Improved Rebound Attack on the Finalist Grøstl

Jérémy Jean[1,⋆,⋆⋆], María Naya-Plasencia[2,⋆], and Thomas Peyrin[3,⋆⋆⋆]

[1] École Normale Supérieure, France
[2] University of Versailles, France
[3] Nanyang Technological University, Singapore

Abstract. Grøstl is one of the five finalist hash functions of the SHA-3 competition. For entering this final phase, the designers have tweaked the submitted versions. This tweak renders inapplicable the best known distinguishers on the compression function presented by Peyrin [18] that exploited the internal permutation properties. Since the beginning of the final round, very few analysis have been published on Grøstl. Currently, the best known rebound-based results on the permutation and the compression function for the 256-bit version work up to 8 rounds, and up to 7 rounds for the 512-bit version. In this paper, we present new rebound distinguishers that work on a higher number of rounds for the permutations of both 256 and 512-bit versions of this finalist, that is 9 and 10 respectively. Our distinguishers make use of an algorithm that we propose for solving three fully active states in the middle of the differential characteristic, while the Super-Sbox technique only handles two.

Keywords: Hash Function, Cryptanalysis, SHA-3, Grøstl, Rebound Attack.

1 Introduction

Hash functions are one of the main families in symmetric cryptography. They are functions that, given an input of variable length, produce an output of a fixed size. They have many important applications, like integrity check of executables, authentication, digital signatures.

Since 2005, several new attacks on hash functions have appeared. In particular, the hash standards MD5 and SHA-1 were cryptanalysed by Wang et al. [21,22]. Due to the resemblance of the standard SHA-2 with SHA-1, the confidence in the former has also been somewhat undermined. This is why the American National Institute of Standards and Technology (NIST) decided to launch in 2008 a competition for finding a new hash standard, SHA-3. This competition received 64 hash function submissions and accepted 51 to enter the first round. Now, three years and two rounds later, only 5 hash functions remain in the final phase of the competition.

Amongst these finalists, there is only one AES-based function, though many were proposed. This hash function is Grøstl [2], and is at the origin of the introduction of a new cryptanalysis technique that has been widely deployed, improved and applied to a large number of SHA-3 candidates, hash functions and other types of constructions. This new technique, called rebound attack, was introduced by Mendel et al. [11] and has become one of the most important tools used to analyze the security margin of many SHA-3 candidates as well as their building blocks. As for Grøstl itself, it has been applied and improved in several occasions [3, 12, 13, 15, 18]. Grøstl is undoubtedly one of the SHA-3 candidates that have received the largest amount of cryptanalysis. When entering the final round, a tweak of the function was proposed, which prevents the application of the attacks from [18]; we denote Grøstl-0 the original submission of the algorithm and Grøstl its tweaked version. Apart from the rebound results, the other main

⋆ Supported by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.
⋆⋆ Supported by the French *Délégation Générale pour l'Armement* (DGA).
⋆⋆⋆ The author is supported by the Lee Kuan Yew Postdoctoral Fellowship 2011 and the Singapore National Research Foundation Fellowship 2012.

analysis communicated on Grøstl was at the presentation of [1] where a higher order property on 10 rounds of Grøstl-256 permutation with a complexity of 2^{509} was shown. In Table 1, we report a summary of the best known results on both 256 and 512-bit tweaked versions of Grøstl, including the ones that we will present in the following.

In this paper, we propose new results regarding both versions of the finalist Grøstl. First, on Grøstl-256, we provide the best known rebound distinguishers on 9 rounds of the permutation. From these results, we show how to make some nontrivial observations on the the compression function, providing the best known analysis on the compression function exploiting the properties of the internal permutations. For Grøstl-512, we considerably increase the number of analyzed rounds, from 7 to 10, providing the best analysis known on the permutation. Both results are obtained using rebound-like attack techniques and an algorithm that we introduce that allows to solve three fully active rounds in the middle of the differential characteristic with a much lower cost than a generic algorithm. Additionnally, we provide in Appendix A the direct application of our new techniques to the AES-based hash function PHOTON.

These results do not threaten the security of Grøstl, but we believe they will have an important role in better understanding Grøstl, and AES-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in these types of constructions.

