Rebound Attack on JH42

María Naya-Plasencia[1*], Deniz Toz[2†], Kerem Varıcı[2†]

[1] FHNW Windisch, Switzerland and University of Versailles, France
maria.naya-plasencia@prism.uvsq.fr
[2] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
{deniz.toz, kerem.varici}@esat.kuleuven.be

Abstract. The hash function JH [20] is one of the five finalists of the NIST SHA-3 hash competition. It has been recently tweaked for the final by increasing its number of rounds from 35.5 to 42. The previously best known results on JH were semi-free-start near-collisions up to 22 rounds using multi-inbound rebound attacks. In this paper we provide a new differential path on 32 rounds. Using this path, we are able to build various semi-free-start internal-state near-collisions and the maximum number of rounds that we achieved is up to 37 rounds on 986 bits. Moreover, we build distinguishers in the full 42-round internal permutation. These are, to our knowledge, the first results faster than generic attack on the full internal permutation of JH42, the finalist version. These distinguishers also apply to the compression function.

Keywords. hash function, rebound attack, JH, cryptanalysis, SHA-3

1 Introduction

A cryptographic hash function is a one way mathematical function that takes a message of arbitrary length as input and produces an output of fixed length, which is commonly called a fingerprint or message digest. Hash functions are fundamental components of many cryptographic applications such as digital signatures, authentication, key derivation, random number generation, etc. So, in terms of security any hash function should be preimage, second-preimage and collision resistant.

*Supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322 and by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014

†This work was sponsored by the Research Fund K.U.Leuven, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the European Commission through the ICT Programme under Contract ICT-2007-216676 (ECRYPT II). The information in this paper is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Most of the recent hash functions use either compression functions or internal permutations as building blocks in their design. In addition to the main properties mentioned above, some ideal properties should also be satisfied for the building blocks. This means that the algorithm should not have any structural weaknesses and should not be distinguishable from a random oracle. The absence of these properties on building blocks may not impact the security claims of the hash function immediately but it helps to point out the potential flaws in the design.

Since many of the hash standards [16,19] have been broken in recent years, the National Institute of Standards and Technology (NIST) announced a competition to replace the current standard SHA-2 with a new algorithm SHA-3. The hash function JH [20], designed by Hongjun Wu, is one of the five finalists of this competition. It is a very simple design and efficient in both software and hardware. JH supports four different hash sizes: 224, 256, 384 and 512-bit. It has been tweaked from the second round to the final round by increasing its number of rounds from 35.5 to 42. The new version is called JH42.

Related Work: We recall here the previously best known results on JH. A marginal preimage attack on the 512-bits hash function with a complexity in time and memory of 2^{507} was presented in [1]. Several multi-inbound rebound attacks were presented in [15], providing in particular a semi-free-start collision for 16 rounds with a complexity of 2^{190} in time and 2^{104} in memory and a semi-free-start near-collision for 22 rounds of compression function with a complexity of 2^{168} in time and 2^{143} in memory. In [12, Sec.4.1], improved complexities for these rebound attacks were provided: 2^{97} in time and memory for the 16 round semi-free-start collision and 2^{96} in time and memory for the 22 rounds semi-free-start near-collision for compression function.

Our Contributions: In this paper we apply, as in [15], a multi-inbound rebound attack, using 6 inbounds that cover rounds from 0 to 32. We first find partial solutions for the differential part of the path by using the ideas from [13]. Due to increased number of rounds compared with the previous attacks, the differential path will have several highly active peaks, instead of one as in [15]. This means that, while in the previous attacks finding the whole solution for the path could be easily done without contradicting any of the already fixed values from the inbounds, now finding the complete solution is the most expensive part. We propose here an algorithm that allows us to find whole solutions for rounds from 4 to 26 with an average complexity of 2^{64}. By repeating the algorithm, the attack can be started from round 0 and extended up to 37 rounds for building semi-free-start near-collisions on the internal state, since we have enough degrees of freedom. Based on the same differential characteristic, we also present distinguishers for 42 rounds of the internal permutation which is the first distinguisher on internal permutation faster than generic attack to the best of our knowledge. We summarize our main results in Table 1.

This paper is organized as follows: In Section 2, we give a brief description of the JH hash function, its properties and an overview of the rebound attack. In

Table 1. Comparison of best attack results on JH (sfs: semi-free-start)

target	rounds	time comp.	memory comp.	attack type	generic comp.	sect.
hash function	16	2^{190}	2^{104}	sfs collision	2^{256}	[15]
hash function	16	$2^{96.1}$	$2^{96.1}$	sfs collision	2^{256}	[12]
comp. function	$19-22$	2^{168}	$2^{143.7}$	sfs near-collision	2^{236}	[15]
comp. function	$19-22$	$2^{95.6}$	$2^{95.6}$	sfs near-collision	2^{236}	[12]
comp. function	26	2^{112}	$2^{57.6}$	sfs near-collision	$2^{341.45}$	§3
comp. function	32	2^{304}	$2^{57.6}$	sfs near-collision	$2^{437.13}$	§3
comp. function	36	2^{352}	$2^{57.6}$	sfs near-collision	$2^{437.13}$	§3
comp. function	37	2^{352}	$2^{57.6}$	sfs near-collision	$2^{396.7}$	§3
internal perm.	42	2^{304}	$2^{57.6}$	distinguisher	2^{705}	§4
internal perm.	42	2^{352}	$2^{57.6}$	distinguisher	2^{762}	§4

Section 3, we first describe the main idea of our attack and then give the semi-free internal near-collision results on the tweaked version JH42. Based on this results, we describe a distinguisher in Section 4 for the full internal permutation, that also applies to the full compression function. Finally, we conclude the paper and summarize our results in Section 5.

