paradox on 672 bits, so our distinguisher has clearly a lower complexity.

5 Distinguisher for 8 Rounds of the Permutation

We use a differential distinguisher which sends a certain number of queries to a black-box B and will decide in the end if the black-box is the permutation of *Luffa* or a random oracle. The innovative idea is that the distinguishing algorithm works in two parts. In the first part, we apply a first test $\mathcal{T}_1$ to N input quadruples, where one value is chosen randomly and the three others differ from the first one in some nibbles in a deterministic manner. The test $\mathcal{T}_1$ is passed if the quadruple fulfills a specific property $\mathcal{P}$. As we will see later, to test the property $\mathcal{P}$ of a quadruple we need on average $2(1+p)$ queries to the black-box, where $p << 1/2$ is a given probability. Thus, $\mathcal{T}_1$ involves $2(1+p)N \approx 2N$ queries to the black-box. Only a subset of the original quadruples will pass $\mathcal{T}_1$. To this subset we will apply a second test $\mathcal{T}_2$ which uses several calls to the black-box for each tested quadruple. However, since the number of quadruples we have to test for $\mathcal{T}_2$ is much lower, the overall complexity is determined by the one of $\mathcal{T}_1$. The probability of passing the two tests is much higher for the permutation of *Luffa* than for a random oracle.

The distinguisher is based on a differential path represented in Fig. 6. As we will see later, the path over the eight rounds is divided in four parts: one inverted first round, three rounds of differential path, three rounds of truncated differential path, and one last round which maintains a distinguishing linear property. Let a quadruple be defined by the four 256-bit states $z=(z_1,z_2,z_3,z_4)$. Our goal is to find quadruples such that the pairs (z_1,z_2) and (z_3,z_4) follow the path. Because of the first inverted round, a quadruple is constructed in the following way: We choose the first message z_1 randomly. Let $LS(z_j^i)$, $RS(z_j^i)$ denote the i'th nibble on the left and right side of z_j, respectively, which will enter the first S-box layer. Then, for the messages z_j, $2\leq j\leq 4$, we keep $64-k_j$ out of the 64 nibbles as in z_1, and replace k_j nibbles by $f^L(LS(z_1^i),\alpha)=S^{-1}(S(LS(z_1^i))\oplus\alpha)$ and $f^R(RS(z_1^i),\alpha)=S'^{-1}(S'(RS(z_1^i))\oplus\alpha)$, respectively. The numbers of changed nibbles are $k_2=32$, $k_3=11$ and $k_4=29$, where 28 nibbles are the same in all four states. The construction of a quadruple can be easily done by either computing f^L and f^R each time or by a lookup in two tables of 16×15 entries. We are able to perform this inversion of the first round (and not more) as it is going to determine a big structured subset of valid input quadruples with the previous equations, where there is no difference in 28 nibbles and all the $LS(z_1^i)$

and $RS(z_1^i)$ can be randomly chosen. For example, if more rounds were inverted, the validity of the distinguisher might become controversial as differences will spread over all the input and, for not having probability involved, some of those values should be fixed for having the wanted difference at the beginning of the differential path. So if there was no structure on the two pairs of inputs and the values were determined, this could be compared with just inverting the permutation from four outputs with the wanted property and obtaining four concrete inputs that have no structure at all, which for obvious reasons can not be considered a distinguisher. This is not the case when we just invert one round. In the following, we won't differentiate each time between the left and the right side if the general method stays the same, instead we use directly f and z_j. Because of the best probability of the differential transitions of the S-boxes, we choose the difference α to be either $0x2$ or $0x4$.

The differential path in Fig. 6 has no difference in 6 nibbles (24 bits) in round 7. The 8th round is formed by a SubCrumb and a MixWord phase. We consider first the SubCrumb phase, and we can notice that it is not going to modify the property of having 6 nibbles with no difference. Then the MixWord is applied. We recall here that this phase is linear. This will mean that, from the output after the 8th round, there are 24-bit linear relations which values collide when the differential path is verified. This is equivalent to finding the collision on the 6 nibbles before the MixWord linear phase, so for the sake of simplicity, we will look for collisions before the last MixWord phase.