Target	Subtarget	Rounds	Time	Memory	Ideal	Reference
Grøstl-256	Permutation	8 (dist.)	2^{112}	2^{64}	2^{384}	[3]
		8 (dist.)	2^{48}	2^{8}	2^{96}	[19]
		9 (dist.)	2^{368}	2^{64}	2^{384}	Section 3
		10 (zero-sum)	2^{509}	–	2^{512}	[1]
Grøstl-512	Permutation	8 (dist.)	2^{280}	2^{64}	2^{448}	Section 4
		9 (dist.)	2^{328}	2^{64}	2^{384}	Section 4
		10 (dist.)	2^{392}	2^{64}	2^{448}	Section 4

Table 1: Best known analysis on the finalist Grøstl. By best analysis, we mean the ones on the highest number of rounds.

2 Generalities

2.1 Description of Grøstl

The hash function Grøstl-0 has been submitted to the SHA-3 competition under two different versions: Grøstl-0-256, which outputs a 256-bit digest and Grøstl-0-512 with a 512-bit fingerprint. For the final round of the competition, the candidate have been tweaked to Grøstl, with corresponding versions Grøstl-256 and Grøstl-512.

The Grøstl hash function handles arbitrary long messages by diving them into blocks after some padding and uses them to update iteratively an internal state (initialized to a predefined IV) with a compression function. This function is itself built upon two different permutations, namely P and Q. Each of those two permutations updates a large internal state using the well-understood wide-trail strategy of the AES. As an AES-like Substitution-Permutation Network, Grøstl enjoys a strong diffusion in each of the two permutations and by its wide-pipe design, the size of the internal states is ensured to be at least twice as large as the final digest.

The compression function f_{256} of Grøstl-256 uses two permutations P_{256} and Q_{256}, which are similar to the two permutations P_{512} and Q_{512} used in the compression function f_{512} of Grøstl-512. More precisely, for a chaining value h and a message block m, the compression functions (Figure 1) produce the output ($\oplus$ denotes the XOR operation):

$$f_{256}(h, m) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus h, \qquad \text{or:} \qquad f_{512}(h, m) = P_{512}(h \oplus m) \oplus Q_{512}(m) \oplus h.$$

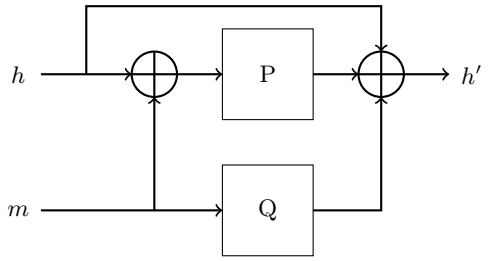


Figure 1: The compression function of Grøstl hash function using the two permutations P and Q.

The internal states are viewed as byte matrices of size 8×8 for the 256-bit version and 8×16 for the 512-bit one. The permutations strictly follow the design of the AES and are constructed as N_r iterations of the composition of four basic transformations:

$$R \stackrel{\text{def}}{=} \textbf{MixBytes} \circ \textbf{ShiftBytes} \circ \textbf{SubBytes} \circ \textbf{AddRoundConstant}.$$

All the linear operations are performed in the same finite field $GF(2^8)$ as in the AES, defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $GF(2)$. The **AddRoundConstant** (AC) operation adds a predefined round-dependent constant, which significantly differs between P and Q to prevent the internal differential attack [18] taking advantage of the similarities in P and Q. The **SubBytes** (SB) layer is the non-linear layer of the round function R and applies the same SBox as in the AES to all the bytes of the internal state. The **ShiftBytes** (Sh) transformation shifts bytes in row i by $\tau_P[i]$ positions to the left for permutation P and $\tau_Q[i]$ positions for permutation Q. We note that τ also differs from P to Q to emphasize the asymmetry between the two permutations. Finally, the **MixBytes** (Mb) operation applies a maximum-distance separable (MDS) circulant constant matrix M independently to all the columns of the state. In Grøstl-256, $N_r = 10$, $\tau_P = [0, 1, 2, 3, 4, 5, 6, 7]$ and $\tau_Q = [1, 3, 5, 7, 0, 2, 4, 6]$, whereas for Grøstl-512, $N_r = 14$ and $\tau_P = [0, 1, 2, 3, 4, 5, 6, 11]$ and $\tau_Q = [1, 3, 5, 11, 0, 2, 4, 6]$.

Once all the message blocks of the padded input message have been processed by the compression function, a final output transformation is applied to the last chaining value h to produce the final n-bit hash value $h' = \text{trunc}_n(P(h) \oplus h)$, where trunc_n only keeps the last n bits.