2 Preliminaries

2.1 The JH42 Hash Function

The hash function JH is an iterative hash function that accepts message blocks of 512 bits and produces a hash value of 224, 256, 384 and 512 bits. The message is padded to be a multiple of 512 bits. The bit '1' is appended to the end of the message, followed by $384-1+(-l \bmod 512)$ zero bits. Finally, a 128-bit block is appended which is the length of the message, l, represented in big endian form. Note that this scheme guarantees that at least 512 additional bits are padded.

In each iteration, the compression function F_d, given in Figure 1, is used to update the 2^{d+2} bits of the state H_{i-1} as follows:

$$H_i = F_d(H_{i-1}, M_i)$$

where H_{i-1} is the previous chaining value and M_i is the current message block. The compression function F_d is defined as follows:

$$F_d(H_{i-1}, M_i) = E_d(H_{i-1} \oplus (M_i || 0^{2^{d+1}})) \oplus (0^{2^{d+1}} || M_i)$$

Here, E_d is a permutation and is composed of an initial grouping of bits followed by $6(d-1)$ rounds, plus a final degrouping of bits. The grouping operation arranges bits in a way that the input to each S-Box has two bits from the message part and two bits from the chaining value. In each round, the input is

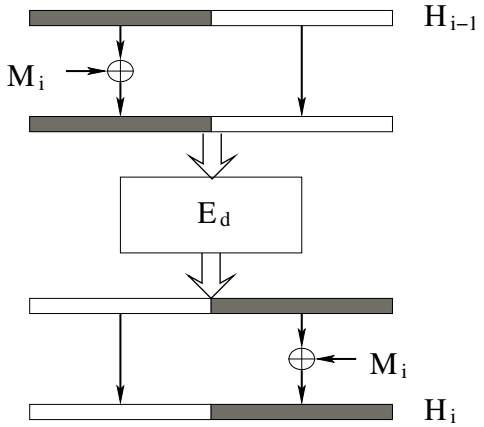

Fig. 1. The compression function F_d that transforms 2^{d+2} bits treated as 2^d words of four bits.

divided into 2^d words and then each word passes through an S-Box. JH uses two 4-bit-to-4-bit S-Boxes ($S0$ and $S1$) and every round constant bit selects which S-Boxes are used. Then two consecutive words pass through the linear transformation L, which is based on a $[4, 2, 3]$ Maximum Distance Separable (MDS) code over $GF(2^4)$. Finally all words are permuted by the permutation P_d. After the degrouping operation each bit returns to its original position.

The initial hash value H_0 is set depending on the message digest size. The first two bytes of H_{-1} are set as the message digest size, and the rest of the bytes of H_{-1} are set as zero. Then, $H_0 = F_d(H_{-1}, 0)$. Finally, the message digest is generated by truncating H_N where N is the number of blocks in the padded message, i.e, the last X bits of H_N are given as the message digest of JH-X where X = 224, 256, 384 and 512.

The official submitted version of JH42 has $d = 8$ and so the number of rounds is 42 and the size of the internal state is 1024 bits. Then, from now on, we will only consider E_8. For a more detailed information we refer to the specification of JH [20].

2.2 Properties of the Linear Transformation L

Since the linear transformation L implements a $[4, 2, 3]$ MDS code, any difference in one of the words of the input (output) will result in a difference in two words of the output (input). For a fixed L transformation, if one tries all possible 2^{16} pairs, the number of pairs satisfying the condition $2 \rightarrow 1$ or $1 \rightarrow 2$ is 3840, which gives a probability of $3840/65536 \approx 2^{-4.09}$. Note that, if the words are arranged in a way that they will be both active this probability increases to $3840/57600 \approx 2^{-3.91}$. For the latter case, if both words remain active ($2 \rightarrow 2$), the probability is $49920/57600 \approx 2^{-0.21}$.

2.3 Observations on the Compression Function

The grouping of bits at the beginning of the compression function assures that the input of every first layer S-Box is xor-ed with two message bits. Similarly, the output of each S-Box is xor-ed with two message bits. Therefore, for a random non-zero 4-bit difference, the probability that this difference is related to a message is $3/15 \approx 2^{-2.32}$.

The bit-slice implementation of F_d uses $d-1$ different round functions. The main difference between these round functions is the permutation function. In each round permutation, the odd bits are swapped by $2^r \bmod (d-1)$ where r is the round number. Therefore, for the same input passing through multiple rounds, the output is identical to the output of the original round function for the $\alpha \cdot (d-1)$-th round where α is any integer.

2.4 The Rebound Attack

The rebound attack was introduced by Mendel et al. [10]. The two main steps of the attack are called inbound phase and outbound phase. In the inbound phase, the available degrees of freedom are used to connect the middle rounds by using the match-in-the-middle technique and in the outbound phase connected truncated differentials are computed in both forward and backward direction.

This attack has been first used for the cryptanalysis of reduced versions of Whirlpool and Grøstl, and then extended to obtain distinguishers for the full Whirlpool compression function [6]. Later, linearized match-in-the-middle and start-from-the-middle techniques are introduced by Mendel et al. [9] to improve the rebound attack. Moreover, a sparse truncated differential path and state is used in the attack on LANE by Matusiewicz et al. [8] rather than using a full active state in the matching part of the attack. Then, these techniques were used to improve the results on AES-based algorithms in the following papers: [2,3,5,11,14,17,18].