A quadruple z fulfills the property $\mathcal{P}$ if and only if after applying the black-box B, we have a collision in the 24 bits of the pair $(\mathsf{MixWord}^{-1}(\mathsf{B}(\mathsf{z_1})), \mathsf{MixWord}^{-1}(\mathsf{B}(\mathsf{z_2})))$ *and* in the pair $(\mathsf{MixWord}^{-1}(\mathsf{B}(\mathsf{z_3})), \mathsf{MixWord}^{-1}(\mathsf{B}(\mathsf{z_4})))$. The probability of $\mathcal{P}$ is defined by the differential path (Section 5.1) and the property described in Section 5.2, which shows that for each pair following the path we can find another pair following the path with probability one.

Test $\mathcal{T}_2$ will use the non-trivial property that for a quadruple fulfilling $\mathcal{P}$ and thus passing $\mathcal{T}_1$, we can change some nibbles in the four states such that with high probability the new quadruple will again fulfill $\mathcal{P}$. This property is described in detail in Section 5.3. Thus from the subset of quadruples that pass $\mathcal{T}_1$, we will be able to find new quadruples passing $\mathcal{P}$ with a much lower complexity.

The distinguisher works on the permutation. However, it can be easily adapted to the compression function by choosing the message according to the chaining value and by considering only the output of one permutation. For a known CV and a random message, let us consider for example the first permutation. Let m_i, h_i be the nibbles of the message and the value determined by the CV, which are XORed as input of the permutation. Then, for the other three message of the corresponding to a quadruple we will change some nibbles from m_i to $f(m_i \oplus h_i, \alpha) \oplus h_i$.

Related work. Our distinguisher is similar to some adaptive attacks on block ciphers. In these attacks a right pair for a differential provides information on

the internal computations, which may speed up the search for the next pairs if it is required. In the attack on RC5 [3] the second and next right pairs could be obtained with significantly lower complexity. In the attack on AES a single right pair provides information on several key bytes, so the attacker partially controls the first round and gets right pairs for the next differential much faster [2].

5.1 The Differential Path

We use the differential path in Fig. 6 which consists of four parts. Note that the last round and thus, part 4 is not represented in this figure:

1. Precomputation (round 0): We can guarantee a given difference in round 1 by choosing the blue nibbles in round 0 accordingly.
2. Differential (round 1-4): We use a difference that has probability 2^{-2} of passing from the input of the S-box to the output (for S and S'), e.g. $\alpha = 0x2$ or $\alpha = 0x4$. This part is responsible for the final probability.
3. Considering unaffected bits (round 4-7): In the end of round 4 we have a difference in one nibble. We consider how many nibbles in round 7 are never "affected" by this difference.
4. Round 8: As mentioned above, the property of unaffected nibbles in round 7 can be checked by linear combination of bits after round 8. For simplicity reasons we will omit this part for the further discussion.

For the differential path we only consider a pair of states (either (z_1, z_2) or (z_3, z_4)) not a quadruple. We will denote by x the first state to which we will add some difference.

In round 0, at every place where there is no difference we choose a random nibble x_i. In the other places we use the pairs $(x_i, f(x_i, \alpha))$, where x_i is chosen randomly. Now we have guaranteed that we have the differences α after the S-box and the corresponding differences in the beginning of round 1.

In the next part, we have three S-boxes layers with a total of $23 + 23 + 8$ active S-boxes, before we arrive in round 4. This gives us a probability of $2^{-54\times 2}$ of following this part, as 2^{-2} is the probability of passing one active S-box for the chosen α.

At the beginning of round 4 we only have a one bit difference. From rounds 4-7, we are only interested in which bits are unaffected by the difference in round 4. Thus, from the one bit difference in round 4, with probability one we have no differences in 24 bits (6 nibbles) at the end of round 7.