2.2 Distinguishers

In this article, we will describe algorithms that find input pairs (X, X') for the permutation P (or the permutation Q), such that the input difference $\Delta_{IN} = X \oplus X'$ belongs to a subset of size IN and the output difference $\Delta_{OUT} = P(X) \oplus P(X')$ belongs to a subset of size OUT. The best known generic algorithm (this problem is different than the one studied in [8] where linear subspaces are considered) in order to solve this problem, known as limited-birthday problem, has been given in [3] and later a very close lower bound has been proven in [16]. For a randomly chosen n-bit permutation π, the generic algorithm can find such a pair with complexity

$\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$. If one is able to describe an algorithm requiring less computation power, then we consider that a distinguisher exists on the permutation π.

In the case of Grøstl, it is also interesting to look at not only the internal permutations P and Q, but also the compression function f itself. For that matter, we will generate compression function input values (h, m) such that $\Delta_{IN} = m \oplus h$ belongs to a subset of size IN, and such that $\Delta_{IN} \oplus \Delta_{OUT} = f(h, m) \oplus f(m, h) \oplus h \oplus m$ belongs to a subset of size OUT. Then, one can remark that:

$$\begin{aligned} f(h, m) \oplus f(m, h) &= P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus P_{256}(m \oplus h) \oplus Q_{256}(h) \oplus h \oplus m, \\ f(h, m) \oplus f(m, h) &= Q_{256}(m) \oplus Q_{256}(h) \oplus h \oplus m, \\ f(h, m) \oplus f(m, h) \oplus h \oplus m &= Q_{256}(m) \oplus Q_{256}(h). \end{aligned}$$

Since the permutation Q is supposed to have no structural flaw, the best known generic algorithm requires $\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$ operations (the situation is exactly the same as the permutation distinguisher with permutation Q) to find a pair (h, m) of inputs such that $h \oplus m \in IN$ and $f(h, m) \oplus f(m, h) \oplus h \oplus m \in OUT$. Note that both IN and OUT are specific to our attacks.

We emphasize that even if trivial distinguishers are already known for the Grøstl compression function (for example fixed-points), no distinguisher is known for the internal permutations. Moreover, our observations on the compression function use the differential properties of the internal permutations.

3 Distinguishers for reduced Grøstl-256 internal permutations

In this section, we describe a distinguisher for the permutation P_{256} of the Grøstl-256 compression function reduced to 9 rounds. We emphasize that in the latest version of the Grøstl submission [20], the permutation Q_{256} has different coefficients in the **ShiftRows** transformation, but the technique we describe in the following applies to Q_{256} as well.

3.1 The truncated differential characteristic

In the following, we will consider truncated differential characteristics, originally introduced by Knudsen [7] for block cipher analysis. With this technique, already proven to be efficient for AES-based hash functions cryptanalysis [5, 6, 10, 17], the attacker only checks if there is a difference in a byte (active byte, denoted by a black square in the Figures) or not (inactive byte, denoted by an empty square in the Figures) without caring about the actual value of the difference.

The truncated differential characteristic we use has the sequence of active bytes

$$8 \xrightarrow{R_1} 1 \xrightarrow{R_2} 8 \xrightarrow{R_3} 64 \xrightarrow{R_4} 64 \xrightarrow{R_5} 64 \xrightarrow{R_6} 8 \xrightarrow{R_7} 1 \xrightarrow{R_8} 8 \xrightarrow{R_9} 64,$$

where the size in the input and output differences subsets are both $IN = OUT = 2^{8\times 8} = 2^{64}$, since there are eight active bytes in each extreme state of the truncated characteristic. The actual truncated characteristic is reported in Appendix B.

Note that we have three fully active internal states in the middle of the differential characteristic, thus impossible to handle with the classical rebound or **SuperSBox** techniques.

3.2 Finding a conforming pair

The method to find a pair of inputs conforming to this truncated differential characteristic is similar to the rebound technique: we first find many solutions for the middle rounds (round 3 to round 6) and then we filter them out during the outwards probabilistic transitions through the **MixBytes** layers (round 2 and round 7). We denote $x \to y$ a non-null truncated differential transition mapping x active bytes to y active bytes in a column through a **MixBytes** (or **MixBytes**$^{-1}$) layer, and the MDS property ensures $x + y \geq 9$. Its differential probability is determined by the number $(8 - y)$ of inactive bytes on the output: $2^{-8(8-y)}$ if the MDS property is verified, 0 otherwise.