3 Semi-free-start internal near-collisions

In this section, we first present an outline for the rebound attack on reduced round versions of JH for all hash sizes. We use a differential characteristic that covers 32 rounds, and apply the start-from-the-middle technique by using six inbound phases with partially active states. We first describe how to solve the multi-inbound phase for the active bytes. Contrary to previous attacks on JH, we now have more fixed values from the inbound phases. So, in order to find a complete solution, we need to merge these fixed values without contradicting any of them. Therefore, we describe next how to match the passive bytes. Finally, we analyze the outbound part.

3.1 Matching the Active Bytes

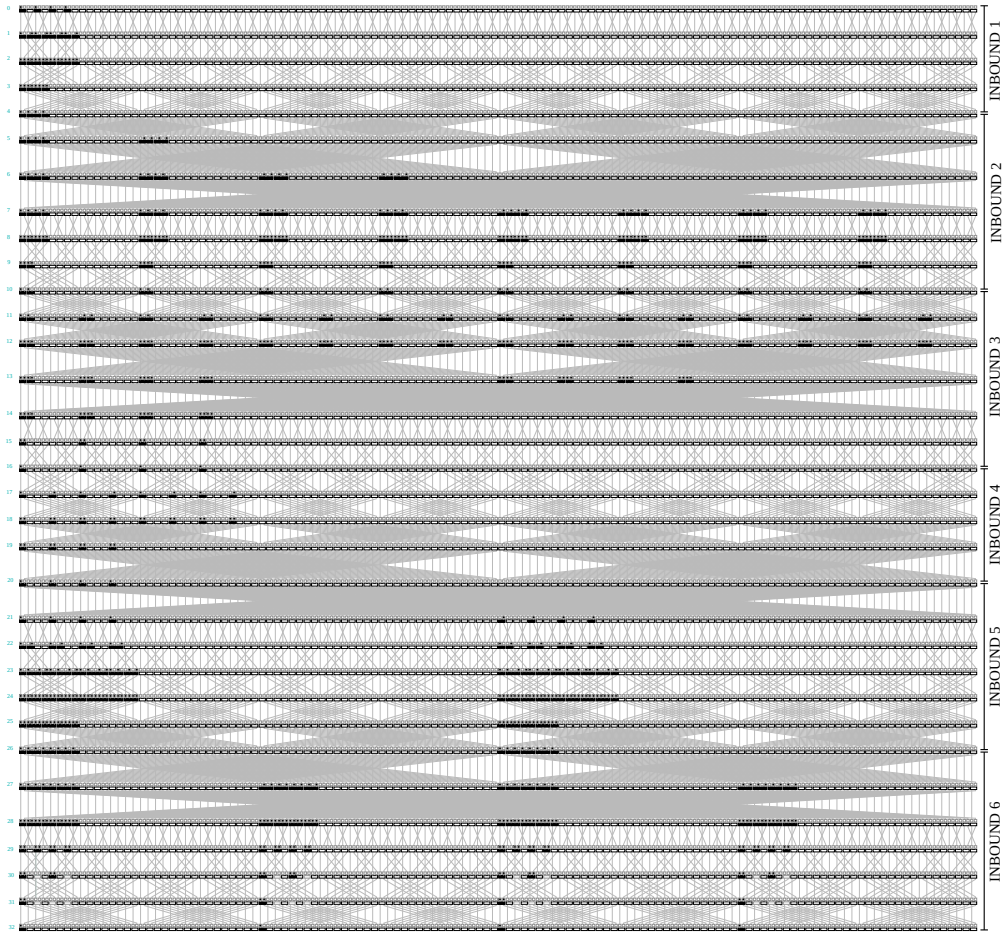

Fig. 2. Differential characteristic for 32 rounds of JH Compression Function (bit-slice representation)

Multi-inbound Phase: The multi-inbound phase of the attack covers 32 rounds and is composed of two parts. In the first part, we apply the start-from-the-middle-technique six times for rounds $0-4$, $4-10$, $10-16$, $16-20$, $20-26$ and $26-32$. In the second part, we connect the resulting active bytes (hence the corresponding state values) by a match-in-the-middle step. The number of active S-Boxes in each of the sets is:

$$4 \leftarrow 8 \leftarrow 16 \rightarrow 8 \rightarrow 4 \tag{1}$$

$$4 \leftarrow 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \tag{2}$$

$$16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \tag{3}$$

$$4 \leftarrow 8 \leftarrow 16 \rightarrow 8 \rightarrow 4 \tag{4}$$

$$4 \leftarrow 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \tag{5}$$

$$16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \tag{6}$$

Here, the arrows represent the direction of the computations for the inbound phases and for a detailed sketch we refer to Figure 2. We start from the middle and then propagate outwards by computing the cross-product[3] of the sets and using the filtering conditions. For each inbound we try all possible 2^{16} pairs in Step 0. The number of sets, the bit length of the middle values (size) of each list, and the number of filtering conditions on words followed by the number of pairs in each set are given in Table 2. The complexities given in the Table 2 are not optimized yet, we will describe the improved complexities later in Section 3.1.

Merging Inbound Phases: The remaining pairs at inbound i are stored on list L_i. Connecting the six lists is performed in three steps as follows:

1. Whenever a pair is obtained from set 2, we check whether it exists in L_3 or not. If it does, another check is done for L_1. Since we have $2^{23.44}$ and $2^{83.96}$ elements in lists 1 and 3 respectively, $2^{83.96}$ pairs passing the second inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{23.44} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{31.36}$. We store these these pairs in list A.

2. Similarly, whenever a pair is obtained from set 5, we check whether it exists in L_6 or not. If it does, another check is done for L_4. Since we have $2^{32.72}$ and $2^{83.96}$ elements in lists 4 and 6 respectively, 2^{80} pairs passing the fifth inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{32.72} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{40.64}$. We store these pairs in list B.

