Finding Collisions for Reduced *Luffa*-256 v2

Bart Preneel[2], Hirotaka Yoshida[1 2], and Dai Watanabe[1]

[1] Systems Development Laboratory, Hitachi, Ltd.,
292 Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa-ken, 244-0817 Japan
[2] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium

Abstract. Luffa is a family of cryptographic hash functions that has been selected as a second round SHA-3 candidate. This paper presents the first collision finding analysis of *Luffa*-256 v2 which is the 256-bit hash function in the Luffa family. We show that collisions for 4 out of 8 steps of Luffa can be found with complexity 2^{90} using sophisticated message modification techniques. Furthermore, we present a security analysis which shows how difficult it is to apply the same approach to *Luffa*-256 v2 reduced to 5 steps: the resulting attack would require a complexity of 2^{224}. This analysis can be seen as an indication that the full 8 steps of the *Luffa*-256 v2 hash function has a large security margin against differential collision search with message modification technique.
Keywords: Hash functions, differential cryptanalysis, collision attack, message modification

1 Introduction

A cryptographic hash function is an algorithm that takes input strings of arbitrary (typically very large) length and maps these to short fixed length output strings. A secure cryptographic hash function has to satisfy the following requirements:

- **preimage resistance**: it is computationally infeasible to find any input which hashes to any pre-specified output.
- **second preimage resistance**: it is computationally infeasible to find any second input which has the same output as any specified input.
- **collision resistance**: it is computationally infeasible to find a collision, i.e. two distinct inputs that hash to the same result.

For an ideal hash function with an n-bit output, finding a preimage or a second preimage requires about 2^n operations and the fastest way to find a collision is the birthday attack which needs approximately $2^{n/2}$ operations.

Recent cryptanalytic results focus on the collision resistance of hash functions. Collision attacks [15] have been shown for many commonly used hash functions, such as MD5 [13] and SHA-1 [11]. In response, NIST launched the SHA-3 competition [12] which aims to find an alternative hash function to the SHA-2 family. NIST received more than 60 candidate hash functions and it currently focuses on the 14 second round candidates. Therefore, the cryptanalysis of these hash function designs is of great interest.

Luffa is a family of cryptographic hash functions that has been selected as a second round SHA-3 candidate. The hash function *Luffa* adopts the structure of a sponge function and a wide-pipe strategy. Furthermore, the design of *Luffa* shows a distinct feature that the internal round function consists of parallel applications of permutations. The four proposed variants of *Luffa* compute a 224-bit, 256-bit, 384-bit, and 512-bit hash value respectively.

In the previous results on *Luffa*, the building blocks have been extensively analyzed: the designers of *Luffa*-256 v2 found a differential path for the internal permutation of *Luffa* with a probability of 2^{-224} [5]. Aumasson and Meier [1] constructed an algebraic zero-sum distinguisher for the same component with a complexity of 2^{82}. Watanabe *et. al* constructed a higher order

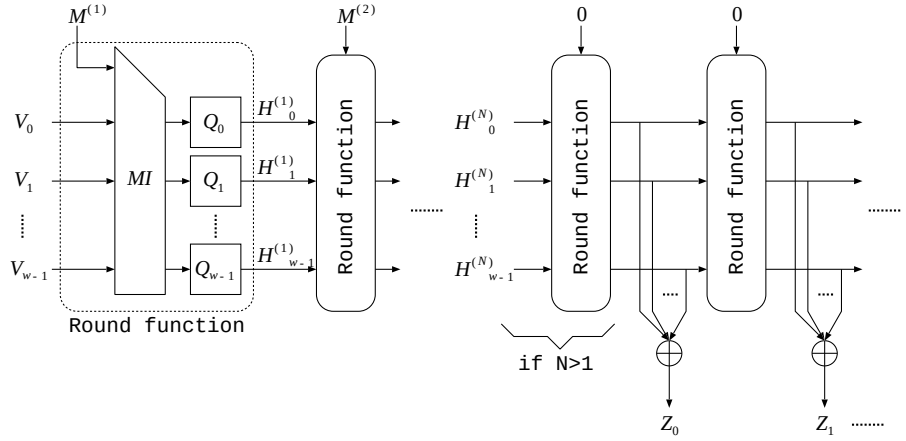

Fig. 1. The *Luffa* construction

distinguisher for 7-steps out of 8 steps of the compression function of *Luffa* v1, requiring 2^{216} one-block messages. Khovratovich *et. al* [9] found a semi-free start collision for the 7-steps of the compression function of *Luffa*-256 v2 with a complexity of 2^{104}, which can be extended to an 8-step distinguisher with the same complexity [9].