Therefore, since in our case we have two transitions $8 \to 1$ (see Figure 2), the outbound phase has a success probability of $\left(2^{-8\times 7}\right)^2 = 2^{-112}$ and is straightforward to handle once we found enough solutions for the inbound phase.

In order to find solutions for the middle rounds (see Figure 2), we propose an algorithm inspired by the ones in [14, 15]: As in [3, 8], instead of dealing with the classical 8-bit **SubBytes** SBoxes, one can consider 64-bit SBoxes (named **SuperSBoxes**) each composed of two `AES` SBox layers surrounding one **MixBytes** and one **AddRoundConstant** function[1]. Indeed, the **ShiftBytes** can be taken out from the **SuperSBoxes** since it commutes with **SubBytes**.

We start by choosing the input difference δ_{IN} after the first **SubBytes** layer in state `S1` and the output difference δ_{OUT} after the last **MixBytes** layer in state `S12` in a way that the truncated characteristic holds in `S0` and `S12`. Note that since we have 8 active bytes in `S1` and `S12`, there are as many as $2^{2\times 64} = 2^{128}$ different ways of choosing $(\delta_{IN}, \delta_{OUT})$. We continue by constructing the 8 forward **SuperSBox** independently by considering the 2^{64} possible input values for each of them in state `S3`: differences in `S1` can be directly propagated to `S3` since **MixBytes** is linear. This generates 8 independent lists, each of size 2^{64} and composed by paired values. Doing the same for the 8 backwards **SuperSBoxes** from state `S12`, we again get 8 independent lists of 2^{64} elements each, and we end up in state `S8` where the 8 forward and the 8 backward lists overlap. In the sequel, we denote L_i the ith forward **SuperSBox** list and L'_i the ith backward one, for $1 \leq i \leq 8$.

In terms of freedom degrees in state `S8`, we want to merge 16 lists of 2^{64} elements each for a merging condition on $2 \times 512 = 1024$ bits (512 for values and 512 for differences): we then expect $2^{16\times 64}\, 2^{-1024} = 1$ solution as a result of the merging process. We detail a method in order to find this solution in time 2^{256} and memory 2^{64} (see Figure 3).

Step 1. We start by considering every possible combination of elements in each of the four lists L'_1, L'_2, L'_3 and L'_4. There are 2^{256} possibilities.

Step 2. This fully constraints 2×4 bytes in each of the 8 lists L_i, $1 \leq i \leq 8$ (i.e. the first 4 columns of the internal state). For each of them, we then expect $2^{64}\, 2^{-8\times 8} = 1$ element to match the randomized bytes. These elements can be found with one operation by sorting the lists L_i beforehand. At this point, note that the second half of the state `S8` has been fully determined by the choice in $L_1, \ldots, L_8$.

Step 3. We now need to ensure that the 4 last lists L'_5, L'_6, L'_7 and L'_8 contain the elements imposed: those lists being of size 2^{64} each, this happens with probability $2^{64}\, 2^{-8\times(2\times 8)} = 2^{-64}$ independently on each list. Again, these elements can be found with one operation by sorting the lists L'_i beforehand.

[1] These **SuperSBoxes** are 64-bit large in the case of `Grøstl`, but only $4 \times 8 = 32$ bits for the `AES`.