3. Last step is merging these sets A and B. We have $2^{31.36}$ elements in A and $2^{40.64}$ elements in B and 32 bits of condition. Therefore the total expected number of remaining pairs is $2^{31.36} \cdot 2^{-32} \cdot 2^{40.64} = 2^{40}$.

Improving the complexity of finding a solution for the differential part: We have described how to obtain the existing 2^{40} solutions for the differential part. We are going to describe here a better way of doing the inbounds, as proposed in [12, Sec.4.1]. This new technique allows us to reduce the previous complexity from $2^{99.70}$ in time and $2^{83.96}$ in memory to $2^{69.6}$ in time and $2^{67.6}$ in memory. As in our further analysis we will just use one solution (and not 2^{40}) for the differential part, we will adapt the values being able to finally reduce the complexity of this part of the attack to $2^{59.6}$ in time and $2^{57.6}$ in memory. This memory is the memory bottleneck of all the analysis presented in this paper.

1. We consider the six inbounds as described in the previous section, with the difference that, for inbounds $2, 3, 5$ and 6 we will not perform the last step,

[3] cross-product is an operation on two arrays that results in another array whose elements are obtained by combining each element in the first array with every element in the second array.

Table 2. Overview of inbound phases of the attack on 32 rounds of JH

	Step	Size	Sets	Filtering Conditions	Pairs Remaining	Complexity Backwards	Complexity Forwards
Inbound 1	0	8	8	1	$2^{11.91}$	–	2^{16}
	1	16	4	2	2^{16}	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
	4	64	1	4[a]	$2^{23.44}$		
Inbound 2	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.46}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 3	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$
Inbound 4	0	8	8	1	$2^{11.91}$	–	2^{16}
	1	16	4	2	2^{16}	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
Inbound 5	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.26}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 6	0	8	32	1	$2^{11.91}$	–	2^{16}
	1	16	16	2	2^{16}	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.0}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$

[a] Check whether the pairs satisfy the desired input difference

but instead we obtain for each inbound $i \in \{2, 3, 5, 6\}$ two lists $L_{A,i}$ and $L_{B,i}$ as a result, each of size $2^{49.80}$ associated to half of the corresponding differential path. As mentioned before, we are only looking to find one solution for the whole differential path. Then, instead of the $2^{49.80}$ existing solutions for each list, we can consider $2^{44.8}$ elements on each list.

2. First, we merge lists $L_{A,2}$ and $L_{A,3}$. We have 16-bit conditions on values and 16-bit conditions on differences. We obtain a new list $L_{A,23}$ of size

$2^{44.8+44.8-32} = 2^{57.6}$. We do the same with $L_{B,2}$ and $L_{B,3}$ to obtain $L_{B,23}$. Note that this list does not need to be stored, as we can perform the following step whenever an element is found.

3. In order to find a whole solution for the differential part of inbounds 2 and 3, one pair of elements from $L_{A,23}$ and from $L_{B,23}$ still needs to satisfy the following conditions: 32 bits from the parts $L_{A,2}$ and $L_{B,3}$, 32 bits from $L_{B,2}$ and $L_{A,3}$, 3.91×4 from the step 5 of inbound 2 that we have not yet verified and 3.91×4 from step 5 of inbound 3 that is not yet verified either. Therefore, we have 95.28-bit conditions in total to merge $L_{A,23}$ and $L_{B,23}$. For each element in $L_{B,23}$ we can check with constant cost if the corresponding element appears in $L_{A,23}$ (it can be done by a lookup in a table, representing the differential transitions of L and next by a lookup in the list $L_{A,23}$ to see if the wanted elements appear. See [13,12] and Figure 3 for more details). When we find a good pair, we store it in the list L_{23} that has a size of about $2^{19.92}$ elements satisfying the differential part of rounds from 4 to 16. The cost of this step is then $2^{57.6+1}$ in time and $2^{59.6}$ in memory.
4. Do the same with inbounds 5 and 6, to obtain list L_{56} of size $2^{19.92}$, with a cost of $2^{57.6+1}$ in time and $2^{57.6}$ in memory.
5. Merge the solutions obtained in the first inbound with the ones in L_{23}, obtaining a new set L_{123} of size $2^{19.92+23.44-32} = 2^{11.36}$.
6. Merge the solutions obtained from step 4 with list L_{56} obtaining a new one, L_{456} of size $2^{19.92+32.72-32} = 2^{20.64}$.
7. Finally, merging L_{123} and L_{456} gives $2^{11.36+20.64-32} = 1$ partial solution for the differential part of the path from round 0 to round 32.

The complexity of obtaining one partial solution for rounds from 0 to 32 is dominated by Steps $2-4$ of the algorithm. As a result, the complexity of matching the active bytes becomes $2^{59.6}$ in time and $2^{57.6}$ in memory.

3.2 Matching The Passive Bytes

In Figure 4, colored boxes denote the S-boxes whose values have already been fixed from the inbound phases. Note that, we have not treated the passive bits yet (i.e., found the remaining values that would complete the path). We will propose a way of finding 2^{32} solutions that verify the path from rounds 4 to 26 with time complexity 2^{96} and memory complexity $2^{51.58}$. This can be done in three steps as follows:

1. (Rounds 10 to 14): The sets of groups of 8 bits denoted by a, b, c, d, e, f in round 14 are independent of each other in this part of the path. In round 10, 32 bits are already fixed for each of these sets (groups of 4 bits denoted by A, B, C, D, E, F). By using all possible values of the remaining 96 passive bits (32 bits not fixed from A, B, C, D, E, F plus 64 from the remaining state at round 10), we can easily compute a list of 2^{96} elements with cost 2^{96} that satisfy the 32 bit conditions for each of the groups.