This article analyses the collision resistance of reduced-round versions of *Luffa* which is the 256-bit hash function in the *Luffa* family. To the best of our knowledge, this is the first analysis of this type, where the attacker is more restricted than in the previous analysis because the initial vector is fixed in ours. We show how collision attacks, using sophisticated message modification techniques, can be mounted on reduced variants of *Luffa*-256 v2. We show an attack on *Luffa*-256 v2 reduced to from 8 to 4 steps with a complexity of 2^{90}. Furthermore, we analyze how difficult it would be to apply the same approach to *Luffa*-256 v2 reduced to 5 steps, which can be seen as an indication that the full 8 steps of the *Luffa*-256 v2 hash function has a high security margin against differential collision search using message modification.

The outline of this paper is as follows. In Sect. 2, we give a short description of the hash function *Luffa*-256 v2 with a focus on the relevant parts for our attacks. In Sect. 3, the results of the collision attacks on 4-step variant of *Luffa*-256 v2 are presented. Section 4 analyzes the resistance against collision attacks of a 5-step variant of *Luffa*-256 v2. Section 5 concludes the paper.

2 Specification of *Luffa*-256 v2

In this section, we introduce a part of the specification of *Luffa* which is needed to describe the attack. The reader is referred to [6] for the details of the specification.

2.1 Chaining

The chaining of *Luffa* is a variant of a sponge function [2, 3], that processes 256 message bits in each iteration. The message is padded with `10...0` to ensure that the padded message has a length divisible by 256. Figure 1 shows the basic chaining structure.

Round Function. The round function is a composition of a message injection function MI and w permutations Q_j of 256 bits input (see Fig. 1). Let the input of the i-th round be

$(H_0^{(i-1)}, \ldots, H_{w-1}^{(i-1)})$, then the output of the i-th round is given by

$$H_j^{(i)} = Q_j(X_j), \quad 0 \le j < w,$$
$$X_0 || \cdots || X_{w-1} = MI(H_0^{(i-1)}, \ldots, H_{w-1}^{(i-1)}, M^{(i)}),$$

where $H_j^{(0)} = V_j$.

In the specification of *Luffa*, the input length of the sub-permutation Q_j is fixed to $n_b = 256$ bits, and the number of the sub-permutations w is 3, 4 and 5 for the hash lengths 256, 384 and 512 bits respectively.

The message injection functions can be represented by a matrix over the ring $\mathrm{GF}(2^8)^{32}$. The map from an 8-word value $(a_0, \ldots, a_7)$ to an element of the ring is defined by $(\sum_{0 \le k < 8} a_{k,l} x^k)_{0 \le l < 32}$. Note that the least significant word a_7 is the coefficient of the heading term x^7 in the polynomial representation.

2.2 Non-Linear Permutation

The permutation Q_j is defined as the composition of an input tweak and iterations of a step function `Step`. The number of iterations of a step function is 8 and the tweak is applied only once per a permutation.

At the beginning of the step function process, the 256 bits data are stored in 8 32-bit registers denoted by $a_k^{(r)}$ for $0 \le k < 8$. The data before applying the permutation Q_j is denoted by b_k and the data after the tweak is denoted by $a_k^{(0)}$. The step function consists of the following three functions: `SubCrumb`, `MixWord`, `AddConstant`. The pseudocode for Q_j is given by

```
Permute(a[8], j){ //Permutation Q_j
    Tweak(a);
    for (r = 0; r < 8; r++){
        SubCrumb(a[0],a[1],[2],a[3]);
        SubCrumb(a[5],[6],a[7],a[4]);
        for (k = 0; k < 4; k++)
            MixWord(a[k],a[k+4]);
        AddConstant(a, j, r);
    }
}
```

Each function is described below in turn and the tweaks are described in Section 2.2. We omit the description of `AddConstant` because it is not needed in this paper.

Substitution. `SubCrumb` substitutes the bits of a_0, a_1, a_2, a_3 (or a_4, a_5, a_6, a_7) by a 4-bit S-box S defined by

$$S[16] = \{\mathtt{13}, \mathtt{14}, \mathtt{0}, \mathtt{1}, \mathtt{5}, \mathtt{10}, \mathtt{7}, \mathtt{6}, \mathtt{11}, \mathtt{3}, \mathtt{9}, \mathtt{12}, \mathtt{15}, \mathtt{8}, \mathtt{2}, \mathtt{4}\}.$$