5.2 Changing the Parity of Differences

We can change the parity of some differences in round 1 such that in the case of a pair following the differential path, the new pair will also follow the differential path with probability one. This property will be used in $\mathcal{T}_1$. The general concept of changing the parity of differences is mainly the following: we consider one nibble with a difference γ, that is a nibble taking the value y for message M_1

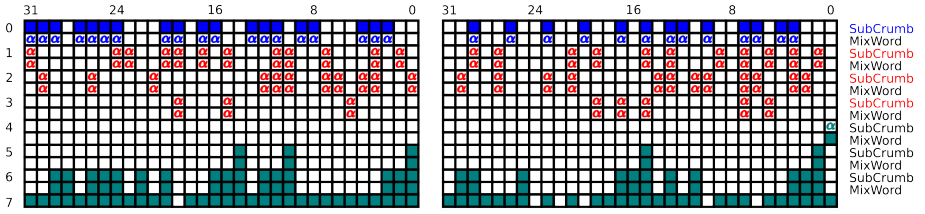

Fig. 6. Differential path used for the distinguisher over 8 rounds.

and the value $y \oplus \gamma$ for message M_2. If we change it's parity, it will take the value $y \oplus \gamma$ for message M_1 and the value y for message M_2. It is obvious to see that if this nibble is the only difference between M_1 and M_2, this parity change will have the effect of interchanging M_1 and M_2 and we will have gained nothing. However, if there is more than one difference, changing the parity of some of them will define two new input messages M_1' and M_2' with the same difference as the original ones and the same values in the nibbles without difference.

Let us consider an example. Let $x_{i_1}, x_{i_2}, x_{i_3}$ be some nibbles in round 1. Let the pair with the corresponding difference be $x_{i_1} \oplus \alpha, x_{i_2}, x_{i_3} \oplus \alpha$. Then, changing the parity of the difference in x_{i_1} leads to the pair $x_{i_1} \oplus \alpha, x_{i_2}, x_{i_3}$ and $x_{i_1}, x_{i_2}, x_{i_3} \oplus \alpha$. This change is equivalent to adding α at the position of x_{i_1} in the two states.

We are going to see how changing the parity of some well chosen differences in a pair of states that satisfies the differential path will immediately give us another pair of states that also verifies the path with probability one. For this, we have to take into account the effect that changing the parity of some differences at round r might have some rounds later:

- SubCrumb Each active S-box that follows the path verifies $S(x \oplus \alpha) \oplus S(x) = \alpha$. This property still holds when we change the parity of the difference of the nibble. This means that through any SubCrumb phase where the only change is the parity of some differences, the path will still be verified.
- MixWord Every difference after the linear transformation that is affected by a parity change in the previous step will also have its parity changed. Thus, it does not introduce any problem for verifying the path in next round (we would be in the starting case). However, when two differences cancel out and one had it's parity changed but not the other, the value of the corresponding nibble (in both states) will change from x to $x \oplus \alpha$. This won't affect the verification of the differential path for this round, but might affect the round $(r+2)$: the values for the nibbles (that have no difference) obtained after the SubCrumb of round $(r+1)$ will change at these positions. This might affect the values associated to nibbles with differences after the MixWord phase of round $(r+1)$. Next, after the SubCrumb phase of round $(r+2)$ the active S-boxes might not output the same difference as for the original pair (as their values are changed).

We are going to use this, once we have obtained a pair of states that verify the differential path, to obtain another pair verifying the path with probability one. In the differential path we want to have the parity of some differences changed at the beginning of the first round. To achieve this, we have to add α after the first S-box to the corresponding positions, like in Fig. 7. We have chosen these three positions as they verify that they generate no changes of values one round later (no differences cancel out where an odd number of parity changes at the beginning of round 2), and this way they will only affect the S-box two rounds later (round 4), as we explained before. As the difference that the S-box of round 4 outputs is not important for the verification of the differential path. When changing the parities of the three difference, the remaining differential path from 1 to 8 will also be verified.

