The Reasons for The Change of *Luffa*

Christophe De Cannière[1], Hisayoshi Sato[2], and Dai Watanabe[2]

[1] ESAT-COSIC, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10 bus 2446, B-3001 Leuven-Heverlee, Belgium
[2] Systems Development Laboratory, Hitachi, Ltd.,
292 Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
dai.watanabe.td@hitachi.com

1 Introduction

The hash function *Luffa* submitted to NIST SHA-3 competition changes its algorithm at the Round 2. The reason for the change is that the algebraic property is not so good as expected. We changed several components of *Luffa* in order to improve the property. In this document, we explain what are changed and why they are changed. In addition, we intend to clarify the influences to the security and the implementation by the changes. Hereafter, the first round algorithm and the second round algorithm of *Luffa* are denoted by *Luffa* v1 and *Luffa* v2 respectively.

The rest of this document is organized as follows: Firstly, the higher order differential attack on *Luffa* v1 is briefly introduced in Section 2. Secondly, the changes of the algorithm and their reasons are explained in Section 3. Then the influences of the changes to the security and to the implementation issues are discussed in Section 4 and 5 respectively.

2 Background

2.1 Higher Order Differential Attack on Step-reduced Variant of *Luffa* v1

Starting from the private communication with Yamada and Kaneko [6][1], we developed the higher order differential attack on the step-reduced variant of *Luffa* v1. The detailed attack is presented in the independent document [8]. Here we just refer the main result.

First of all, we found that the algebraic degree of the underlying non-linear permutation Q_j grows slower than an ideal case. In addition, we found that there are very efficient distinguishers[2] for the step-reduced variant of *Luffa* v1. According to our theoretical estimate, we can construct distinguishers for step-reduced variants of *Luffa* v1 up to 7 out of 8 steps by using a block message. The

[1] The early result will be published in [7].

[2] In this document, we use the terminology *distinguisher* for functions which detect a kind of non-randomness property according to [1]. Note that a distinguisher is usually a terminology for a function which distinguishes two random variables.

expected degrees of the distinguishers are summarized in Table 1. For 7 steps, the algebraic degree of the distinguisher is expected 214 and the distinguisher requires 2^{216} inputs.

This distinguisher can be extended to the hash function by the careful choice of the messages without an extra cost. This attack does not threaten any security of full-step of *Luffa* v1.

Table 1. The summary of the algebraic degrees

Number of steps	1	2	3	4	5	6	7	8
Distinguisher's degree	–	2	5	13	33	84	214	545

We identified that the this property is mainly caused by the following two reasons:

- The algebraic expressions of the Sbox are poor[3].
- `MixWord` preserves a part of this property.

In addition, no blank round for a block message enables the direct extension of the higher order differential property of Q_j to the distinguishing attack on the hash function.

2.2 Impact of The Distinguishing Attack

Even though there is a distinguisher for 7 out of 8 steps of *Luffa*, we believe that this does not affect the security of *Luffa* v1. There are two reasons:

- The security margin against this attack is sufficiently large.
- The attack does not make sense under any conceivable setting.

Security Margin The current distinguisher applies to 7 steps. Whereas we expect that the data requirement of this distinguisher (2^{216} messages) can be reduced using more advanced techniques, the large margin predicted by our theoretical estimates suggests that it is rather unlikely that a similar property could be detected for 8 steps, even if all 2^{256-1} available messages would be used.

Advantage of The Adversary Different from keyed cryptographic functions, it is not immediately obvious what advantage such a distinguisher would give to the adversary.

Let us consider two hypothetical situations: The first situation is that the adversary is given a hash function as a black box, and is asked to determine

[3] This has been pointed out in [4].

whether or not it is 7 steps *Luffa*. Clearly, the higher order differential distinguisher is of no use in this case, since *Luffa* is a white box. It suffices for the adversary to feed a few inputs to the black box to verify that the black box is indeed 7 steps *Luffa* by calculating the outputs for the same inputs by himself.