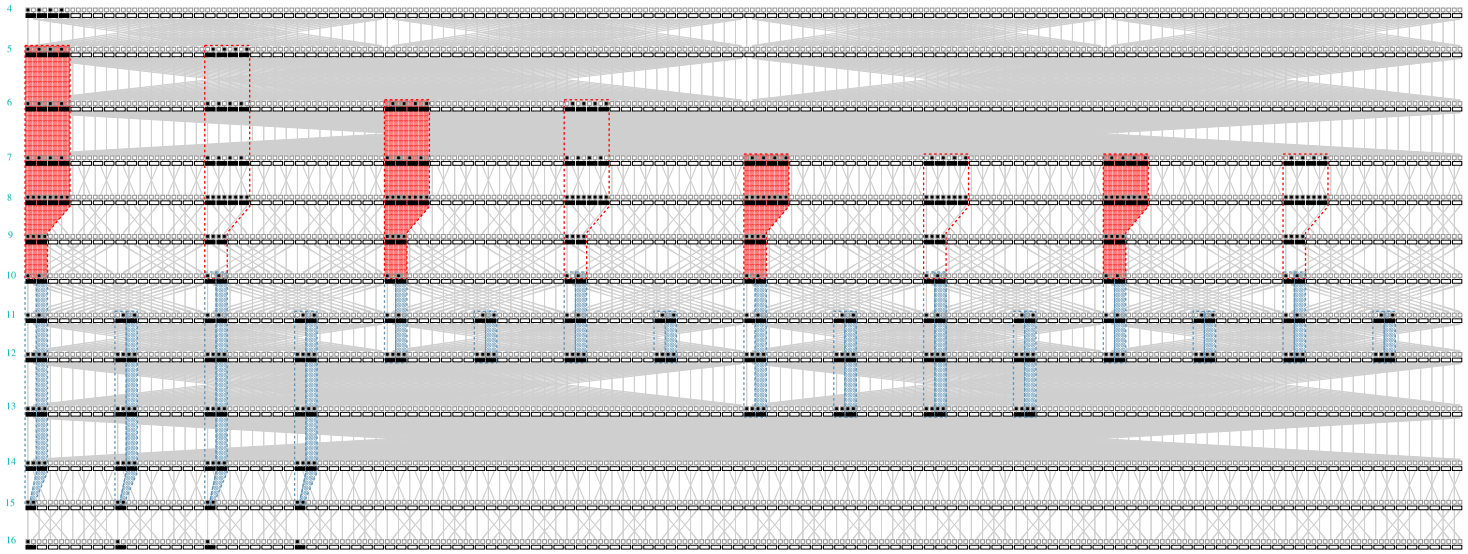

Fig. 3. Improving the complexity of finding a solution for the differential part for rounds $4-16$: Red boxes (above) denote the sets $L_{A,2}$ (filled) and $L_{B,2}$, blue boxes (below) denote the sets $L_{A,3}$ and $L_{B,3}$ (filled).

2. (Rounds 14 to 20): In round 20, we have 256 bits (green S-boxes ■) whose values are fixed from the solutions of the second inbound phase. We can divide the state in round 19 (until the state in round 14) in 4 independent parts (m, n, o, p). In Figure 4, the fixed bits coming from round 20 are denoted by green lines and the ones of the first inbound phase are denoted in blue "■". Note that the three parts m, n, o are identical, while p is different since there are some differences and some additional fixed values in it.

   We fix the parts m and n to some values that satisfy all the conditions of the fixed bits in rounds 19 and 14. This can be done as follows: Similar to what we have done in step 1, we can divide the state of rounds $16 - 19$ (for each part separately) into four groups (x, y, z, u) such that they are independent of each other when computing forwards.

   In round 16, each group has 16 bits whose values have already been fixed and 48 bits of freedom. We see that each group affects only one fourth of the green lines (16 bits in total) in round 19. Therefore, there exist $2^{48-16} = 2^{32}$ possibilities for each group x, y, z, u but we just need one. This one can then be found with a cost of about 2^{16}

3. (Merging) Each of the sets $L_a, \ldots, L_f$ has 2^{96} possible values from step 1, and fixing m and n fixes 64 bits for each of them in round 14. This gives us in average $2^{96-64} = 2^{32}$ possible values for each set in the half of the state associated to o and p in round 14.

   For the part p we use the same idea explained in step 2. Group x is completely fixed due to the differential characteristic, and only the groups y, z, u have freedom, so there exists $(2^{32})^3 = 2^{96}$ possibilities. For each possibility, we compute the part of state in round 14 associated to p. We have 32 bits of condition for each of lists, and in average 2^{32} values are associated to each list. Thus, for each of the computed values, we will have only one remaining element that will determine the values at positions $a - f$ in the part o.

   Now, we have 2^{96} possible o values. The probability that a fixed value verifies the conditions of o in round 19 is $(2^{-4})^{16} = 2^{-64}$. Therefore, we obtain $2^{96-64} = 2^{32}$ solutions that verify the whole path from round 4 to round 26 with a complexity in time of 2^{96}.

Note that we do not need to store the lists $L_a, \ldots, L_f$ of elements from round 14 each of size 2^{96} but we can instead store for each of them two lists of size 2^{48} corresponding to the upper and down halves of the corresponding groups in state 13. Then, when fixing a value of m and n we can check with a cost of 2^{32} which will be the list of 2^{32} values for o and p that we obtained in step 3. Finally, we have obtained 2^{32} complete solutions for the path from 4 to 26 with a cost of 2^{96} in time, and $6 \cdot 2 \cdot 2^{48} \approx 2^{51.58}$ in memory.