Let the output of `SubCrumb` be x_0, x_1, x_2, x_3 (or x_4, x_5, x_6, x_7). Then the substitution by `SubCrumb` is given by

$$x_{3,l} || x_{2,l} || x_{1,l} || x_{0,l} = S[a_{3,l} || a_{2,l} || a_{1,l} || a_{0,l}], \quad 0 \le l < 32,$$
$$x_{4,l} || x_{7,l} || x_{6,l} || x_{5,l} = S[a_{4,l} || a_{7,l} || a_{6,l} || a_{5,l}], \quad 0 \le l < 32.$$

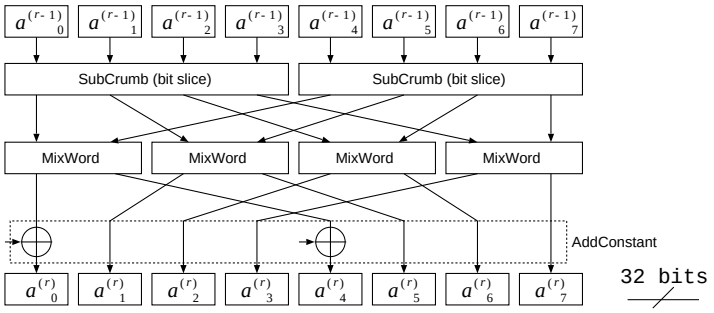

Fig. 2. The step function

Linear Diffusion. `MixWord` is a linear permutation of two words. Let the output words be y_k and y_{k+4} where $0 \leq k < 4$. Then `MixWord` is given by the following equations:

$$\begin{aligned} y_{k+4} &= x_{k+4} \oplus x_k, \\ y_k &= x_k \lll \sigma_1, \\ y_k &= y_k \oplus y_{k+4}, \\ y_{k+4} &= y_{k+4} \lll \sigma_2, \\ y_{k+4} &= y_{k+4} \oplus y_k, \\ y_k &= y_k \lll \sigma_3, \\ y_k &= y_k \oplus y_{k+4}, \\ y_{k+4} &= y_{k+4} \lll \sigma_4. \end{aligned}$$

The parameters σ_i are given by $\sigma_1 = 2, \sigma_2 = 14, \sigma_3 = 10, \sigma_4 = 1$.

Tweaks. For each permutation Q_j, the least significant four words of a 256-bit input are rotated by j bits to the left in 32-bit registers. Let the j-th block, k-th word input be $b_{j,k}$ and the tweaked word (namely the input to the first step function) be $a^{(0)}_{j,k}$, then the tweak is defined by

$$\begin{aligned} a^{(0)}_{j,k,l} &= b_{j,k,l}, \quad 0 \leq k < 4, \\ a^{(0)}_{j,k,l} &= b_{j,k,(l-j \bmod 32)}, \quad 4 \leq k < 8. \end{aligned}$$

3 The Collision Attack on 4-step *Luffa*-256 v2

We present a collision attack on 4-step *Luffa*-256 v2 using three message blocks. Our attack constructs a differential path [4] producing a collision and then applies message modification [15] in order to reduce the complexity. The overview of our strategy to carry out this technique is that we give the degrees of freedom of message bundles to the active S-boxes through steps in a way that we apply to the modification as independently as possible, which can be performed by developing an algorithm for assigning degrees of freedom in an appropriate way and by considering the order in which message bundles or a group of them are used.

3.1 Preliminary

In order to simplify the description of our attack, we will view the 256-bit message block as 32 8-bit bundles and consider their positions t ($0 \leq t < 32$), to which we will refer as *message*

bundle and *message bundle position* respectively. Each of these bundles is obtained in a bit-slice manner as adopted in *Luffa*-256 v2: one bit of a bundle is taken from one 32-bit word of in the message block.

For the same reason, we will view the 256-bit internal state of the permutation Q_j as 64 4-bit bundles, each of which is taken as input to S-box, and consider their positions u ($0 \leq u < 64$), to which we will refer as *S-box position*. We call position u less than 32 higher and otherwise a lower position.

The higher 4-bits of a message bundle of position t only affect the input to an S-box at the higher position t at the first step in Q_j while the lower 4-bits only affect the S-box at the lower position $t + 32$.

3.2 The Differential Path

We construct good differential paths for the round function from good truncated differential paths for the permutation Q_j. To search for the latter paths, we consider the linear code given by the iteration of `MixWord`. The reason for this is explained below. Let A be the representation matrix of `MixWord` and $G_n = I||A|| \cdots ||A_n$ be the generator matrix of a code. Then good truncated paths can be directly obtained from low weight code words of the linear code G_n, under the assumption that the output differences of the S-boxes are the same at all positions. Now our way of constructing a good differential path can be summarized as follows:

1. Construct a good truncated differential path which is the same for permutations by constructing a low weight code for `MixWord` in the way that was explained in the previous section.
2. Convert it into (non-truncated) differential paths for the whole round function by choosing appropriate differences of S-boxes.

