New Non-Ideal Properties of AES-Based Permutations: Applications to ECHO and Grøstl

Yu Sasaki[1], Yang Li[2], Lei Wang[2], Kazuo Sakiyama[2], and Kazuo Ohta[2]

[1] NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11 Midoricho, Musashino-shi, Tokyo 180-8585 Japan
sasaki.yu@lab.ntt.co.jp
[2] The University of Electro-Communications
1-5-1 Choufugaoka, Choufu-shi, Tokyo, 182-8585 Japan
{liyang,wanglei,saki,ota}@ice.uec.ac.jp

Abstract. In this paper, we present *non-full-active Super-Sbox analysis* which can detect non-ideal properties of a class of AES-based permutations with a low complexity. We apply this framework to SHA-3 round-2 candidates ECHO and Grøstl. The first application is for the full-round (8-round) ECHO permutation, which is a building block for 256-bit and 224-bit output sizes. By combining several observations specific to ECHO, our attack detects a non-ideal property with a time complexity of 2^{182} and 2^{37} amount of memory. The complexity, especially in terms of the product of time and memory, is drastically reduced from the previous best attack which required $2^{512} \times 2^{512}$. To the best of our knowledge, this is the first result on the full-round ECHO permutation with both time and memory below 2^{256} or 2^{224}. Note that this result does not impact the security of the ECHO compression function nor the overall hash function. We also show that our method can detect non-ideal properties of the 8-round Grøstl-256 permutation with a practical complexity, and finally show that our approach leads to an improvement on a semi-free-start collision attack on the 7-round Grøstl-512 compression function. Our approach is based on a series of attacks on AES-based hash functions such as rebound attack and Super-Sbox analysis. The core idea is using a new differential path consisting of only non-full-active states.

Keywords: AES-based permutation, ECHO, Grøstl, SHA-3, non-full-active Super-Sbox analysis

1 Introduction

In the SHA-3 competition [1], 14 algorithms are being considered as round 2 candidates. At the present time, none of them has been seriously broken in terms of the important security properties of hash functions such as collision resistance or preimage resistance. However, regarding some candidates, building blocks such as compression functions or internal permutations have been shown that they do not satisfy ideal properties. Although it does not damage the security of hash functions immediately, the analysis against building blocks are useful to know the potential weakness, security margin, validity of the security proof, potential improvement in the future, and so on.

Many of the SHA-3 candidates are based on the design strategy of AES [2, 3]. Recently, an outstanding progress in the cryptanalysis against AES-based hash functions or permutations has been made [4–14]. Specifically, *Rebound attack* proposed by Mendel *et al.* at FSE 2009 [5], *Start-from-the-Middle attack* proposed by Mendel *et al.* at SAC 2009 [6], and *Super-Sbox analysis* applied to the rebound attack by Lamberger *et al.* at Asiacrypt 2009 [13] and by Gilbert and Peyrin at FSE 2010 [7] have wide range of their applications and are powerful analytic tools. In fact, the rebound based attack has been applied to several SHA-3-candidates [5–12, 15] such as Grøstl [16], ECHO [17], JH [18], Cheetah [19], LANE [20], Twister [21]. It has also been applied to other hash functions [5–7, 13, 14] such as Whirlpool [22] and AES hashing modes.

ECHO [17] is one of the round 2 algorithms in the SHA-3 competition, which was designed by Benadjila *et al.* It has an AES-based structure and its building block is a 2048-bit AES-based permutation. The number of rounds in the permutation is 8 for ECHO-224 and ECHO-256, and 10 for ECHO-384 and ECHO-512. At FSE

2010, Gilbert and Peyrin showed that the full-round (8-round) ECHO permutation could be distinguished from an ideal permutation with time of 2^{768} and memory of 2^{512} by using the Super-Sbox analysis [7]. After that, Peyrin [23] reduced the complexity to 2^{512} in both time and memory. Hence, the 8-round ECHO permutation was shown to be non-ideal as a 2048-bit permutation. However, because the 8-round ECHO permutation is a building block to generate 256-bit or 224-bit hash values and compression part from 2048-bits to 256-bits or 224-bits is not considered, the impact of this attack seems almost negligible. In addition, as long as it is evaluated by the Super-Sbox analysis with the framework of [7], the time or memory cannot be below 2^{512}. In fact, Gilbert and Peyrin claimed as follows [7, Section 4.4]:

> "Moreover, since the Super-Sbox cryptanalysis of the ECHO permutation presented above requires at least 2^{512} computations and memory, it is not a well suited starting point for trying to mount a distinguisher or a collision search attack against one of the compression functions of ECHO-256 or ECHO-512 (or one of their single-pipe variants)."

Besides, Peyrin mentioned attacks on the 8-round ECHO permutation as follows [23, Appendix B]:

> "Indeed, the improved Super-Sbox attack for ECHO can not be used anymore as too many active cells are present in the middle rounds. One has to use the Super-Sbox method instead, which is inefficient in the case of ECHO since it requires at least 2^{512} computations and memory."

To sum up, there is no powerful analysis on the ECHO hash function nor compression function. Even though attacks on the permutation reached full-round with Super-Sbox analysis, the complexity is too high due to the property of the Super-Sbox analysis.

Note that besides the ECHO internal permutation, the round-reduced ECHO compression function is attack by Peyrin [23, 24]. Recently, after the submission of the Second SHA-3 Candidate Conference, Schläffer attacked the round-reduced ECHO hash function and improved the attacks on the compression function [25]. These results are listed in Table 1.

Our Contributions

In this paper, we present *non-full-active Super-Sbox analysis* which can detect non-ideal properties of a class of AES-based permutations with a low complexity. To demonstrate its applicability, we first apply the non-full-active Super-Sbox analysis to the 8-round Grøstl-256 permutation, which is an AES-based permutation consisting of the 8×8 state. This attack can detect a non-ideal property of the 8-round Grøstl-256 permutation with time of 2^{48} and memory of 2^{8}, while detecting the same property of an ideal permutation requires 2^{96}. We then apply this framework to the full-round (8-round) ECHO permutation by optimizing the attack with taking several properties specific to ECHO into account. This attack can detect a non-ideal property of the 8-round ECHO permutation with time of 2^{182} and memory of 2^{37}, while detecting the same property of an ideal permutation requires 2^{256}. Note that the 8-round ECHO permutation is a building block for ECHO-256 and ECHO-224. As far as we know, this is the first result on the full-round ECHO permutation which can work with both time and memory (or product of these factors) below 2^{256} (or 2^{224}). Note, however, that the role of the convolution in the ECHO compression function is very important for its security and our distinguisher cannot be extended to the ECHO compression function, nor the hash function. Finally, we show that our approach also improves the amount of memory for the semi-free-start collision attack on the 7-round Grøstl-512 compression function to 2^{56} from 2^{64}. The attack results are summarized in Table 1. The technical details in this paper are as follows.