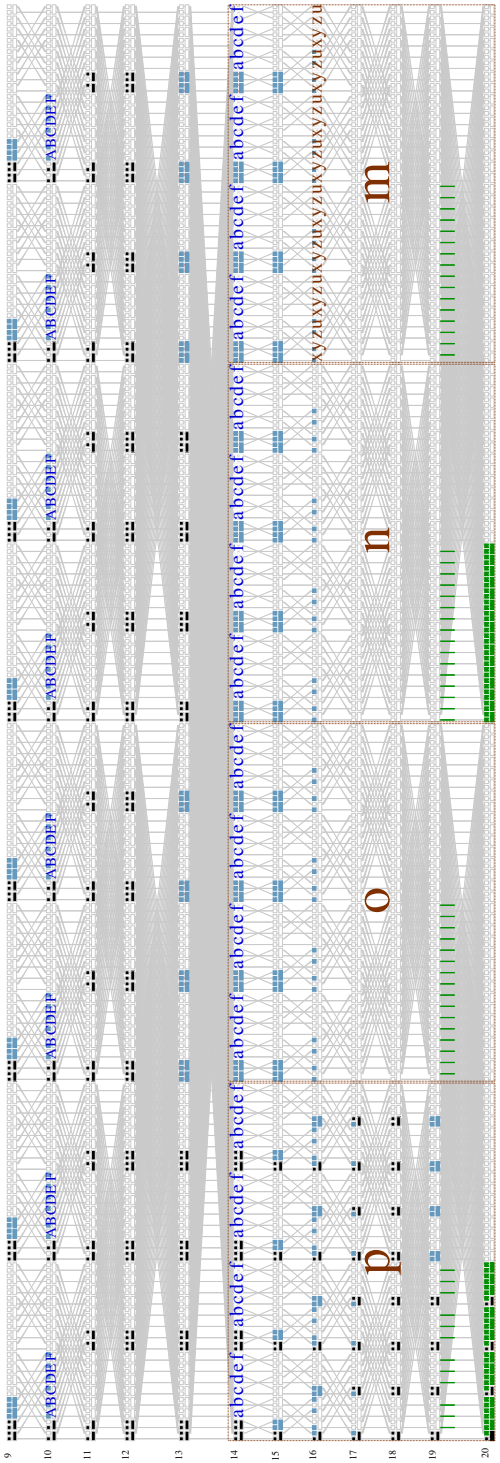

Fig. 4. Matching the passive bytes (bit-slice representation)

Semi-free-start near-collisions up to 32 rounds: Up to now, we have found solutions for the passive bytes from rounds $4-26$. If we want a solution for the path from round 0 to round 26, we will have to repeat the previous procedure of matching the passive bytes 2^{16} times (as the probability of passing from round 0 to 4 is 2^{-48} and we have 2^{32} pairs). Then, we can find a solution for rounds $0-26$ with complexity 2^{112} in time. In order to extend this result to 32 rounds, we have to repeat the previous procedure 2^{192} times (since we have 64 and 128 bits of condition from rounds 26 and 27 respectively). Therefore, the complexity for finding a complete solutions for rounds from 0 to 32 is $2^{112} \cdot 2^{192} = 2^{304}$ in time.

Note that, we still have enough degrees of freedom. In step 1, we started with 768 bits (128×6 from the groups $a-f$) in round 14 and matched 192 bits (32×6 for $A-F$) in round 10. In Step 2, we have 48 bits in round 16 coming from the fourth inbound phase and we matched another 240 bits from the fifth inbound phase in round 19. So in total we have $768-192-48-240 = 288$ bits of degrees of freedom remaining.

3.3 Outbound Phase:

The outbound phase of the attack is composed of 5 rounds in the forward direction. A detailed schema of this trail is shown in Figure 5 in appendix, and for the pairs that satisfy the inbound phase, we expect to see the following differential trail in the outbound phase:

$$\text{Inbound Phase} \ \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 8$$

Semi-free-start near-collisions up to 37 rounds: For 32 rounds of the JH compression function, we obtain a semi-free-start near-collision for 1002 bits. We can simply increase the number of rounds by proceeding forwards in the outbound phase. Note that, we have an additional probability of $2^{-32} \times 2^{-16}$ coming from the eight filtering conditions in round 34 and the four filtering conditions in round 35. Thus, the complexity of the active part of the attack remains the same: $2^{59.6}$ in time and $2^{57.6}$ in memory. This is the case as one solution for the differential part is enough for the attack, as it will have different values at the bits with conditions in the outbound part when the passive part is modified. The complexity of the passive part becomes $2^{304} \cdot 2^{48} = 2^{352}$ in time and $2^{51.58}$ in memory.

The details can be seen in Table 3. We also take into account the colliding bits that we obtain at the output of the compression function after the final degrouping with the differences from the message.