We exhaustively searched for low weight code words of G_n which allow us to obtain truncated differential paths with a small number of active S-boxes in the permutation Q_i. Furthermore, in order to bypass the effect of the Tweak function, we impose some condition on the inputs generating the low weight code words that there are only 1's in the higher 32-bit word and therefore there are only 0's in the lower 32-bit word.

Our experiments found many good truncated differential paths. The best one we found has 49 active S-boxes which is shown in Table 1.

Table 1. The truncated differential path for Q_j

Step	Weight	0	1	2	3	4	5	6
		0123456789	0123456789	0123456789	0123456789	0123456789	0123456789	0123
0	07	0000000100	0100000001	0101010000	0100000000	0000000000	0000000000	0000
1	08	0100000100	0000010001	0100000101	0000000000	0000000000	0000000000	0010
2	19	0001010100	0100011100	1001000100	1100000000	0000000110	1010000000	1110
3	15	0000010000	0001010100	0101000100	0010110010	0010000000	0000000010	1010
4	(42)	0011011000	1110110001	0110011010	1111101110	1111110111	1110011110	0111

We now convert the truncated differential path to a (non-truncated) differential path for the whole round function by choosing the input differences of the S-boxes, satisfying the above assumption.

We firstly consider to determine an input difference in the message block. The active message bundle positions are the same as the active S-box positions at the first step in the path shown

in Table 1. For each of active message bundles, we use the same one-byte difference which will be determined below. Since the truncated path has active S-boxes only in higher positions, this one-byte difference in a message bundle must be the form of `0xY0` where `Y` is a non-zero 4-bit value.

After the message injection function MI, it is necessary that the active S-box positions are the same for all of three Q_js. It follows that the above one-byte difference is now limited to be `0x10`, `0x20`, or `0x30`. This is because that, in the path, one message bundle position t ($0 \leq t < 32$) affects the S-boxes at two positions (t and $t+32$), a wrong choice other than these three for the one-byte difference in a message bundle at position g would cause an undesirable situation where positions g and $g+32$ are active in some Q_j while position g is active but $g+32$ is passive in the other $Q_{j'}$'s.

Next we determine the (non-truncated) output differences for Q_0, Q_1, and Q_2 per bundle, which is consistent to the truncated path in Table 1. Depending on the bundle position, the above output difference of Q_j must be a form of `0xYZ` or `0xY0` or `0x0Z` where both of the `Y` and `Z` are non-zero values.

Considering this and the linear condition of producing a collision after MI in the third round function, we can determine the (non-truncated) output differences for Q_0, Q_1, and Q_2 per bundle: (`0x40`, `0x10`, `0xb0`) or (`0x04`, `0x01`, `0x0b`) or (`0x44`, `0x11`, `0xbb`).

It follows that the output differences of S-boxes at the fourth step for Q_0, Q_1, and Q_2 are `0x4`, `0x1`, and `0xb` respectively. Starting from these output differences and the above three input difference candidates, namely (`0x1`, `0x2`, `0x4`), (`0x2`, `0x4`, `0x8`), (`0x3`, `0x6`, `0xc`), for Q_0, Q_1, and Q_2, we searched for difference paths of the S-boxes that give the best value for the product of differential probabilities over Q_js. This search has been performed based on the differential profile table for the S-box shown in [5]. Among the above three input differences, two of them give the the best value for the product of differential probabilities. In the end, we determine the input difference to be (`0x1`, `0x2`, `0x4`) because this gives a differential path where we place S-box input differences with a low probability in the steps with a low weight, namely step 0 and step 1 in the truncated differential path in Table 1. This is not the case for the other input difference (`0x2`, `0x4`, `0x8`). Hence, we also determine the one-byte difference in the message bundle to be `0x1` out of three candidates. Our result is shown in Table 2 that lists the input differences of S-boxes and the products (over Q_js) of differential probabilities of S-boxes at the same position.

Table 2. The differences and the product of differential probabilities for S-boxes at each position.

Step	Q_0	Q_1	Q_2	Product of the probabilities
0	`0x1`	`0x2`	`0x4`	2^{-7}
1	`0x6`	`0x2`	`0x4`	2^{-7}
2	`0x4`	`0xd`	`0x4`	2^{-6}
3	`0x4`	`0x1`	`0x4`	2^{-6}
	`0x4`	`0x1`	`0xb`	

In the way explained the above, we have constructed a differential path with the desired property that the active S-box positions u ($0 \leq u < 64$) at each step in each permutations Q_j are exactly the same in all of three Q_js.