Low complexity distinguishers on AES-based permutations We present a new strategy of the Super-Sbox analysis which can work for a class of AES-based permutations in generic. The core idea is using a differential path whose inbound part, in particular inside the Super-Sbox, consists of only non-full-active states. Regarding non-active bytes, the difference is always 0 through the SubBytes and InverseSubBytes operations regardless of its value. Hence, attackers can freely choose the value without breaking the differential path. This freedom degrees enable attackers to control values (or differences through the SubBytes operation) of other bytes inside the Super-Sbox to be connected efficiently.

Table 1. Comparison of attack results on ECHO and on Grøstl.

Target	Rounds	Time	Memory	Attack Type	Paper
ECHO-256/-224	8 (full)	2^{768}	2^{512}	Distinguisher	[7]
Permutation	8 (full)	2^{512}	2^{512}	Distinguisher	[23]
	8 (full)	2^{182}	2^{37}	Distinguisher	Ours Section 5.2
Grøstl-256	8	2^{112}	2^{64}	Distinguisher	[7]
Permutation	8	2^{48}	2^{8}	Distinguisher	Ours Section 4.4
Grøstl-512	7	2^{152}	2^{64}	Semi-free-start collision	[15]
Comp. Function	7	2^{152}	2^{56}	Semi-free-start collision	Ours Section 5.3
ECHO-256	4	2^{64}	2^{64}	Collision	[25]
Hash Function	5	2^{96}	2^{64}	Distinguisher	[25]
ECHO-256	3	2^{64}	2^{64}	Semi-free-start collision	[24]/[23]
Comp. Function	6.5	2^{96}	2^{64}	Free-start-near collision	[25]
	7	2^{107}	2^{64}	Distinguisher	[25]
ECHO-512	3	2^{96}	2^{64}	Semi-free-start collision	[24]/[23]
Comp. Function	6.5	2^{96}	2^{64}	Free-start-near collision	[25]
	7	2^{106}	2^{64}	Distinguisher	[25]
Grøstl-256 Comp. Func.	10 (full)	2^{192}	2^{64}	Distinguisher	[24]/[23]
Grøstl-512 Comp. Func.	11	2^{640}	2^{64}	Distinguisher	[24]/[23]

Observations on the property of ECHO permutation We explain two new observations on the ECHO permutation when dealing with the byte-wise truncated differential path. First, we find that the linearity of the jointed two linear operations (MixColumns inside the BigSB and the following BigMC) should be taken into account in order to correctly calculate the complexity for a certain differential path. Second, there are freedom of the differential paths inside BigSB available to attackers to reduce the complexity.

This paper is organized as follows. In Section 2, we briefly describe AES-permutation, ECHO, and Grøstl. In Section 3, we introduce previous work. In Section 4, we present the framework of non-full-active Super-Sbox analysis. We show its application to the 8-round Grøstl-256 permutation as an example. In Section 5, we apply our attack to the full-round ECHO permutation with taking several properties specific to ECHO into account. We also apply our attack to the semi-free-start collision attack on the 7-round Grøstl-512 compression function. In Section 6, we conclude this paper.

2 Specifications

In 2000, the block cipher Rijndael designed by Daemen and Rijmen was selected as Advanced Encryption Standard (AES) [2, 3]. AES as a block cipher is with a 128-bit block represented by a 4×4 state where each element is a byte. Here we consider a general AES-based permutation with the internal state represented by an $r \times r$ matrix where each element in it is a c-bit word. The row and column position of a word/byte is denoted by i and j, respectively where $i, j \in [0, r-1]$. As shown in Fig. 1, the state is updated by four operations in a round of the AES-based permutation.

- SubBytes (SB) as the non-linear layer substitute each word/byte of the internal state according to an S-box table.
- In ShiftRows (SR), each word/byte at row j of the state is cyclically shifted to left by j positions.
- The linear operation MixColumns (MC) multiply each column of the state by a Maximum-Distance Separable (MDS) matrix .
- The AddRoundKey (AK) operation performs the bit-wise exclusive-or operation between the current state with a constant.

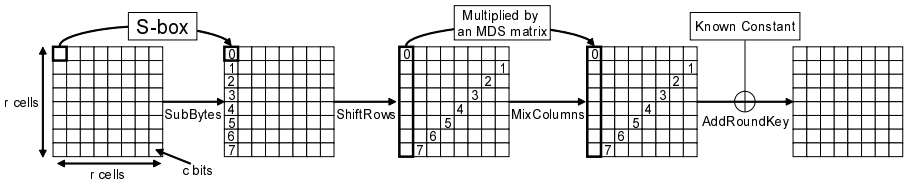


Fig. 1. The operations inside a round of AES-based permutation.

2.1 ECHO Permutation

The ECHO hash function [17] designed by Benadjila *et al.* is a candidate in the second round of the SHA-3 competition. It has an AES-based structure and its building block is a 2048-bit AES-based permutation. The number of rounds in the permutation is 8 for ECHO-224 and ECHO-256, and 10 for ECHO-384 and ECHO-512. The 2048-bit internal state of ECHO can be represented by a 4×4 matrix where each element in it is a 128-bit AES state called a BigWord. The round operations in the ECHO permutation manipulate 128-bit BigWords instead of 8-bit bytes. As shown in Fig. 2, one round of ECHO permutation has three operations as follows.

- BigSB as the non-linear layer substitute each 128-bit BigWord of the ECHO state by applying 2-round AES permutation on it.
- In BigSR, each 128-bit BigWord at row j is cyclically shifted to left by j positions.
- The linear operation BigMC multiply each 4 bytes of the ECHO state by a MDS matrix.

To simplify the dedicated analysis of the ECHO permutation, as introduced by [24], we denote 4 types of byte-wise truncated differences of the BigWord as shown in Fig. 3, where active bytes are in grey.

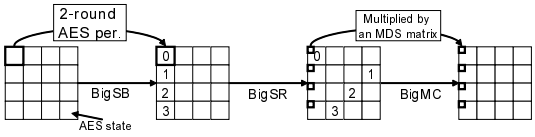


Fig. 2. One round of ECHO permutation.

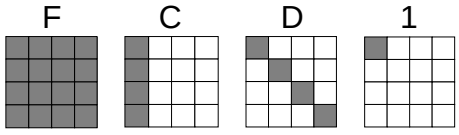


Fig. 3. Notations for BigWords of ECHO.