The second situation is that the adversary is given the XOR of the hash values of a set of 2^{216} messages satisfying the requirements of the distinguisher, and is asked to determine, without knowing exactly which messages were hashed, whether 7 steps *Luffa* was used or not. Using the distinguisher, the adversary can easily answer this question by verifying that this XOR contains some repeated words.

Again, the advantage of the adversary is not clear in this situation. In the higher order differential attack, a part of the message is fixed and the remaining part takes all possible values so that the adversary knows that messages with all possible values of varying part are fed to the black-box and he only does not know what the fixed value is. Therefore only 2^{40} bits are secret for the adversary in this situation[4].

In more abstract description, we consider the adversary who intends to distinguish a family of functions consisting of 2^{40} elements from a random function. The trivial distinguisher makes a few queries and compares the outputs with those of the all possible 2^{40} elements. Then the distinguisher determines whether or not the black box is a function from the family according to the comparison between the outputs and the possible values calculated in advance. This fact indicates that any distinguisher which requires more queries (or computational complexity) than the above trivial distinguisher is of no use.

Of course one could imagine even more contrived situations in which the advantage provided by the distinguisher would be more significant, but then again, for any given hash function, one may devise artificial protocols which would become insecure when instantiated with this particular hash function. Moreover, the fact that the distinguisher only applies to a very restricted class of messages (i.e., messages of at most 255 bits), really limits the possibilities of the adversary.

2.3 Reason for Changes

Despite the fact that we do not consider the distinguisher discussed above to be a threat for *Luffa* v1, we realized that we could further increase the security margin against this distinguisher without sacrificing any of the existing security or efficiency properties of the hash function. In view of this, we felt that it would make little sense not to consider this improvement.

3 The Changes and The Reasons

We changed the algorithm of *Luffa* at the following three points:

[4] It might be pointed out that the adversary does not know which bits of the word are fixed. In this case, we have to take in account about $2^{17.6}$ additional secret bits derived from the possibilities to choose 5 out of 32 bit positions.

- In the finalization process, we apply a blank round even for a block message.
- The Sbox is changed.
- The order of the inputs $\mathtt{a}[4], \mathtt{a}[5], \mathtt{a}[6], \mathtt{a}[7]$ to `SubCrumb` is changed.

The details of the changes are explained below.

3.1 Finalization

We changed the finalization of *Luffa* as follows:

Luffa **v1** A blank round is applied if the message length is not smaller than 256 bits.
Luffa **v2** A blank round is always applied.

This change is intended to reinforce the security of *Luffa* against the attack introduced in Section 2 and this is the most essential improvement. If a blank round is always applied, the attacker has to construct the distinguisher for 16 step functions to detect a kind of non-randomness so that the change is expected to significantly increase the security margin not only against the higher order differential attack but also against any kind of distinguishing attack.

3.2 Sbox

We changed the Sbox of *Luffa* as follows:

Luffa **v1**

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

Luffa **v2**

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

This change is intended to reinforce the security of *Luffa* against the attack introduced in Section 2. Hereafter the Sbox for *Luffa* v1 and *Luffa* v2 are denoted by S_1 and S_2 respectively for the convenience. The Sbox S_1 has the following properties:

- Its bit slice implementation can be executed in 6 cycles on Intel Core2 processors [5].
- The maximum differential and linear probabilities are optimum.
- The Boolean polynomial expressions of the map are of degree 3.
- They have no fixed point.

The new Sbox S_2 satisfies these properties and we chose it from ones whose polynomial expressions have more monomials than that of S_1. We had a bit of difficulty to find such an Sbox, because the Sboxes output by our old search program have very similar polynomial expressions. In order to fix this problem, we relaxed the constraints for the combination of instructions. We are going to publish our approach to find the Sboxes in the near future.

[5] Intel is a registered trademark and Core is a name of products of Intel Corporation in the U.S. and other countries.