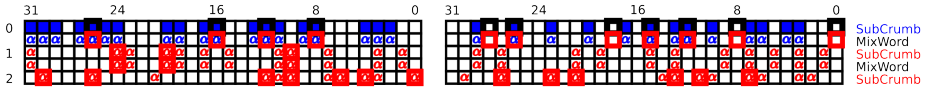

Fig. 7. Changing the parity of differences.

In Fig. 7, the red bordered rectangle means that we added α to the corresponding nibble. To see the effect that this has in the input pair, a thick black bordered rectangle in round 0 means that we have to change the value x_i to $f(x_i, \alpha)$. This can happen to positions with or without a difference in the original pair. With this, once we have found a pair of messages that verifies the path, if we generate a new pair from it by changing the nibbles associated to the positions of the black bordered rectangle in round 0 from the value x_i to the values $f(x_i, \alpha)$, we will have automatically generated a new pair that also verifies the path.

5.3 Changing Values Without a Difference

If we change in round 1 the value of the nibble at the 14th position on the right side, this will have an effect on 4 S-boxes in round 3. The same happens when we change the nibble at position 17 on the right side. If we change the two at the same time, this has an effect on 7 S-boxes. This property will be used in $\mathcal{T}_2$. The influenced bits are shown in Fig. 8. A red, blue and violet squares mean that a nibble was influenced, respectively by the nibble 14, by the nibble 17 or by the two nibbles.

We have 15 possibilities to change only position 14, 15 possibilities to change only position 17 and $15 \times 15 = 225$ to change the two.

Getting all possible values of the nibble in position 14 is equivalent to adding the difference β to the nibble in round 1, for all $\beta \in \{0x1, \ldots, 0xF\}$. This can be done by changing 8 nibbles in round 0 from x_i to $f(x_i, \beta)$, which will add β

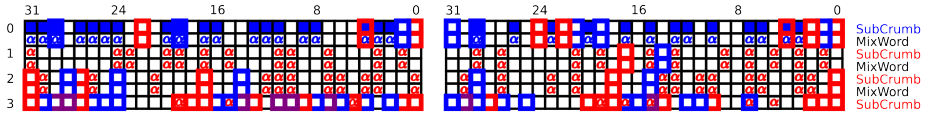

Fig. 8. Influence of the nibbles 14 and 17.

to 8 nibbles after the first S-box layer. The position of the 8 nibbles are marked in Fig. 8. The same can be done for position 17. If we change the two nibbles at the same time we will have to change 8 nibbles to $f(x_i, \beta_{14})$ and 8 nibbles to $f(x_i, \beta_{17})$.

Now for the original pair of messages we change the values of the nibbles at position 14 and 17 (on both messages) at round 1, send the corresponding values at round 0 to the black box and look if the result has again a collision in the 24 bits. For $i = 0, 1$, let p_i be the probability that when starting from a pair following the differential path and trying out all the 15 other values at round 0 that produce all the nibble values at position 14 (or 17), we obtain exactly i pairs with no differences in the 24 bits. Let q_i be the same probability when trying the 225 possibilities where the nibbles at position 14 and 17 are changed. Then we have

$$\begin{aligned} p_0 &= (1 - 2^{-8})^{15} \\ p_1 &= 15 \times 2^{-8}(1 - 2^{-8})^{14} \\ q_0 &= (1 - 2^{-14})^{225} \\ q_1 &= 225 \times 2^{-14}(1 - 2^{-14})^{224}. \end{aligned}$$

The total probability of having at least 2 new pairs out of 255 tried is

$$1 - \left(p_0^2 q_0 + 2 p_0 p_1 q_0 + p_0^2 q_1\right) = 2^{-8.3}.$$

For a pair not following the differential path, this happens with a probability of

$$1 - (1 - 2^{-24})^{255} - 255 \times 2^{-24}(1 - 2^{-24})^{254} = 2^{-33}.$$

5.4 Complete Distinguisher

For a random state z_1 we define the quadruple z as following: The state z_2 is set such that together with z_1 it forms an input pair of the differential path. The states z_3, z_4 is obtained by changing the parity of difference of the pair (z_1, z_2).