3.3 The Message Modification Technique

The main technical difficulty in our collision attacks lies in applying the message modification technique which has been used as a key tool to find differential collisions in hash functions.

For a given differential path, this technique allows to find a set of messages that give a higher differential probability than one would expect from randomly chosen messages. More specifically, one bit condition on an input typically increases the differential probability by a factor of two on average.

However, there is a potential problem in the practical application of this technique: it could be difficult to satisfy many bit-conditions because some of them are likely to conflict with the others. Therefore, an important task for the attacker is to ensure that the message bundles are used without encountering any contradiction over which values are set for them.

We will show that one can apply the message modification technique to 4-step *Luffa*-256 v2 without having the problem. Roughly speaking, this can be achieved by developing a tool for assigning degrees of freedom in the message to the inputs of the active S-boxes, each of which generally depends on plural message bundles, except for one in the first step where it depends on a single message bundle.

3.4 Choosing Good Internal States with the First Message Block.

For each message bundle position g, the product of the differential probabilities for each permutation Q_j could vary from 2^{-6} to 2^{-18} as there are three permutations where the differential probability for S-box lies between 2^{-2} and 2^{-3}. For the first step, we could face the difficulty of lack of degrees of freedom: we may have to satisfy an 18-bit condition, where we have only 8 bits of degrees of freedom in the message bundle.

However, we will solve it by randomly choosing the first message block $M^{(0)}$ to find a good internal state $H^{(1)}$. We mean by good internal state that there exists a second message block satisfying conditions imposed by the active S-boxes in the first step. We confirmed by experiment that the probability to have one byte of a good internal state in each permutation is 2^{-2}. Since there are 7 active S-boxes in each permutation in the first step, the complexity required for obtaining a good internal state $H^{(1)}$ is 2^{14}.

3.5 The Second Message Block of the Differential Path

Assuming that $H^{(1)}$ is a good internal state, we now use the second message block $M^{(1)}$ in order to find right values for the differential path.

Firstly, we deal with the first step of *Luffa*-256 v2. As for a message bundle of position t, the input difference is `0x10`. From this input difference it follows that the S-box differential paths at position t in the first step are `0x1` $\rightarrow$ `0x6`, `0x2` $\rightarrow$ `0x2`, and `0x4` $\rightarrow$ `0x4`, for the permutations Q_0, Q_1, Q_2 respectively. Note that we have no difference in the lower S-boxes. We confirmed by experiment that the probability for a message bundle being right for this path is 2^{-5}. After applying the basic message modification to 7 active S-boxes, the remaining degrees of freedom in the second message block is 221 bits out of 256 bits. A detailed description of the remaining degrees of freedom at each message bundle position is shown in Table 3.

Table 3. The degrees of freedom remaining after the first step.

Message bundle position	0	10	20	30
Degrees of freedom	8888888388	8388888883	8383838888	83

For the second and the third steps of *Luffa*-256 v2, we apply the message modification technique to check whether there exist right values for the differential path. Our approach is to verify that one can give the attacker enough degrees of freedom in the message bundles, that

would allow him to fulfill the conditions on the input to each active S-boxes by adjusting the second message block $M^{(1)}$. Our careful choice of message bundles ensures that this modification for the third step can be performed independently of the modification for the second step. The approach for this can be explained in the following way: the second step modification uses 12 message bundles and it determines 10 of these bundles which will be fixed during the third-step modification while the message bundles at positions 9 and 13 will not be fixed. From our way of using the degrees of freedom and consideration of the effect of the `MixWord` linear layer, we can verify that the inputs of active S-boxes at the third step can be modified only by using the message bundles at positions 9 and 13, and message bundles on which the inputs of active S-boxes at the second step do not depend. Hence, at the third step, we do not use any of the 10 message bundles which have been fixed at the second step. On the other hand, for the message bundles at positions 9 and 13, we store values for them which give the right values to the active S-boxes the second step and then at the third step we choose the right values from them, which results in increasing the complexity of the third step complexity by a small factor of 2^5. This is how we apply the message modification between steps and how we apply it within one step will be later on explained in discussions on the complexity of the message modification.

The verification was performed by means of an experiment for which an algorithm is given in 1.