2.2 Grøstl Permutation and Compression Function

The Grøstl hash function [16] designed by Gauravaram *et al.* is another candidate in the second round of the SHA-3 competition. The compression function of Grøstl is built upon the AES-based permutations as well. In the case of Grøstl-256 permutation, the internal state can be represented by an 8×8 matrix where each element is an 8-bit byte. While the internal state of Grøstl-512 permutation is an 8×16 matrix of bytes. The number of rounds in the permutation is 10 for Grøstl-224 and Grøstl-256, and 14 for Grøstl-384 and Grøstl-512.

The Grøstl-256 permutation are the AES-based permutation with $r = 8$ and $c = 8$. As shown in Fig. 4, The Grøstl-512 permutation is different from Grøstl-256 with respect to only the ShiftRows operation, where the bytes at row 7 are cyclically shifted to left by 11 positions. The compression function is based on two AES-based permutations P and Q, where only the constant numbers for AddRoundKey are different. As described in Fig. 5, the Grøstl compression function takes a chaining variable CV and a message M as the inputs. The output of the compression function is the new chaining variable CV' which is the XOR of the outputs of permutations P, Q and the original chaining variable CV.

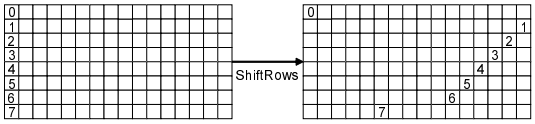


Fig. 4. ShiftRows operation for Grøstl-512.

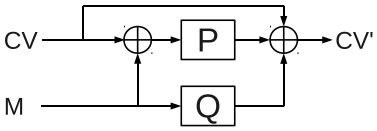


Fig. 5. Compression function of Grøstl.

3 Previous Work

Discovering non-ideal properties are valuable when evaluating the security of certain hash functions. This section introduces recently proposed analytic tools that are applied to explore the non-ideal properties of the internal permutations used in certain hash functions.

Rebound attack. Rebound attack was first proposed by Mendel *et al.* at FSE 2009 [5]. It is a very useful technique to analyze the AES-based permutations under a known-key assumption. Rebound attack divides a differential path into two parts as inbound and outbound. For the inbound phase, attackers can control the most expensive part of the differential path with a very low average complexity. Then outbound differential path is satisfied probabilistically. Rebound attack needs to make sure the total number of starting points generated at the inbound phase is enough to fulfill the outbound complexity. For more details, we refer to the original publication [5].

Start-from-the-Middle attack. Start-from-the-Middle attack was proposed by Mendel *et al.* at SAC 2009 [6]. It improves the original rebound attack by extending the number of controlled rounds from 2 to 3 so that the total complexity can be greatly reduced. The underlying idea is to utilize the independence and the freedom of each search procedure as much as possible. As a result, without increasing the time and memory, 3 rounds of the differential path can be fulfilled. For more details, we refer to the original publication [6].

Super-Sbox analysis. Super-Sbox analysis for the rebound attack against AES-based permutations was independently proposed by Lamberger *et al.* at Asiacrypt 2009 and by Gilbert and Peyrin at FSE 2010 [7]. It combines 2 non-linear layers (SubBytes) and 1 diffusion layer to 1 non-linear layer with a larger substitution-box named Super-Sbox. By doing so, 2-round AES-based permutation can be regarded as a non-linear layer with a following diffusion layer. When it is applied to the rebound attack, the inbound phase can be extended by one more round. As a side effect, attackers need to spend more time and memory to exploit the differential property of the Super-Sbox. For more details, we refer to the original publication [7].

Dedicated analysis on ECHO. Peyrin proposed a differential path with an increased granularity for a dedicated analysis on ECHO [23, 24]. Since the permutation for each BigWord inside BigSB of the ECHO permutation are still AES-based, the truncated difference can be detailed to byte-wise instead of 128-bit word-wise. By doing so, the number of active bytes for the outbound search can be reduced, so that the total attack complexity can be reduced. For more details, we refer to the original papers [23, 24].

4 Non-Full-Active Super-Sbox Analysis on AES-Based Permutations

In this section, we use the following notations to discuss generic AES-based permutations:

r: a number of rows and columns in a state.
c: a number of bits of each cell (word) in a state.
s: a number of non-active columns in the initial state of the differential path.

$Col(x)$: a state where x columns are fully active, namely, $r \times x$ bytes are active.
$SR(Col(x)), SR^{-1}(Col(x))$: a state where $Col(x)$ is passed through the ShiftRows or InverseShiftRows.
F: a state where all bytes are active.
x/y: a state where y bytes become non-active from a state x.

In the Super-Sbox analysis, as long as we follow the strategy of Gilbert and Peyrin [7], the attack complexity is lower-bounded by 2^{rc}, which is the number of bits in one column, or the number of elements inside the Super-Sbox. In this section, we present a new framework called non-full-active Super-Sbox analysis which can detect non-ideal properties with a lower complexity. In our approach, we first make a truncated differential path whose inbound part, in particular inside the Super-Sbox, consists of only non-full-active states. For non-active bytes, the differential transition 0 to 0 is always held regardless of its value, and thus attackers can freely choose the value without breaking the path. This gives attackers the freedom degrees to adjust other bytes inside the Super-Sbox to be connected efficiently.

Non-full-active Super-Sbox analysis can be applied to a class of AES-based permutations, where its general description is given by [7]. In addition, we assume that the MixColumns operation is designed to satisfy the property of MDS [3]. Namely, the sum of the number of active bytes in input and output of the MixColumns operation is greater than or equal to $r + 1$, otherwise 0.

4.1 Non-Full-Active Truncated Differential Path

We show a generic description of the non-full-active differential path. The differential path has a parameter s, where s is the number of non-active columns in the initial state. The parameter s will determine the complexity of the distinguishing attack. The differential path is depicted in Fig. 6 with instantiating the case $r = 8$ and $s = 3$.

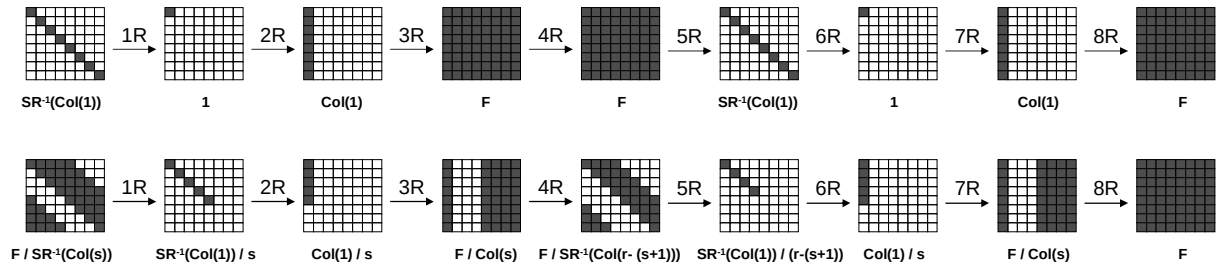


Fig. 6. (Bottom) New differential path for 8-round AES-based permutations with instantiating $r = 8$ and $s = 3$. (Top) Previous differential path used in the Super-Sbox analysis [7].