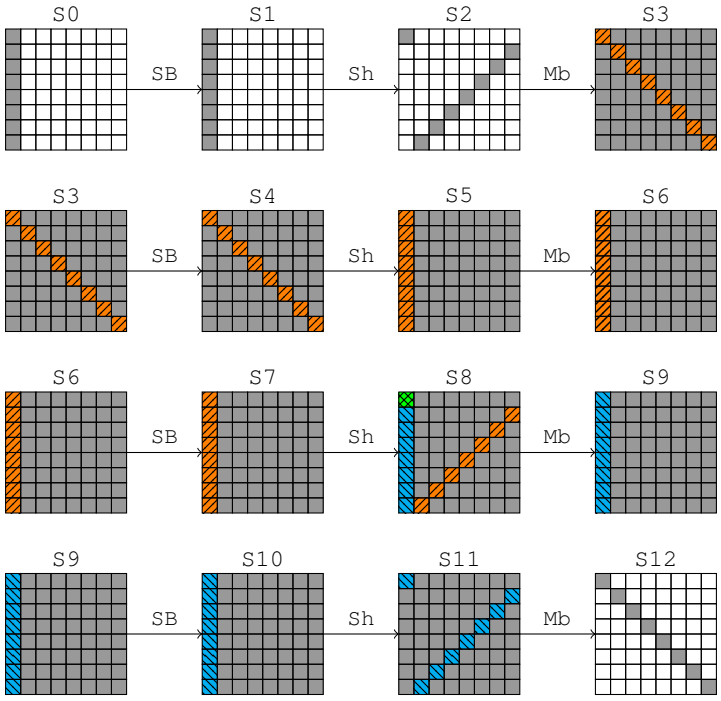
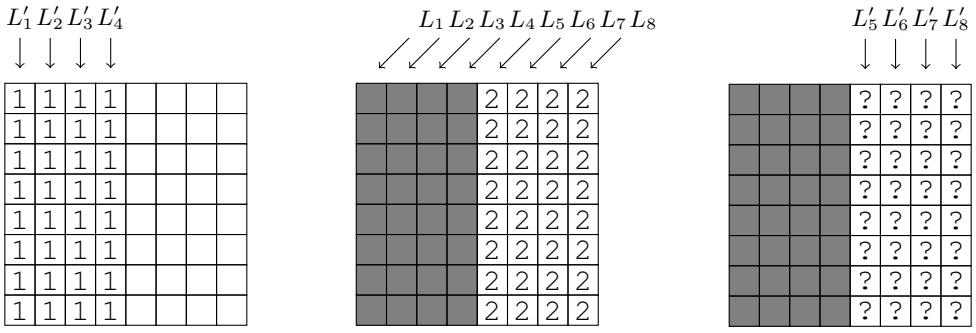


Figure 2: Inbound phase for the 9-round distinguisher attack on the `Grøstl` permutation P_{256}. The four rounds represented are the rounds 3 to 6 from the whole truncated differential characteristic. A gray byte indicates an active byte; hatched and coloured bytes emphasize one **SuperSBox**: there are seven similar others.



(a) Step 1. (b) Step 2. (c) Step 3.

Figure 3: Steps to merge the 16 lists. Grey cells denote bytes fully constrained by a choice of elements in $L'_1, \ldots, L'_4$ during the first step.

All in all, trying all the 2^{256} elements in (L'_1, L'_2, L'_3, L'_4), we expect to find $2^{256}\, 2^{-64\times 4} = 1$ solution that will verify the 1024 bits of condition and we can find this solution with only a few operations.

Hence, from random differences $(\delta_{IN}, \delta_{OUT})$, we find a pair of internal states of the permutation that conforms to the middle rounds in time 2^{256} and memory 2^{64}. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging 2^{112} times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation that conforms to the truncated differential characteristic in time complexity 2^{368} and memory complexity 2^{64}.

3.3 Comparison with ideal case

In the ideal case, obtaining a pair whose input and output differences lie in a subset of size $IN = OUT = 2^{64}$ for a 512-bit permutation requires 2^{384} computations: we can directly conclude that this leads to a distinguishing attack on the 9-round reduced version of the Grøstl-256 permutation with 2^{368} computations and 2^{64} memory. Similarly, as explained in Section 2.2, this result also induces a nontrivial observation on the 9-round reduced version of the Grøstl-256 compression function with identical complexity.

Finally, one can also derive slightly cheaper distinguishers by aiming less rounds: instead of using the 9-round truncated characteristic from Appendix B, it is possible to remove either round 2 or 8 and spare one $8 \rightarrow 1$ truncated differential transition. Overall, the generic complexity remains the same and this gives a distinguishing attack on the 8-round reduced version of the Grøstl-256 permutation with 2^{312} computations and 2^{64} memory. Unfortunately, this is worse than previously known results.

4 Distinguishers for reduced Grøstl-512 internal permutations

The 512-bit version of the Grøstl hash function uses a non-square 8×16 matrix as 1024-bit internal state, which therefore presents a lack of optimal diffusion: a single difference generates a fully active state after three rounds where a square-state would need only two. This enables us to add an extra round to the generalization of the regular 9-round characteristic of AES-like permutation (Section 3) to reach 10 rounds.