Algorithm 1 An algorithm of verifying that one can give the attacker enough degrees of freedom in the message bundles

```
1: for i from 0 to 63 do
2:    if S-box is active at position i then
3:       for j from 0 to 6 do
4:          for k from 0 to 63 do
5:             if S-box of ord_f[i] depends on message bundle of ord_l[k] then
6:                if ord_l[k] > 0 then
7:                   ord_l[k]− = 1; /* use degrees of freedom */
8:                   break;
9:                end if
10:            end if
11:         end for
12:      end for
13:   end if
14: end for
```

In the algorithm, $\texttt{ord}_\texttt{f}$ is an array storing the order in which degrees of freedom are assigned for active S-boxes; $\texttt{ord}_\texttt{l}$ is an array storing the order in which message bundles are used. Before the algorithm is carries out, sorting procedures are performed with $\texttt{ord}_\texttt{f}$ and $\texttt{ord}_\texttt{l}$ such that the algorithm assigns the degrees of message freedom to the more restricted active S-boxes with higher priorities. In the algorithm, the reason why j varies from 0 to 6 is that the product of probabilities for active S-boxes for Q_j at the second step is 2^{-7} in Table 2.

Table 4 indicates the correspondence between the active S-box positions and the message bundle positions for the second and the third step.

In the second and the third step, the products of the differential probabilities for the S-boxes in the same (S-box) position over Q_j are 2^{-7} and 2^{-6} respectively. After applying the message modification to 8 active S-boxes and 19 active S-boxes, the degrees of freedom in the second message block is 165 bits (out of 256 bits) remaining after the second step and 51 bits remaining after the third step. A detailed description of the remaining degrees of freedom in each message bundle position is shown in Table 5. The complexity required for this procedure is negligible, which will be explained in the next sub-section. Roughly speaking, the reason for this is that

Table 4. The correspondence between the conditions for the active S-box positions and the message bundle positions for the second and the third step.

Step 1	S-box pos.	27	7	62	21	29	1	15	19		
	Message bundle pos.	7,19,21	1,19,31	15	1,15	9,23	25,27	9,27	11, 13		
Step 2	S-box pos.	27	62	3	30	5	17	52	20	7	11
	Message bundle pos.	2	2,13	9,24	3,24,26	26	29	29,30	3,30	10	10,12
	S-box pos.	47	31	15	16	23	61	48	50	60	
	Message bundle pos.	5,12	3,5	28	0,28	0,4	4	6	8	14	

it follows from the correspondence of the active S-boxes and the message bundles that one can find the right values in the same method as one can solving systems of linear equations using the substitution method.

Table 5. The degrees of freedom remaining after the second and the third steps.

Message bundle position	0	10	20	30
After step 1	8088888081	8084808880	8080808088	80
After step 2	0000052020	0000208880	8080000000	00

As for the fourth step, the product of differential probabilities for S-boxes in the same (S-box) position over Q_j is 2^{-6}. Since there are 15 active S-boxes, too few degrees of freedom are remaining to apply message modification. Therefore, we randomly choose the first message block $M^{(1)}$ to repeat the whole attempt $2^{90-51} = 2^{39}$ times. After this procedure, we find a right input satisfying all the conditions for the 15 active S-boxes. As a result, we expect to find a collision for 4-step *Luffa*-256 v2 with a total complexity of $2^{90} \leftrightharpoons 2^{39}(2^{14} + 2^{51})$.

On the Complexity of the Message Modification. In the previous paragraph, we showed how we assign the degrees of message freedom to the active S-boxes. We here discuss the complexity of the message modification. At the first step, it is clear how to apply the technique. The complexity of the message modification is only 7×2^8 because there are 7 active S-box positions and we can independently modify values for each message bundles.

However, we face more difficult situations at the second and the third steps due to the effect of the `MixWord` which ensures that the input to an S-box at the second step depends on *multiple* message bundles and that one message bundle may affect *multiple* active S-boxes. Hence, the potential problem is that even if a condition on the input of an active S-box is fulfilled by means of a modification of some message bundle, this fulfillment can be afterwards destroyed by means of a following modification of another message bundle which again affects the input of this active S-box. Therefore, an appropriate order in which message modification is applied has to be determined to reduce the complexity.

Here we investigate the case of the third step as the second step could be dealt in the same way. From all message bundles to be modified and all the active S-boxes, we consider a group whose element has a form of a pair of (p and $(q_1, q_2, ...)$) where p is a message bundle position and q_is are the each active S-box positions influenced by its modification. We can construct seven groups where, for each element, at least one active S-box position is appeared in another element. For example, 30 of (3, (20,30,31)) is appeared in (24, (3, 30)) as well.

In this way, we can apply message modification group per group. Here we take as an example the largest group shown in Table 7 because the corresponding complexity is dominant in the message modification.