To make the differential path, we start from the state after the 2nd and 5th round, whose states are $Col(1)/s$ and $SR^{-1}(Col(1))/(r - (s + 1))$, respectively. The differential propagation through the 3rd round in forward and the 5th round in backward are deterministic, which result in $F/Col(s)$ and $F/SR^{-1}(Col(r - (s + 1)))$, respectively. We then need to check that the differential propagation through the MixColumns operation in the 4th round is consistent with the MDS property. Because input and output states have $r - s$ and $r-(r-(s+1))$ active bytes in each column respectively, the sum of active bytes in the input and output is $r - s + r - (r - (s + 1)) = r + 1$. Hence, the differential path is consistent with the MDS property. Next, we determine the differential propagation through the 6th round in forward. The number of active bytes should be reduced as much as possible after the 6th round in order to make the target non-ideal property hard for an ideal permutation. Hence, we maximize the number of non-active bytes with satisfying the MDS property, which results in the state $Col(1)/s$. Similarly, we determine the differential propagation through

the 2nd round in backward. We make the number of active bytes to be the same as the state after the 6th round[1], which results in $SR^{-1}(Col(1))/s$. The rest of the path is deterministic, and we thus omit the details.

4.2 Low Complexity Inbound Phase

We explain how to compute the inbound phase for the non-full-active differential path shown in Section 4.1. We show the details of each state inside the inbound phase in Fig. 7. As described in Fig. 7, we denote each state in the inbound phase by $\#i$, where $0 \leq i \leq 8$. The inbound phase starts from the state after the

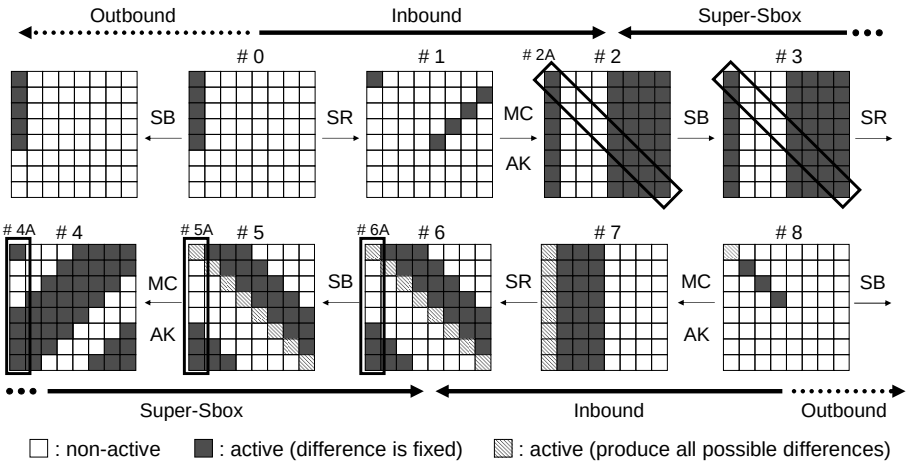


Fig. 7. Inbound phase for the new differential path with non-full-active states.

SubBytes in the 3rd round (#0) and the state input to the 6th round (#8). The goal of the inbound phase is finding paired values satisfying the differential path through #0 to #8. We find 2^c such paired values with 2^c computations and 2^c amount of memory.

States #0 and #8 include $r-s$ and $s+1$ active bytes, respectively. First, we choose and fix the differences of all active bytes in #0 and the differences of s active bytes out of $s+1$ active bytes in #8. Then, for each 2^c possible differences of the last active bytes in #8, we aim to store a corresponding paired value. Due to the linearity of the operations, we can compute the corresponding differences in state #2 and corresponding s-byte differences in each column of #6. The Super-Sbox analysis can be applied to the computations between #2 and #6, namely we can compute them column by column independently. Previous Super-Sbox analysis spent 2^{rc} of time and 2^{rc} of memory for this computation, while we efficiently connect these two states by using the freedom degrees of the non-active states. In the following, we only show the Super-Sbox computations in the left most column, which is emphasized with bold squares in Fig. 7. The other columns can be connected with the same procedure.

Computation procedure inside the Super-Sbox. The operations inside the Super-Sbox are shown in Fig. 8. Because the ShiftRows operation does not give any impact inside the Super-Sbox, we omit it in Fig. 8. To stress that each Super-Sbox is computed column by column, we denote the states inside the Super-Sbox by #2A, #4A, #5A, and #6A in Fig. 8. The goal of this procedure is efficiently producing 2^c paired values

[1] With a lower probability, the number of active bytes can be smaller than the state after the 6th round. However, this will not lead to any advantage in the distinguishing attack because attackers for an ideal permutation has an access to both encryption and decryption oracles.

which satisfy the fixed part of the differences of #2A and #6A. This procedure finds 2^c paired values with a time complexity of 2^c and 2^c memory. The attack procedure is as follows.

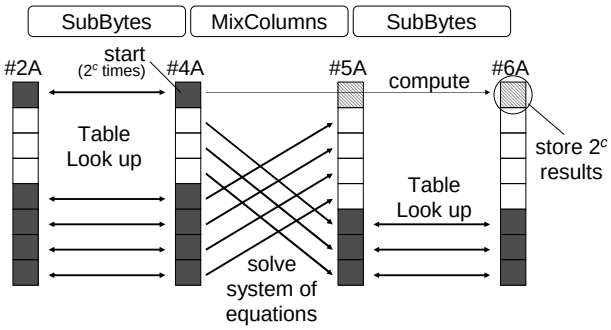


Fig. 8. Computation procedures inside each Super-Sbox.

0. For each active byte whose difference is fixed in #2A and #6A, compute SubBytes and Inverse-SubBytes for all possible 2^c values and a fixed difference. Store these 2^c values and corresponding output differences as a look up table. As a result, $(r-s)+s=r$ look-up tables are prepared.
1. Choose a difference of one active byte in #4A. (The top byte of #4A is chosen in Fig. 8.)
2. We have other $r-s-1$ active bytes in #4A and need to make sure that the same number of bytes in #5A are non-active. This can be done by solving a system of equations and we will obtain one solution of the system. As a result, differences in #4A and #5A become consistent and are uniquely fixed.
3. From a fixed difference of #4A and the given difference of #2A, for each active byte, we obtain a pair of values which connects these differences by looking up tables generated in Step 0. Do the same for fixed s-byte differences of #6A and #5A. Note that values for non-active bytes are not fixed yet at this stage.
4. The remaining work is connecting the values of active bytes of #4A and #5A. We use the freedom degrees of non-active bytes to effectively achieve this. There are s non-active bytes in #4A and s active bytes in #5A whose values are fixed in Step 3. By solving a system of equations, we calculate the values of s non-active bytes in #4A so that the fixed s bytes of #5A can be consistent.
5. With the fixed values in #4A, we compute the non-fixed active byte in #5A, and further compute the corresponding value in #6A. We store entire values and differences of states #2A and #6A in a table.
6. We iterate Step 1 to Step 5 2^c times by changing the difference of the chosen active byte in #4A.