4.1 The truncated differential characteristic

To distinguish its permutation P_{512} [2] reduced to 10 rounds, we use the truncated differential characteristic with the sequence of active bytes

$$64 \xrightarrow{R_1} 8 \xrightarrow{R_2} 1 \xrightarrow{R_3} 8 \xrightarrow{R_4} 64 \xrightarrow{R_5} 128 \xrightarrow{R_6} 64 \xrightarrow{R_7} 8 \xrightarrow{R_8} 1 \xrightarrow{R_9} 8 \xrightarrow{R_{10}} 64.$$

where the size of the input differences subset is $IN = 2^{512}$ and the size of the output differences subset is $OUT = 2^{64}$.

The actual truncated characteristic is appended in Appendix C. Again, we split the characteristic into two parts: the inbound phase involving a merging of lists in the four middle rounds (round 4 to round 7), and an outbound phase that behaves as a probabilistic filter ensuring both $8 \longrightarrow 1$ transitions in the outward directions. Again, passing those two transitions with random values occurs with probability 2^{-112}.

4.2 Finding a conforming pair

In the following, we present an algorithm to solve the middle rounds in time 2^{280} and memory 2^{64}. In total, we will need to repeat this process 2^{112} times to get a pair of internal states that conforms to the whole truncated differential characteristic, which would then cost $2^{280+112} = 2^{392}$ in time and 2^{64} in memory. The strategy of this algorithm (see Figure 4) is similar to the ones presented in [14, 15] and the one from the previous section: we start by fixing the difference to a random value δ_{IN} in S1 and δ_{OUT} in S12 and linearly deduce the difference δ'_{IN} in S3 and δ'_{OUT} in S10. Then, we construct the 32 lists corresponding to the 32 **SuperSBoxes**: the 16 forward **SuperSBoxes** have an input difference fixed to δ'_{IN} and cover states S3 to S8, whereas the 16 backward **SuperSBoxes** spread over states S10 to S6 with an output difference fixed