3.3 SubCrumb

In addition to the change of the Sbox, we changed the function SubCrumb as follows:

Luffa **v1**

$$\texttt{SubCrumb}(\texttt{a}[0], \texttt{a}[1], \texttt{a}[2], \texttt{a}[3]);$$
$$\texttt{SubCrumb}(\texttt{a}[4], \texttt{a}[5], \texttt{a}[6], \texttt{a}[7]);$$

Luffa **v2**

$$\texttt{SubCrumb}(\texttt{a}[0], \texttt{a}[1], \texttt{a}[2], \texttt{a}[3]);$$
$$\texttt{SubCrumb}(\texttt{a}[5], \texttt{a}[6], \texttt{a}[7], \texttt{a}[4]);$$

This change is intended to break a kind of symmetry in the non-linear permutation Q_j. This symmetry is considered a factor of the undesirable algebraic property of Q_j. In fact, we found that this change more strongly improves the algebraic property of Q_j than the change of the Sbox.

4 Improvements of The Security by The Changes

The additional blank round for a block message does not affect the known security evaluations except the newly reported higher order differential attack [7, 8]. If a blank round is always applied, the attacker has to construct the distinguisher for 16 step functions and the best known higher order differential attack provides distinguishers up to 7 steps [8]. Therefore *Luffa* v2 is considered to have sufficient resistance against the higher order differential attacks. In addition, this change is considered to be efficient to avoid an extension of a zero-sum attack [2] which is a variant of the higher order differential attack.

The changes in SubCrumb are subtler approaches.

The maximum differential probability and the maximum linear probability of the Sbox S_2 are the same as those of S_1. The algebraic degrees of the polynomial expression of S_2 are also the same as those of S_1 and they do not have a fixed point. Therefore the change of the Sbox does not affect the known security evaluation results. The change of the order of input words does not affect the known security evaluation results for the same reason.

The next problem is if these changes reinforce the security of *Luffa* against the higher order differential attack. Table 2 shows the theoretical estimate of the algebraic degrees of the distinguisher for S_1 and S_2. The effect of the change of the order of input words is ignored in the estimation.

The substitution of the order of input words makes the theoretical estimate difficult because the mixing of MixWord is no more ignorable. We investigated the effects by the experiments.

Table 3 and 4 show the higher order differential properties for the combinations of the Sboxes and the order of the input words. The numeric values in the table mean the probabilities of the distinguishing attacks being successful.

Table 2. The summary of the algebraic degrees

	Number of steps							
	1	2	3	4	5	6	7	8
v1	–	2	5	13	33	84	214	545
v2	–	2	5	13	35	94	252	675

Table 3. The summary of the algebraic degrees of distinguishers

Sbox	`SubCrumb`	Order												
		11	12	13	14	15	16	17	18	19	20	21	22	23
v1	v1	0.90	0.97	1	1	1	1	1	1	1	1	1	1	1
v2	v1	0.78	0.90	1	1	1	1	1	1	1	1	1	1	1
v1	v2	0.55	0.85	0.99	1	1	1	1	1	1	1	1	1	1
v2	v2	0	0	0.01	0.01	0.09	0.18	0.34	0.53	0.73	0.86	0.97	0.99	1

Table 4. The summary of the algebraic degrees of distinguishers

Sbox	`SubCrumb`	Order										
		22	23	24	25	26	27	28	29	30	31	32
v1	v1	0.02	0.03	0.12	0.20	0.37	0.45	0.69	0.81	0.83	0.91	0.99
v2	v1	0	0	0	0	0	0.01	0	0.03	0.05	0.07	0.16
v1	v2	0	0	0	0	0	0	0	0	0	0	0
v2	v2	0	0	0	0	0	0	0	0	0	0	0

These experimental results indicate that each change of `SubCrumb` improves the resistance against the higher order differential attack. Especially, the effect by the change of the order of input words is significant.