Complexity of inbound phase. In the above procedure, Step 0 requires 2^c computations and 2^c amount of memory. Step 1 to Step 4 can be computed with a complexity of 1 and Step 5 uses a memory of 1. Because Steps 1 to 5 are repeated 2^c times in Step 6, the complexity of this procedure is time 2^c and memory 2^c. Note that 2^c values and differences of the non-fixed active byte are stored in the table. Therefore, we obtain 1 solution in average for any difference of the non-fixed byte.

After we finish the computation for all Super-Sboxes, we choose a difference of the non-fixed byte in #8 in Fig 7. For each of its possible 2^c differences, we compute the corresponding difference in #6, and obtain the value which connects #2 to #6 by looking up each Super-Sbox. Note, we obtain one solution in average for any pair of differences in #2 and #6. To sum up, we can obtain 2^c starting points, which are solutions of the inbound phase, with time 2^c and memory 2^c. In other words, we obtain a starting point with time 1 in average.

4.3 Outbound phase and the freedom degrees

After the inbound phase, we need to compute the outbound phase. The differential path described in Fig. 6 has two probabilistic differential propagations: 1) the backward computation through the 2nd round and 2) the forward computation through the 6th round. In both rounds, the MixColumns or InverseMixColumns operations need to produce s non-active bytes. Therefore, for each of these rounds, the success probability is 2^{-cs}. Finally, this attack requires 2^{2cs} starting points for the outbound phase, and each starting point is generated with time 1 in average. Therefore, with a time 2^{2cs}, we can find a pair that follows the differential path.

We also need to confirm that the available freedom degree is enough. Our attack starts from the states #0 and #8 in Fig. 7. #0 and #8 include $r-s$ and $s+1$ active bytes respectively, and thus we have $2^{c(r+1)}$ freedom degrees in total. Hence, as long as the parameter s satisfies $2^{c(r+1)} \geq 2^{2cs}$, which is converted as shown below, we have enough freedom degrees.

$$s \leq \frac{r+1}{2} \tag{1}$$

4.4 Target Class of AES-Based Permutations and an Example

Let us consider the complexity for an ideal permutation. As was done in the previous analysis [7], the last MixColumns is not taken into account because it is fully linear. Therefore, the problem is regarded as finding a crs-bit collision. A crs-bit collision can be found by the birthday attack because attackers have enough freedom degrees due to Eq. (1). Hence, the complexity for an ideal permutation is $2^{\frac{crs}{2}}$. The comparison of the non-full-active Super-Sbox analysis and the ideal case is as follows.

Table 2. The complexity to find a property with our attack and ideal permutation.

s	1	2	3	4	5	6	7	8
Ours	2^{2c}	2^{4c}	2^{6c}	2^{8c}	2^{10c}	2^{12c}	2^{14c}	2^{16c}
Ideal	$2^{\frac{cr}{2}}$	2^{cr}	$2^{\frac{3cr}{2}}$	2^{2cr}	$2^{\frac{5cr}{2}}$	2^{3cr}	$2^{\frac{7cr}{2}}$	2^{4cr}

From Table 2, we can find a condition of r where our attack can detect a property of the analysis target faster than the ideal permutation. We can see as long as the permutation is a general AES-based permutation, our attack cannot be faster than the ideal permutation if $r \leq 4$. Therefore, our attack does not improve the attack on AES. However, for $r > 4$, our attack can work in generic. Note that the ECHO permutation is regarded as an AES-based permutation with $r = 4$ at a BigWord level. However, it has other internal structures and thus not a pure AES-based permutation. This enables us to greatly reduce the attack complexity on the ECHO permutation even the parameter r is 4 at a BigWord level. See Section 5 for details.

Let us consider an application of the low complexity distinguisher to a real primitive. Grøstl-256 uses an AES-based permutation with $r = c = 8$. In previous Super-Sbox analysis [7], the 8-round permutation is distinguished with time 2^{112} and memory 2^{64}, which is too expensive with the current computation resource to detect a pair of values satisfying the property. In our attack, we choose the differential path with $s = 3$, where its differential path is the same as shown in Fig. 6. Consequently, from Table 2, we can detect a pair of values following the differential path with time 2^{48} and 2^8 memory, while finding a pair of values in an ideal permutation requires 2^{96}, which is infeasible. Choosing other s is also possible as long as $s \leq 4$.

5 Applications to ECHO and Grøstl

5.1 New Observations on ECHO

In this section, we explain several new observations on the ECHO permutation when dealing with the dedicated byte-wise differential path. First, we explain that the complexity analysis cannot be separated for

the MixColumns and the following BigMC. Second, we show that the freedom of the differential path inside BigSB can be used to reduce the attack complexity.

Complexity analysis for jointed MixColumns and BigMC. In the ECHO permutation, 2-round AES permutation inside BigSB can be considered as a non-linear layer with Super-Sboxes and a diffusion layer consisting of ShiftRows, MixColumns and AddRoundKey. Notice that the second MixColumns inside BigSB and the following BigMC are successively performed when we neglect the in-between AddRoundKey and BigSR that do not affect the difference. We show that the linearity of jointed MixColumns and BigMC should be considered to correctly compute the complexity for certain differential paths.

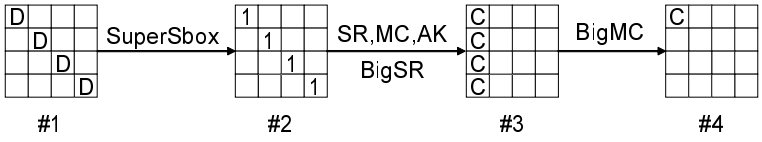


Fig. 9. A differential path for a 1-round ECHO permutation.

As an example, let us check the complexity for the differential path shown in Fig. 9 assuming the differences and real values at state #1 have full freedom. In the previous analysis [23, Appendix B], the complexity for this differential path is likely to be divided into three parts and analyzed independently. The differential path from state #1 to #2 can be fulfilled when the output of each active Super-Sbox has only 1 active byte. Since there are totally 12 bytes required to become zero probabilistically, the probability from #1 to #2 is regarded as 2^{-96}. The complexity from #2 to #3 is 1. And since there are totally 12 bytes required to become zero from #3 to #4 probabilistically, probability is regarded as 2^{-96} as well. As a result, the total probability for the differential path shown in Fig. 9 is regarded as $2^{-96\times 2} = 2^{-192}$. However we show that MC and BigMC cannot be considered separately, and thus the correct probability needs to be reconsidered.