Table 3. Comparison of complexity of the generic attack for near-collisions and our results

#Rounds	# Colliding bits	Generic Attack Complexity	Our Results
23	892	$2^{230.51}$	$2^{59.6}$
$24-26$	762	$2^{99.18}$	$2^{59.6}$ [a]
26	960	$2^{341.45}$	2^{112}
27	896	$2^{236.06}$	2^{112}
32	1002	$2^{437.12}$	2^{304}
33	986	$2^{396.77}$	2^{304}
34	954	$2^{329.97}$	2^{304}
35	986	$2^{396.77}$	2^{336}
36	1002	$2^{437.12}$	2^{352}
37	986	$2^{396.77}$	2^{352}
38	928	$2^{284.45}$	2^{352}

[a] Obtained directly from the solutions of the active part, without need of matching the passive bits

4 Distinguishers on JH

Indifferentiability is considered to be a desirable property of any secure hash function design. Moreover, for many of the designs, the indifferentiability proofs for the mode of operation are based on the assumption that the underlying permutation (function) is ideal (i.e., random permutation). This is the case of the indifferentiability proof of JH [1], that supposes that E_d is a random permutation.

In this section, we present a distinguisher for E_8 showing that it is distinguishable form a random permutation. Using the differential path that we presented in the previous section, we can build the distinguishers on the full 42 rounds of the internal permutation E_8 with no additional complexity. As a result of our distinguisher, the proof from [1] does not apply to JH as the assumption of E_8 behaving like random does not hold. Next, we explain how these distinguishers on the internal permutation can be easily extended to distinguishers on the compression function.

There exists also a known trivial distinguisher on the construction of the compression function of JH: If the chaining value has a difference that can be cancelled by the message block, then the output will have a difference directly related to the one coming from the message block. This implies that both the message and the chaining values have differences. Contrary to the trivial one, our compression function distinguisher exploits the properties of the internal permutation and only needs differences in the message or in the chaining value.

4.1 Distinguishers on the reduced round internal permutation

Let us remark here briefly that if we find solutions for rounds 4 to 20, and then let them spread freely backward (difference in 64 bits) and forward (difference in 256 bits), we can obtain a distinguisher for 26 rounds with a much lower complexity: $2^{59.6}$ in time and $2^{57.6}$ in memory (the cost of the differential part). As in this paper the aim is reaching a higher number of rounds, we do not go further into the details.

4.2 Distinguishers on the full internal permutation

In the previous sections we showed that a solution for 37 rounds can be obtained with a time complexity of 2^{352} in time and $2^{57.6}$ in memory. In Figure 5 from the appendix, we see how these active words diffuse to the state after 42 rounds with probability one. Therefore, before the degrouping operation we have 64 active and 192 passive words in the state. The number of active and passive bits still remain the same after the degrouping operation. It is important to remark that the positions of the active bits are fixed, also after the degrouping operation.

We can then build a distinguisher that will distinguish the 42-round permutation E_8 from a random permutation using this path. This distinguisher aims at finding a pair of input states (A, A') such that $E_8(A) \oplus E_8(A')$ collide in the 768 bits mentioned above. Let $A \oplus A' = \Delta_1$ correspond to the input difference of the differential path, then $|\Delta_1| = 8$ bits. Similarly, let $B = E_8(A)$ and $B' = E_8(A')$, then the output difference is $B \oplus B' = \Delta_2$ where $|\Delta_2| = 256$.

In the case of a random function, we calculate the complexity of such a distinguisher as follows: We fix the values of the passive bits in the input; but not the ones of the active bits. Then, we have $2^{|\Delta_1|}$ possibilities for the values from the active bits. We compute the output of E_8 for each one of these values and store them in a list. From this list we can obtain $\binom{2^{|\Delta_1|}}{2}$ pairs with the given input difference pattern. The probability of satisfying the desired output difference pattern is $2^{|\Delta_2|-1024}$ for each pair, so we repeat the procedure with a new value for the input passive bits until we find a solution. The time complexity of finding such an input pair will be:

$$\frac{2^{|\Delta_1|}}{2^{(|\Delta_1|-1)} \cdot (2^{|\Delta_1|} - 1) \cdot 2^{|\Delta_2|-1024}} = 2^{761}.$$

Instead, in our case the complexity of finding such an input pair is the complexity of finding a solution for the path, that is 2^{352} in time and $2^{57.6}$ in memory.

Another distinguisher of E_8 can be built if we consider the scenario where the differential path for rounds $0-4$ does not need to be verified, i.e., $|\Delta_1| = 64$. In this case, we consider that from round 4 to 0 we obtain the differences that propagate with probability one. Therefore, the matching of the passive part does not need to be repeated 2^{208} times but only 2^{160} (as we do not need 2^{48} extra repetitions for verifying rounds 0 to 4). The complexity of this distinguisher will then be 2^{304}, and provides a pair of inputs A and A' that produce an output

with 768 colliding bits as the ones represented in Figure 5 from appendix. The complexity of such a generic distinguisher would be $\frac{2^{64}}{(2^{64}-1)\cdot 2^{63-768}} = 2^{705}$, while in our case is 2^{304} in time and $2^{57.6}$ in memory.

4.3 Distinguishers on the full compression function

We should emphasize that our distinguishers on E_8 can be easily converted to a distinguisher on the full compression function of JH42. We only need to xor this message difference to the output of E_8 as specified.

For our first distinguisher, the input difference is already arranged such that we only have difference in the message. These active bits coming from the message coincide with the active bits in the output at the xor operation. As a result, we have the same 768 passive bits. The same applies for our second distinguisher when we have differences only in the chaining value.

5 Conclusion

In this paper, we have presented semi-free-start internal near-collisions up to 37 rounds by using rebound attack techniques. We first obtained a 960-bit semi-free-start near-collision for 26 rounds of the JH compression function with a time complexity of 2^{112} and a memory complexity of $2^{57.6}$. We then extended this to 986-bit semi-free-start near-collision for 37 rounds by repeating the algorithm. Time complexity of the attack is increased to 2^{352} and the memory complexity remains the same. We also presented semi-free-start near-collision results for intermediate rounds $26-37$ in Table 3. Our findings are summarized in Table 1.