From the Table 3, the algebraic degree of the distinguisher for 4 steps in *Luffa* v2 is estimated at 23 and it is much larger than that in *Luffa* v1. If the Sbox and the `MixWord` have ideal properties, the degree of distinguisher should be $3^3 = 27$. The experimental result indicates that the step function of *Luffa* v2 achieves a good property and the degree of the distinguisher approaches the ideal case.

5 Influences to The Implementation Aspects

Because of an additional blank round for a short message, the processing speed of *Luffa* v2 for a very short message becomes half of *Luffa* v1. On the other hand, there is no influence if the message whose length is not less than 256 bits.

The influences of the changes in `SubCrumb` are relatively minor. The Sbox S_2 is decomposed into 16 instructions and it can be executed in 6 cycles on the Intel

Core2 processors[6]. This is same as the property of S_1 so that their speeds on this platform are expected to be the same. The change of the order of the input words has a small influence in the implementations on some embedded CPUs. The comparison of performances of *Luffa* v1 and *Luffa* v2 on AVR ATmega8515[7] and Intel Core2 are shown in Table 5 and 6 respectively.

Table 5. Execution time and memory requirements on AVR ATmega8515

Version	Execution time		Memory requirements	
	One-block msg. (cycles/message)	Very long msg. (cycles/byte)	Code size + constant data (bytes)	RAM (bytes)
Luffa v1	23,458	695.7	640+120	132
Luffa v2	46,627	738.1	690+120	134

Table 6. Speed of ANSI C codes on Intel Core2 Duo

Version	32-bit		64-bit	
	One-block msg. (cycles/message)	Very long msg. (cycles/byte)	One-block msg. (cycles/message)	Very long msg. (cycles/byte)
Luffa v1	1,303	33,4	1,211	32.0
Luffa v2	2,157	31.2	1,830	26.2

In addition, the set of instructions for S_2 is identical to that for S_1 except their order. This indicates that the hardware implementation cost and the speed of *Luffa* v2 are expected to be close to that of *Luffa* v1. Besides, an additional blank round disables the very small implementations which process only a block messages. The comparison of hardware performances of *Luffa* v1 and *Luffa* v2 are given in Table 7.

Table 7. Hardware performance of the *Luffa*-256

Optimize	Version	Area [GE]	Frequency [MHz]	# of cycles per round	Throughput [Mbps]	
					One-block msg.	Very long msg.
Area	v1	10,157	100	891	28.7	28.7
	v2	19,646	344	891	49.3	98.7
Throughput	v1	25,833	1,149	9	32,683.0	32,683.0
	v2	30,834	1,124	9	15,980.0	31,960.0

[6] Intel is a registered trademark and Core is a name of the product of Intel Corporation in the U.S. and other countries.

[7] Atmel, AVR, and AVR Studio are registered trademarks of Atmel Corporation in the United States and/or other countries.

References

1. J.P. Aumasson, I. Dinur, W. Meier and A. Shamir "Cube Testers and Key Recovery Attacks On Reduced-Round MD6 and Trivium," *Fast Software Encryption, FSE 2009*, Lecture Notes in Computer Science, vol. 5665, Springer-Verlag, pp. 1–22, 2009.
2. 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`.
3. 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/`.
4. 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/`.
5. L. R. Knudsen, "Truncated and Higher Order Differentials," *Fast Software Encryption, FSE '94*, Lecture Note in Computer Science vol. 1008, pp. 196–211, Springer-Verlag, 1994.
6. T. Yamada and T. Kaneko, Private communication, 2 July 2009.
7. T. Yamada, D. Watanabe, Y. Hatano, and T. Kaneko, "A Higher Order Differential Property of the Non-Linear Permutation in Hash Function Luffa," *Computer Security Symposium 2009 (CSS2009)*, 26-28 October, 2009 (*in Japanese*).
8. D. Watanabe and Y. Hatano, "Higher Order Differential Attack on Reduced Round *Luffa*," to be supplied with the Second Round Package. Also available at `http://www.sdl.hitachi.co.jp/crypto/luffa/HigherOrderDifferentialAttackOnLuffa_v1.pdf`.