We can see that the freedom of the difference for state #2 or #3 is at most 2^{32}, since #2 has only 4 active bytes and the transformation from #2 to #3 is deterministic. As a contradiction for the previous analysis, the freedom of difference at #3 (2^{32}) seems impossible to fulfill the differential propagation to #4 (2^{-96}). However, we show that the transformation from #2 to #4 is fulfilled only with a probability of 2^{-24}, and thus 2^{32} freedom degrees are enough to fulfill the differential path.

This fact can be understood from two directions. First, for a position-fixed active byte and the fixed MDS matrix used in MixColumns between #2 and #3, the 4 active bytes inside each active BigWord at #3 has a fixed linear relationship. Then if BigMC generates the required difference at #4 for one of 4 active-byte positions with a probability of 2^{-24} (e.g. 4 top-left bytes from 4 active BigWord at #3 generate 1 active byte at the top-left of state #4), the other three active-byte positions become the same differential pattern at #4 with probability 1. In fact, the 4 active bytes in the left-top BigWord at #4 have the same linear relationship determined by the used MDS matrix in MixColumns. Another understanding method is that one can switch the sequence of the jointed linear operations by performing BigMC first and MixColumns later. When 4 active bytes in #2 generate only 1 active byte through BigMC with a probability of 2^{-24}, the differential path from #3 to #4 through MixColumns is fulfilled with probability 1. As a result, we can explain the complexity for the differential path shown in Fig. 9 is actually $2^{96+24} = 2^{120}$ instead of 2^{192}.

Freedom of the differential path inside BigSB. Another useful observation is that we can use the freedom of the differential path inside BigSB to reduce the attack complexity. When we distinguish the ECHO permutation from an ideal one, we only care about the differences at the start state and the end state of the permutation. We notice that when dedicated analysis is applied, in the meanwhile of keeping

the differential path at a BigWord level be untouched, attackers can use the freedom of the differential path inside the BigSB transformation to reduce the complexity.

We can still use the differential path in Fig. 9 as an example. In order to fulfill the differential path, the 4 active bytes in state #2 must be at the same position inside the leftmost column of each BigWord[2]. As a result, the differential path inside the BigSB has 4 choices for the positions of active bytes. By using all these 4 differential-path variations, the complexity for the differential path in Fig. 9 can be reduced by 4 from 2^{120} to 2^{118}.

5.2 Attack on Full-Round ECHO Permutation

Truncated Differential path. In this attack, we use the differential path explained in Section 4.1 with parameter $s = 1$ at a BigWord level. As explained in Section 4.4, the path cannot be used for AES-based permutations with $r = 4$. However, by taking the properties inside BigWords into account, we can efficiently apply our path to the ECHO permutation. We show the differential path in Fig. 10. We use the notation BigSB$[x, y, z]$, where $x, y, z \in \{F, D, C, 1\}$ to show that x, which is the input differential pattern to BigSB, changes into y after the 1st AES-round and into z after the 2nd AES-round.

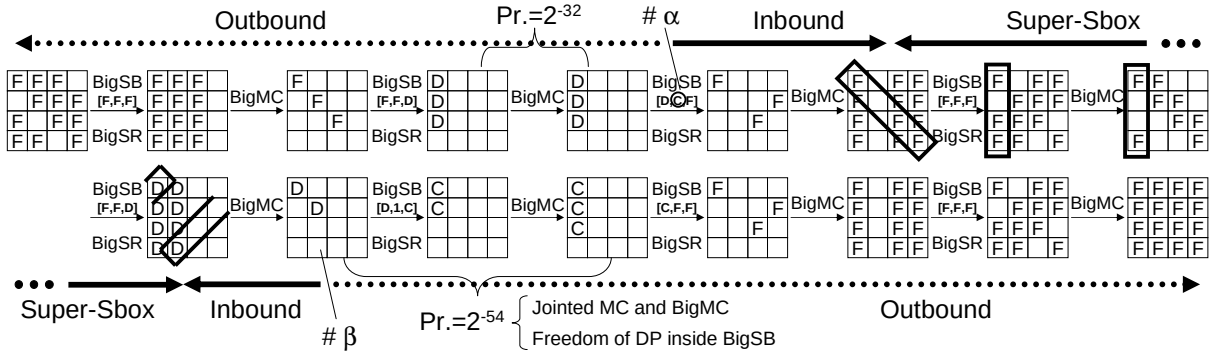


Fig. 10. Differential path for the low complexity distinguishing attack on 8-round ECHO permutation.

Inbound phase. The detailed differential path for the inbound phase is described in Fig. 11. The inbound phase starts from a middle of BigSB in the 3rd round ($\#\alpha$) and the input state to the 6th round ($\#\beta$), where the differential form in $\#\alpha$ is C. We first choose and fix a difference of $\#\alpha$ and a difference of one of active BigWords of $\#\beta$, and compute the corresponding differences of #2 and #6. In the inbound phase, for each of 2^{32} possible differences of the non-fixed active BigWord in $\#\beta$, we find a pair of values that satisfies the chosen differences of $\#\alpha$ and $\#\beta$. The attack procedure follows the one explained in Section 4.2 with some optimization specific to ECHO. In the followings, we describe the detailed procedure to compute 1 Super-Sbox of ECHO with the size of 128 bits.

0. Generate a look-up table for each of 4 active BigWord whose difference is fixed in #2A and #6A, With the procedure in Section 4.2, this costs 2^{128} time and 2^{128} memory. However, [6] pointed out that this could be performed efficiently by looking inside BigSB. The BigSB can be regarded as 4 Super-Sboxes (SB, SR, MC, AK, SB) with the size of 32 bits and the linear part (SR, MC, AK). Hence, look-up tables for 4 active BigWords can be generated by computing 16 Super-Sboxes, which requires 16×2^{32} in both time and memory.

[2] One can check that when the 4 active bytes in state #2 are not at the same position inside each BigWord, the differential path for Fig. 9 becomes impossible. This may be used as an efficient countermeasure against our attack.