Test $\mathcal{T}_1$ We try N different quadruples. For each quadruple z we first test if the pair (z_1, z_2) has a collision in the 24 bits. The probability of this property is 2^{-24} in the general case. The probability of following the differential path is 2^{-108}. Only if we find a collision, we will test the pair (z_3, z_4). For a pair following the

path, this leads to a collision in the 24 bits with probability one, otherwise this will happen with probability 2^{-24}.

Thus in the case of the black-box being the permutation of *Luffa* we need $2N+2^{-23}N+2^{-107}N \approx 2N$ queries and find $N2^{-48}+N2^{-108}$ quadruples having the property $\mathcal{P}$ and, thus, passing $\mathcal{T}_2$.

In the case of a random function we will have $2N+2^{-23}N \approx 2N$ queries and will find $N2^{-48}$ quadruples with property $\mathcal{P}$.

As this test is not enough for distinguishing *Luffa*'s permutation from a random one, we need to define test $\mathcal{T}_2$.

Test $\mathcal{T}_2$ This test is applied only on those pairs that passed $\mathcal{T}_1$. It exploits the property of Section 5.3. The test $\mathcal{T}_2$ on a quadruple z works as follows. For each of the 255 pair of differences $(\beta_{14}, \beta_{17}) \in \{0x0, 0x1, \ldots, 0xF\}^2 \backslash (0,0)$, we create a corresponding quadruple z' by an addition of this differences to the positions 14 and 17 in round 1 of z_1, z_2, z_3, z_4. The test $\mathcal{T}_2$ is passed if and only if, out of the 255 new quadruples, at least 2 have the property $\mathcal{P}$. For the test of $\mathcal{P}$ we check again first if (z'_1, z'_2) have a collision and only in this case we check (z'_3, z'_4).

For each quadruple z' it is still valid that the pair (z'_3, z'_4) is obtained from (z'_1, z'_2) by changing the parity of differences. Thus (z'_1, z'_2) follows the differential path if and only if (z'_3, z'_4) follows the differential path. In the case of a quadruple z following the differential path, we find at least 2 new quadruples z' following the differential path and thus having property $\mathcal{P}$ with probability $2^{-8.3}$.

In the random case, we find at least two z' such that (z'_1, z'_2) has a collision in the 24 bits with probability $2^{-33.1}$. The probability of each of these quadruples also having a collision for (z'_3, z'_4) is 2^{-24}. So the probability of passing $\mathcal{T}_2$ is $2^{-33.1-2\times 24} = 2^{-81.1}$. We get the same result by considering the probability

$$1 - (1-2^{-48})^{255} - 255 \times 2^{-48}(1-2^{-48})^{254} = 2^{-81}.$$

In the worst case we find a collision for all the 255 differences (β_{14}, β_{17}), which would mean that we have to send about 2^{10} queries to the black-box. However, the percentage of initial quadruples passing $\mathcal{T}_1$ is much less than 2^{-10}. This means that the dominant costs come from $\mathcal{T}_1$.

Combining the two Tests In the case of the permutation of *Luffa* the probability of finding a quadruple following the differential path and passing $\mathcal{T}_2$ is $2^{-108-8.3} = 2^{-116.3}$. For a random oracle, or in the general case, a quadruple build from a random value z_1 passes $\mathcal{T}_1$ and $\mathcal{T}_2$ with probability $2^{-48-81.1} = 2^{-129.1}$. Thus, for $N = 2^{116.3}$ we will be able to distinguish with high probability the permutation of *Luffa* from a random oracle. The time complexity of this distinguisher is $2N$ queries. The memory complexity is negligible, since we apply the two tests on the fly.