Table 6. A Group of relations between message bundle positions and active S-box positions

message bundle position	3	5	9	10	12	24	26	29	30
active S-box position	20,30,31	31,47	31	7,11	11,47	3,30	5,30	17,52	20,52

Now our way of applying message modification to this group can be summarized in Table 7 that indicates how and in which order the message bundles are used for the corresponding the active S-boxes at the third step, which means that the attacker firstly chooses values for message bundles at positions 3, 9, 10, 12, 29, 30 and store the values giving a right input to the S-box at position 52 and next the attacker chooses values for message bundles at positions 0 and 2 and store the values giving a right input to the S-box at position 17 and so on. the dominant complexity corresponds to the first procedure in message modification of this group, which is $2^{8\times5+1} = 2^{41}$ derived from the degrees of freedom described in Table 5. Putting things together, we estimate that the time complexity for the message modification at the third step is 2^{41}, which is not dominant in computing the total complexity because this complexity of 2^{41} is significantly less than the complexity of 2^{51} corresponding to the use of the second message block.

Table 7. Message modification to the largest group, indicating how and in which order the message bundles are used for the corresponding the active S-box at the third step.

order	1	2	3	4	5
message bundle position	3, 9, 10, 12, 29, 30	0,2	4,8	6,26	5,14,24
active S-box position	52	17	20	5	3,7,11,30,31,47

4 Security Analysis of 5-step *Luffa*-256 v2

We present a security analysis which shows how difficult it is to apply the same approach to *Luffa*-256 v2 reduced to 5 steps, which can be seen as an indication that the full 8 steps of the *Luffa*-256 v2 hash function has a large security margin against differential collision search with message modification [15].

4.1 The Differential Path

As we performed for 4-steps of *Luffa*-256 v2 in section 3, our experiments found many good truncated differential paths. The best one we found has 76 active S-boxes which is shown in Table 8.

Table 8. The truncated differential path for Q_j

Step	Weight	0	1	2	3	4	5	6
		0123456789	0123456789	0123456789	0123456789	0123456789	0123456789	0123
0	10	0000010001	1010000011	0001000100	1100000000	0000000000	0000000000	0000
1	24	0011101001	0110010100	1000000100	1000110011	0001100001	1010001000	1010
2	12	0011100000	0000010100	0001000000	0001010000	0000010000	0001010000	0001
3	16	1101100000	0000000001	1100001000	0010110010	0000000000	1101000000	0001
4	14	0000000101	1010101000	1010000010	1000000000	0001000000	0001000100	0100
5	(34)	1110010111	0110101011	1110001010	1101001110	1000011001	0011000111	0001

We now convert the truncated differential path to a (non-truncated) differential path for the whole round function by choosing the S-box input differences in Table 9.

Table 9. The S-box input differences in Q_j and the product of differential probabilities for the S-boxes in Q_j.

Step	Q_0	Q_1	Q_2	Product of probability
0	0x1	0x2	0x4	2^{-7}
1	0x8	0x2	0x4	2^{-6}
2	0x2	0x2	0x4	2^{-6}
3	0x2	0x2	0x4	2^{-6}
4	0xd	0xd	0x4	2^{-7}
	0x4	0x1	0xb	

4.2 The First Message Block to Choose Good Internal States.

We randomly choose a first message block $M^{(0)}$ to find a good internal state $H^{(1)}$. We confirmed by experiment that the probability to have one byte of a good internal state in each permutation is 2^{-2}. Since there are 10 active S-boxes in each permutation in the first step, the complexity required for obtaining a good internal state $H^{(1)}$ is 2^{20}.

4.3 The Second Message Block of the Differential Path

Assuming that $H^{(1)}$ is a good internal state, we now use the second message block $M^{(1)}$ in order to find right values for the differential path.

We confirmed by experiment that the probability for a message bundle being right for this path is 2^{-5}. After applying the message modification to 10 active S-boxes with a probability 2^{-2}, the degrees of freedom remaining in the second message block is 206 bits out of 256 bits. Both in the second and the third steps, the products of differential probabilities for S-boxes in the same (S-box) position over Q_j are 2^{-6}. After applying the message modification to 24 active S-boxes and 8 active S-boxes out of 12, the degrees of freedom in the second message block are 62 bits and 14 bits out of 256 bits remaining after the second and the third steps respectively. A detailed description of the degrees of freedom remaining after each step at each message bundle position is shown in Table 10.

Table 10. The degrees of freedom remaining after each step.

	0	1	2	3
Message bundle position	0123456789	0123456789	0123456789	01
After step 0	8888838883	3838888833	8883888388	33
After step 1	0040800580	0003802600	0080008020	00
After step 2	0000300020	0000100000	0000008000	00

As for the second and the third step of *Luffa*-256 v2, the conditions on the input to each of 24 active S-boxes for each permutation can be fulfilled by adjusting the second message block $M^{(1)}$. Table 11 indicates the correspondence between the active S-box positions and the message bundle positions for the second and the third step.