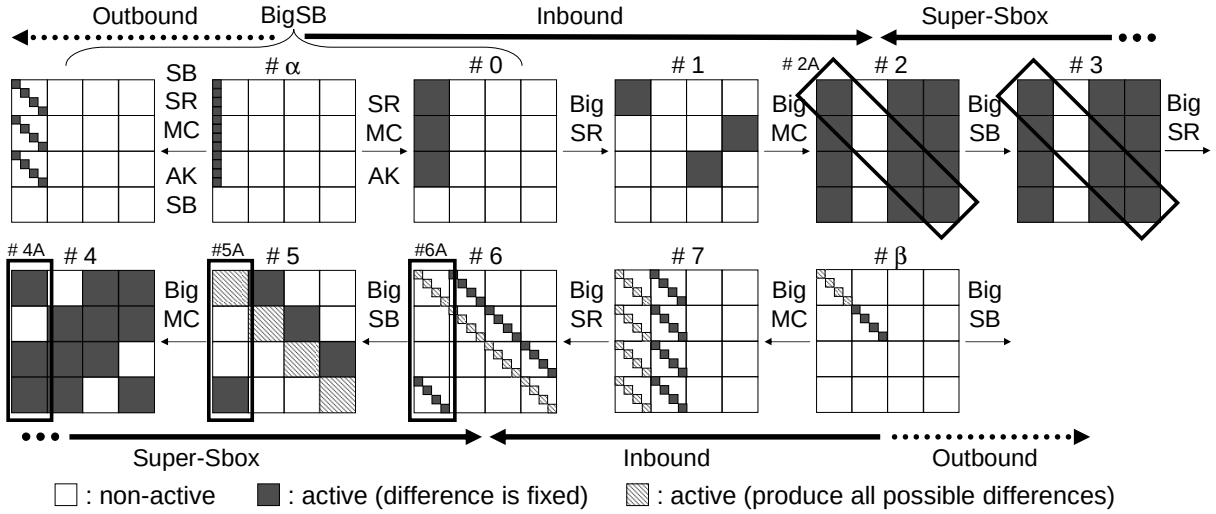


Fig. 11. Inbound phase for 8-round ECHO permutation.

1. Choose a difference of one active BigWord in #4A.
2. By solving a system of equations, compute differences of 2 active BigWords in #4A so that 2 target BigWords in #5A can be non-active.
3. For each active BigWord with fixed difference, obtain a pair of values which connects differences between #4A and #2A, and between #6A and #5A by looking up tables generated in Step 0.
4. By solving a system of equations, calculate the value of 1 non-active BigWord in #4A so that the fixed value of 1 BigWord in #5A can be consistent.
5. With the fixed paired values in #4A, compute the non-fixed active BigWord in #5A and #6A. Only if the computed difference of #6A has the diagonal form D, store entire values and differences of states #2A and #6A in a table.
6. Iterate Steps 1 to 5 2^{128} times by changing the difference of the chosen BigWord in #4A.

In Step 0, look up tables are generated with 2^{36} time and 2^{36} memory. Steps 1 to 5 are iterated 2^{128} times. In Step 5, the computed difference has the diagonal form D with a probability of $2^{32}/2^{128} = 2^{-96}$, and thus we store 2^{32} data after 2^{128} iterations. Hence, the complexity for 1 Super-Sbox with the size of 128 bits is 2^{128} computations and $2^{36} + 2^{32}$ memory. Note that we need $2^{36} + (4 \times 2^{32}) < 2^{37}$ memory for 4 Super-Sboxes. In the end, the inbound phase generates 2^{32} starting points with 2^{128} computations and 2^{37} memory, which is 2^{96} computations in average to generate 1 starting point.

Success probability and freedom degrees. If details are considered, Step 3 succeeds only probabilistically. Look-up tables for each BigWord consists of 4 Super-Sboxes with the size of 32 bits. Assume that each Super-Sbox has the same property as the AES Sbox. Namely, for a randomly given a pair of input and output differences, with a probability of approximately 2^{-1}, there exists approximately 2 paired values satisfying the differences. In Step 3, we look-up 16 Super-Sboxes. Hence, the success probability is 2^{-16} and we obtain 2^{16} paired values. We compute Steps 4 and 5 for all 2^{16} paired values, and thus they are computed 2^{128} times in total by the 2^{128} iteration of Step 6. Consequently, the total time and memory for the inbound phase will not change. Note that the estimation by using average numbers is imprecise only if the cost for the outbound phase is cheaper than the inbound phase. Because our attack iterates the inbound phase 2^{54} times, the evaluation with average numbers is valid.

We also need to check the freedom degrees. In the initial part of the inbound phase, we can choose up to 2^{96} differences for $\#\alpha$ and 2^{32} differences for the fixed active BigWord in $\#\beta$. Hence, the above inbound

phase can be iterated 2^{128} times and thus we can generate 2^{160} starting points in maximum, which are enough to satisfy the outbound phase.

Outbound phase. The differential path shown in Fig. 10 includes two probabilistic differential propagations.

1. **InverseBigMC in the 2nd round.** For each of diagonal byte-positions, Inverse MixColumns needs to output one non-active byte. Hence, the success probability is $(2^{-8})^4 = 2^{-32}$.
2. **BigSB and BigMC in the 6th round.** Observations explained in Section 5.1 are applied for this part. The probability that the differences in 2 BigWords propagate as $D \rightarrow 1 \rightarrow C$ is $(2^{-24})^2 = 2^{-48}$. By taking the freedom of the differential path inside BigSB into account, the probability becomes $4 \times 2^{-48} = 2^{-46}$. In the BigMC operation, MC is computed for 4 positions. Due to the property of jointed MixColumns and BigMC operations, all of the 4 positions will make 1 non-active byte with a probability of 2^{-8} in total. As a result, the total success probability of the 6th round is $2^{-46} \times 2^{-8} = 2^{-54}$.

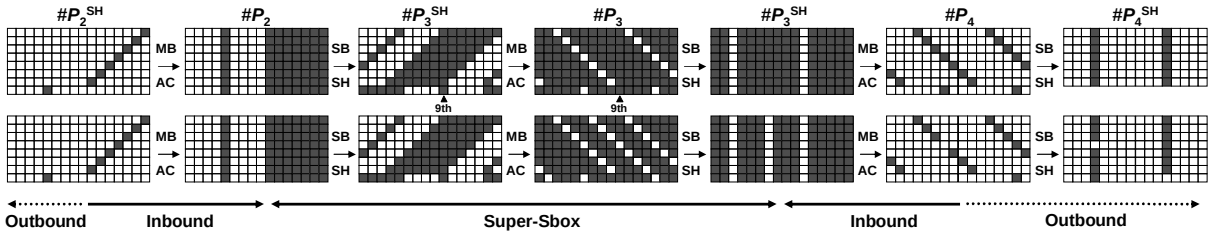
In the end, the success probability of the outbound phase is $2^{-32} \times 2^{-54} = 2^{-86}$.

Total complexity and comparison with ideal case. In our attack, we need to generate 2^{86} starting points and each starting point is generated with 2^{96} computations in average. Hence, the total complexity is $2^{86} \times 2^{96} = 2^{182}$. Note that this attack requires 2^{37} amount of memory. On the other hand, for the ideal case, the property is regarded as finding a 512-bit collision. Because available freedom degrees are enough to mount the birthday attack, the complexity is 2^{256}, which is much higher than our attack on ECHO.