6 Conclusion

We developed a number of new differential techniques for the analysis of *Luffa*. Our results do not threaten the security of *Luffa* as they are on building blocks and not on the full hash function. Even though they do not contradict the designers' claims, our results improve upon previous work in several ways. When considering collision attacks on the hash function with limited access to internal variables, also less degrees of freedom are available for an attacker. Still, we argue that the new techniques in this paper will be very useful to analyze *Luffa* further in this setting. Also, the improvements to the rebound attack are likely to be useful in the attacks on non-AES-based designs.

Acknowledgements

We would like to thank Christian Rechberger for his many helpful comments and his contribution to this work.

References

1. Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, available at `http://www.131002.net/data/papers/AM09.pdf` (2009)
2. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: CRYPTO'09. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009)
3. Biryukov, A., Kushilevitz, E.: Improved cryptanalysis of RC5. In: EUROCRYPT'98. pp. 85–99 (1998)
4. Dai Watanabe, Y.H., Yamada, T., Kaneko, T.: Higher Order Differential Attack on Step-Reduced Variants of Luffa v1. FSE (2010), to appear
5. De Cannière, C., Sato, H., Watanabe, D.: Hash Function Luffa: Specification. Submission to NIST (Round 1) (2008), `http://ehash.iaik.tugraz.at/uploads/e/ea/Luffa_Specification.pdf`
6. De Cannière, C., Sato, H., Watanabe, D.: Hash Function Luffa: Specification. Submission to NIST (Round 2) (2009), `http://www.sdl.hitachi.co.jp/crypto/luffa/Luffa_v2_Specification_20091002.pdf`
7. De Cannière, C., Sato, H., Watanabe, D.: Hash Function Luffa: Supporting Document. Submission to NIST (Round 2) (2009), `http://www.sdl.hitachi.co.jp/crypto/luffa/Luffa_v2_SupportingDocument_20090915.pdf`
8. Indesteege, S.: The LANE hash function. Submission to NIST (2008), available at `http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf`
9. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In: Jacobson, Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. LNCS, vol. 5867, pp. 16–35. Springer (2009)
10. 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)
11. Wagner, D.: A generalized birthday problem. In: CRYPTO'02. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer (2002)

A The Truncated Differential Path for each Permutation

In the main article we showed only a differential path for the permutation Q_0. Here, we show also the remaining two paths for permutations Q_1 and Q_2 for the same position of the output difference as in Q_0. In all three examples, the output difference is in the 14th nibble of the left side.

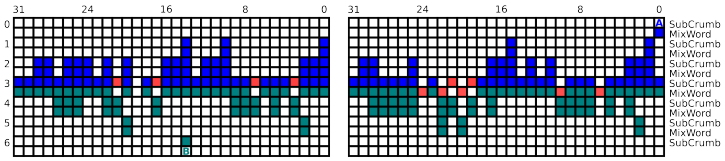

Fig. 9. Truncated differential path for Q_0. The value **A** corresponds to a difference of 0001, 0010 or 0011, the value **B** to 1101.

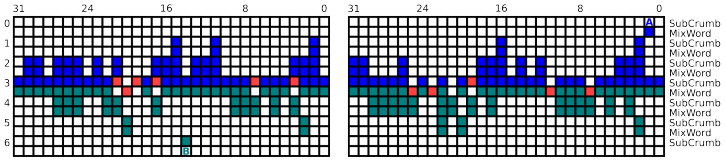

Fig. 10. Truncated differential path for Q_1. The value **A** corresponds to a difference of 0010, 0100 or 0110, the value **B** to 1000.

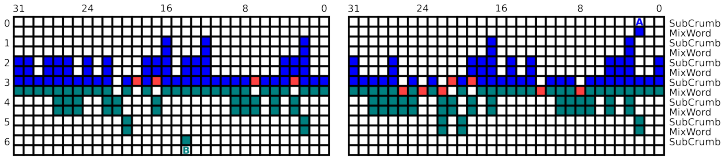

Fig. 11. Truncated differential path for Q_2. The value **A** corresponds to a difference of 0100, 1000 or 1100, the value **B** to 0010.