Even more, we have presented distinguishers on the full 42 rounds of the internal permutation E_8 of the tweaked SHA-3 finalist JH. The best distinguisher has a time complexity of 2^{304} in time and $2^{57.6}$ in memory and provides solutions for the differential path on the 42 rounds. Obtaining such a pair of inputs producing a same truncated differential in the output for a random function would cost 2^{705} in time. Our internal permutation distinguishers can easily be extended to compression function distinguishers with the same complexity.

Although our results do not present a threat to the security of the JH hash function, they invalidate the JH indifferentiability proof presented in [1].

References

1. Bhattacharyya, R., Mandal, A., Nandi, M.: Security Analysis of the Mode of JH Hash Function. In: Hong and Iwata [4], pp. 168–191
2. Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.): Information Security - 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6531. Springer (2011)
3. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. In: Hong and Iwata [4], pp. 365–383

4. Hong, S., Iwata, T. (eds.): Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6147. Springer (2010)
5. Ideguchi, K., Tischhauser, E., Preneel, B.: Improved collision attacks on the reduced-round grøstl hash function. In: Burmester et al. [2], pp. 1–16
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui [7], pp. 126–143
7. Matsui, M. (ed.): Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5912. Springer (2009)
8. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In: Matsui [7], pp. 106–125
9. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jr., M.J.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, 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. Lecture Notes in Computer Science, vol. 5665, pp. 260–276. Springer (2009)
11. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attacks on the reduced grøstl hash function. In: Pieprzyk, J. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 5985, pp. 350–365. Springer (2010)
12. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010), http://eprint.iacr.org/2010/607.pdf, (extended version). urlhttp://eprint.iacr.org/
13. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Crypology: CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 188–205. Springer (2011)
14. Peyrin, T.: Improved differential attacks for echo and grøstl. In: Rabin, T. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6223, pp. 370–392. Springer (2010)
15. Rijmen, V., Toz, D., Varici, K.: Rebound Attack on Reduced-Round Versions of JH. In: Hong and Iwata [4], pp. 286–303
16. Rivest, R.L.: The MD5 Message-Digest Algorithm. RFC 1321 (1992), http://www.ietf.org/rfc/rfc1321.txt, http://www.ietf.org/rfc/rfc1321.txt
17. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-sbox analysis: Applications to echo and grøstl. In: Abe, M. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 38–55. Springer (2010)
18. Schläffer, M.: Subspace distinguisher for 5/8 rounds of the echo-256 hash function. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 369–387. Springer (2010)
19. of Standards, N.I., Technology: FIPS 180-1:Secure Hash Standard (1995), http://csrc.nist.gov, http://csrc.nist.gov
20. Wu, H.: The Hash Function JH. Submission to NIST (2008), http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf, http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf

A Outbound Phase Figure

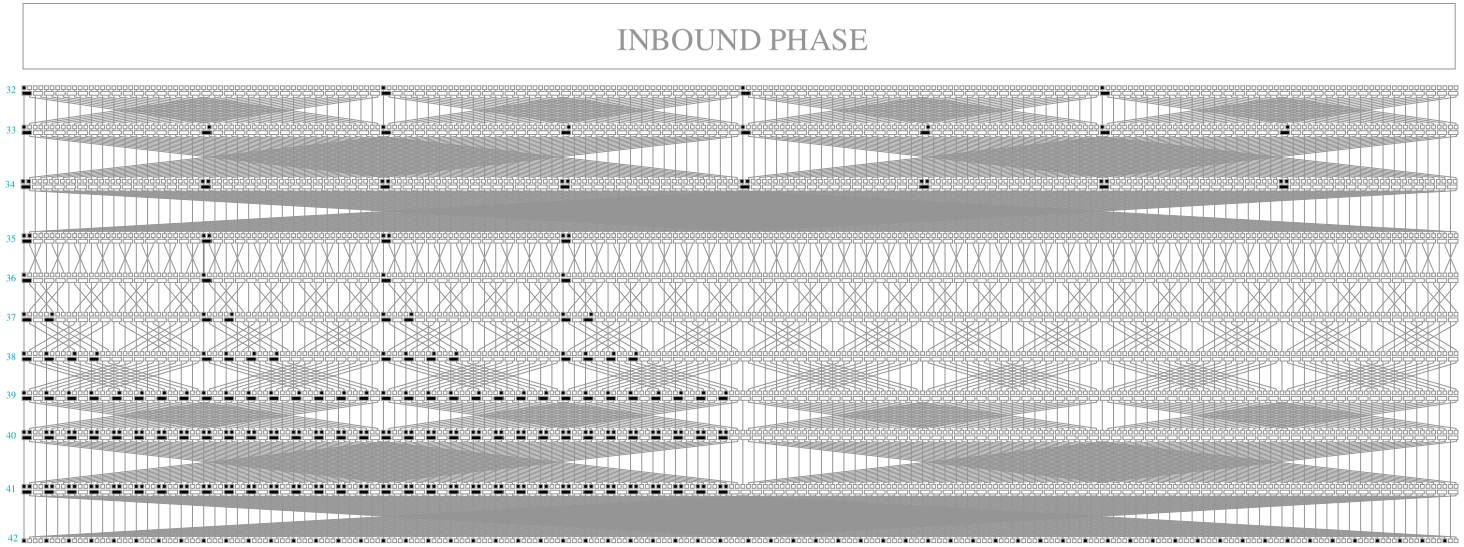

Fig. 5. Differential characteristic for the outbound phase of JH Compression Function (bit-slice representation)