5.3 Improving Semi-Free-Start Collision Attack on 7-round Grøstl-512

We improve the semi-free-start collision attack on 7-round Grøstl-512 compression function, which was proposed by Mendel *et al.* [15]. The authors of [15] used the same strategy as previous Super-Sbox analysis and thus required 2^{64} amount of memory for the inbound phase. We show that the amount of memory can be reduced to 2^{56} with the non-full-active Super-Sbox analysis. Because our outbound phase is the same as [15], we only explain the inbound phase.

We start from an observation that in the Super-Sbox analysis for AES-based permutations with a rectangle state such as $r \times 2r$, several Super-Sboxes usually include non-active bytes. Hence, the framework explained in Section 4 can be applied and the data stored for each Super-Sbox can be smaller than 2^{rc}. However, in the previous differential path [15, Fig.7] shown in the top of Fig. 12, the 9th Super-Sbox (counted from 0th) from the left side in the state $\#P_3^{SH}$ takes a full-active column as input and output a full-active column. Hence, attackers need 2^{64} amount of memory for this part, and this is a bottleneck of the memory size in the entire attack.



SH, MB, and AC represents the ShiftBytes, MixBytes, and AddRoundConstant operations used in Grøstl which correspond to the ShiftRows, MixColumns, and AddRoundKey operations of the AES permutation.

Fig. 12. (Bottom) New differential path for Grøstl-512 compression function. (Top) Previous path.

In our attack, we reduce the number of active bytes where we choose their differences at the initial step of the inbound phase ($\#P_4$). Due to this effort, we can make a differential path where each Super-Sbox has at least one non-active byte. The new path is shown in the bottom of Fig. 12. Inside of each Super-Sbox can be computed based on the procedure explained in Section 4.2, which results in generating 2^{56} starting points with 2^{56} time and 2^{56} memory. In the end, we can reduce the memory size to 2^{56} with maintaining the same average complexity of the inbound phase. Note that the differential propagation from $\#P_3^{SH}$ to $\#P_3$ must be consistent with the MDS property. We confirmed that the amount of memory could not be below 2^{56} due to this limitation.

Because we reduced the number of active bytes at the initial step of the inbound phase, the freedom degree was also reduced. In this attack, the success probability of the outbound phase is 2^{-152}, and thus we need 2^{152} starting points. Because our attack can choose 22-byte differences (8-byte differences for $\#P_2^{SH}$ and 14-byte differences for $\#P_4$) at the initial step, our attack can produce $2^{8\times 22} = 2^{176}$ starting points in maximum. Therefore, we have enough freedom degrees to find a pair of values following the path.

6 Conclusions

We presented the non-full-active Super-Sbox analysis which can detect non-ideal properties of a class of AES-based permutations with a low complexity. The core idea is using a differential path consisting of only non-full-active states. This gives us the freedom to efficiently control inside the Super-Sbox. We then applied this framework to the full-round ECHO permutation by taking properties specific to ECHO into account. Consequently, our attack could detect a non-ideal property with time 2^{182} and memory 2^{37}. However, because of the convolution operation that is used, our distinguisher cannot be extended to the ECHO compression function. We then applied our approach to Grøstl to obtain the distinguishing attack on the 8-round Grøstl-256 permutation with a practical cost, and to obtain an improvement on the semi-free-start collision attack on the 7-round Grøstl-512 compression function.

References

1. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices. (2007) http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.
2. U.S. Department of Commerce, National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES) (Federal Information Processing Standards Publication 197). (2001)
3. Daemen, J., Rijmen, V.: The design of Rijndael: AES – the Advanced Encryption Standard (AES). Sprnger-Verlag (2002)
4. Peyrin, T.: Cryptanalysis of Grindahl. In Kurosawa, K., ed.: Advances in Cryptology - ASIACRYPT 2007. Volume 4833 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2007) 551–567
5. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In Dunkelman, O., ed.: Fast Software Encryption 16th International Workshop, FSE 2009. Volume 5665 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 260–276
6. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography 2009. Volume 5867 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 16–35
7. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong, S., Iwata, T., eds.: Preproceedings of FSE 2010. (2010) 368–387
8. Rijmen, V., Toz, D., Varici, K.: Rebound attack on reduced-round versions of JH. In Hong, S., Iwata, T., eds.: Preproceedings of FSE 2010. (2010) 289–306
9. Wu, S., Feng, D., Wu, W.: Cryptanalysis of the LANE hash function. In Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography 2009. Volume 5867 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 126–140

10. Wu, S., Feng, D., Wu, W.: Practical rebound attack on 12-round Cheetah-256. ICISC 2009, Session 8A in Pre-proceedings distributed at the conference (2009)
11. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full LANE compression function. In Matsui, M., ed.: Advances in Cryptology - ASIACRYPT 2009. Volume 5912 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 106–125
12. Mendel, F., Rechberger, C., Schläffer, M.: Cryptanalysis of Twister. In Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D., eds.: Applied Cryptography and Network Security, ACNS 2009. Volume 5536 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 342–353
13. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full Whirlpool compression function. In Matsui, M., ed.: Advances in Cryptology - ASIACRYPT 2009. Volume 5912 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2009) 126–143
14. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The rebound attack and subspace distinguishers: Application to Whirlpool. Cryptology ePrint Archive, Report 2010/198 (2010) http://eprint.iacr.org/2010/198.
15. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attack on the reduced Grøstl hash function. In Pieprzyk, J., ed.: CT-RSA 2010. Volume 5985 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (2010) 350–365
16. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl addendum. Submission to NIST (updated) (2009)
17. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (updated) (2009)
18. Wu, H.: The hash function JH. Submission to NIST (updated) (2009)
19. Khovratovich, D., Biryukov, A., Nikolić, I.: The hash function Cheetah: Specification and supporting documentation. Submission to NIST (2008)
20. Indesteege, S.: The LANE hash function. Submission to NIST (2008)
21. Ewan Fleischmann, C.F., Gorski, M.: The Twister hash function family. Submission to NIST (2008)
22. Rijmen, V., Barreto, P.S.L.M.: The Whirlpool hashing function. Submitted to NISSIE (2000)
23. Peyrin, T.: Improved cryptanalysis of ECHO and Grøstl. Cryptology ePrint Archive, Report 2010/223 (2010) http://eprint.iacr.org/2010/223, Extended final version of the CRYPTO 2010 article.
24. Peyrin, T.: Improved cryptanalysis of ECHO and Grøstl. Rump session at FSE 2010 (2010)
25. Schläffer, M.: Subspace distinguisher for 5/8 rounds of the ECHO-256 hash function. Cryptology ePrint Archive, Report 2010/321 (2010) http://eprint.iacr.org/2010/321.