The complexity required for this procedure is negligible if we assume that we could perform as we did in 3.5, which is optimistic for the attacker.

Table 11. The correspondence of the active S-boxes to the message bundles

Step 1	S-box position	15	11	6	4	20	49	30	35	39	12
	Message bundle position	9,27	5,23	18,30	16	0,12	0,2	10,24	20	2,20,24	6
	S-box position	43	56	27	9	38	50	52	60	62	44
	Message bundle position	6,24	7,19	21	1,21	1,31	1,3	3,13	11	11,13,25	25
	S-box position	2	17	3	34						
	Message bundle position	28	29	15,29	15,17						
Step 2	S-box position	45	35	17	23	4	15	55	2		
	Message bundle position	2,7	7,16,17	14,17	14	13,22,28	22	4,22	8		

In order to fulfill conditions for the remaining 8 active S-boxes at the third step, and the ones at the forth and fifth steps, we randomly choose the first message block $M^{(1)}$ to repeat the whole attempt $2^{204=((12-8)\times 6)+(16\times 6)+(14\times 7)-14)}$ times. Therefore, this approach would take a high total complexity of $2^{224} \fallingdotseq 2^{204}(2^{20} + 2^{14})$ even if the optimistic assumption on the complexity of message modification is satisfied.

5 Conclusion

We show that collisions for 4 out of 8 steps of *Luffa* can be found with complexity 2^{90} using sophisticated message modification techniques. Furthermore, we present a security analysis which shows how difficult it would be to apply the same approach to *Luffa*-256 v2 reduced to 5 steps. For the future work, it would be interesting to construct a differential path constructed from differential paths for the internal permutations Q_is that are different from one Q_i to another. It would be worthwhile to search for differential paths that are most suitable for message modification, even though its differential probability is not necessarily the highest one. However, we speculate that it could be very difficult to carry out these two approaches. Based on our results and speculation, we conclude that *Luffa*-256 v2 has a substantial security margin against differential collision search with message modification.

References

1. J.P. Aumasson and W. Meier, "Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi," 2009. Available at `http://www.131002.net/data/papers/AM09.pdf`.
2. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Sponge Functions," Ecrypt Hash Workshop 2007.
3. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "On the Indifferentiability of the Sponge Construction," *Advances in Cryptology, Eurocrypt 2008*, pp. 181–197, 2008.
4. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 1993.
5. C. De Cannière, H. Sato, D. Watanabe, "Hash Function Luffa: Supporting Document," Submission to NIST SHA-3 Competition, 2008. Available at `http://www.sdl.hitachi.co.jp/crypto/luffa/`.
6. C. De Cannière, H. Sato, D. Watanabe, "Hash Function Luffa: Specification," Submission to NIST SHA-3 Competition, 2008. Available at `http://www.sdl.hitachi.co.jp/crypto/luffa/`.
7. I. B. Damgård, "A design principle for hash functions," Advances in Cryptology - CRYPTO '89, LNCS, vol. 435, pp. 416–427, 1990.
8. M. Knezevic and I. Verbauwhede, "Hardware evaluation of the Luffa hash family," `ftp://ftp.esat.kuleuven.ac.be/cosic/knudsen/trunc.ps.Z`.
9. D. Khovratovich, M. N. Plasencia, A. Roeck, M. Schlaeffer, "Cryptanalysis of Luffa v2 components," Selected Areas in Cryptography, SAC2010, August 2010.
10. R. C. Merkle, "One way hash functions and DES," CRYPTO'89, LNCS, vol. 435, pp. 428–446, 1990.
11. National Institute of Standards and Technology, "Secure hash standard," Federal Information Processing Standards Publication 180-2, August 2002. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`.
12. National Institute of Standards and Technology, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family," `http://csrc.nist.gov/groups/ST/hash/documents/`, November 2007.

13. R. Rivest, "The MD5 message-digest algorithm," Request for Comments, no. 1321, April 1992. `ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt`.
14. D. Watanabe, Y. Hatano, T. Yamada and T. Kaneko, "Higher Order Differential Attack on Step-Reduced Variants of *Luffa v1*," *Fast Software Encryption, FSE'2010*, Lecture Notes in Computer Science, LNCS 6147, Springer-Verlag, pp. 270–285, 2010.
15. X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," Advances in Cryptology - CRYPTO 2005, LNCS, vol. 3621, pp. 17–36, 2005.