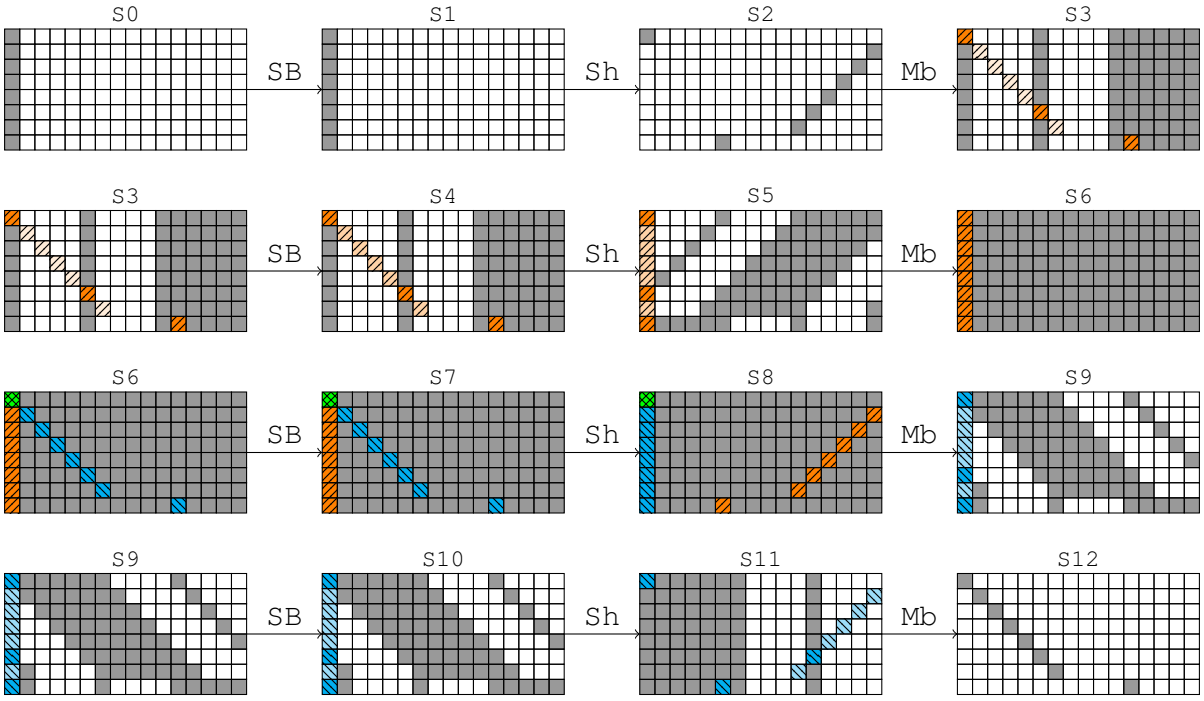


Figure 4: Inbound phase for the 10-round distinguisher attack on the Grøstl-512 permutation P_{512}. The four rounds represented are the rounds 4 to 7 from the whole truncated differential characteristic C. A gray byte indicates an active byte; hatched and coloured bytes emphasize the **SuperSBoxes**.

to δ'_{OUT}. In the sequel, we denote L_i the 16 forward **SuperSBoxes** and L'_i the backward ones, $1 \leq i \leq 16$.

The 32 lists overlap in S8, where we merge them on 2048 bits[3] to find $2^{64\times 32}\, 2^{-2048} = 1$ solution, since each list is of size 2^{64}. The naive way to find the solution would cost 2^{1024} in time by considering each element of the Cartesian product of the 16 lists L_i to check whether it satisfies the output 1024 bit difference condition. We describe now the algorithm that achieves the same goal in time 2^{280}.

First, we observe that due to the geometry of the non-square state, any list L_i intersects with only half of the L'_i. For instance, the first list L_1 associated to the first column of state S7 intersects with lists L'_1, L'_6, L'_{11}, L'_{12}, L'_{13}, L'_{14}, L'_{15} and L'_{16}. We represent this property with a 16×16 array on Figure 5: the 16 columns correspond to the 16 lists L'_i and the lines to the L_i, $1 \leq i \leq 16$. The cell (i, j) is white if and only if L_i has a non-null intersection with the list L'_j, otherwise it is gray.

Then, we note that the **MixBytes** transition between the states S8 and S9 constraints the differences in the lists L'_i : in the first column of S9 for example, only three bytes are active, so that the same column in S8 can only have $2^{3\times 8}$ different differences, which means that knowing three out of the eight differences in an element of L'_1 is enough to deduce the other five. For a column-vector of differences lying in a n-dimensional subspace, we can divide the 2^{64} elements of the associated lists in 2^{8n} disjointed sets of 2^{64-8n} values each. So, whenever we know the n independent differences, the only freedom that remains lie in the values. The bottom line of Figure 5 reports the subspace dimensions for each L'_i.

[2] It would work exactly the same way for the other permutation Q_{512}.
[3] The 2048 bits come from 1024 bits of values and 1024 bits of differences.

Using a guess-and-determine approach, we derive a way to use the previous facts to find the solution to the merge problem in time 2^{280}. As stated before, we expect only one solution; that is, we want to find a single element in each of the 32 lists. We start by guessing the values and the

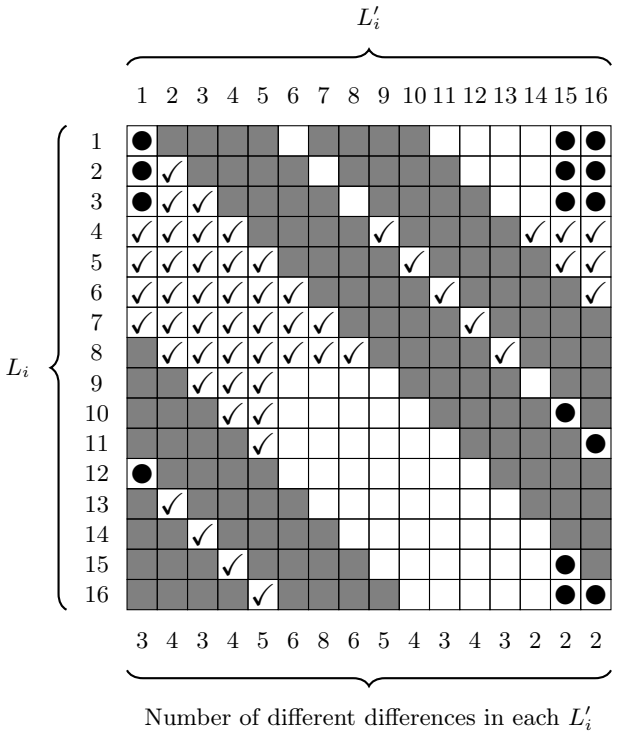
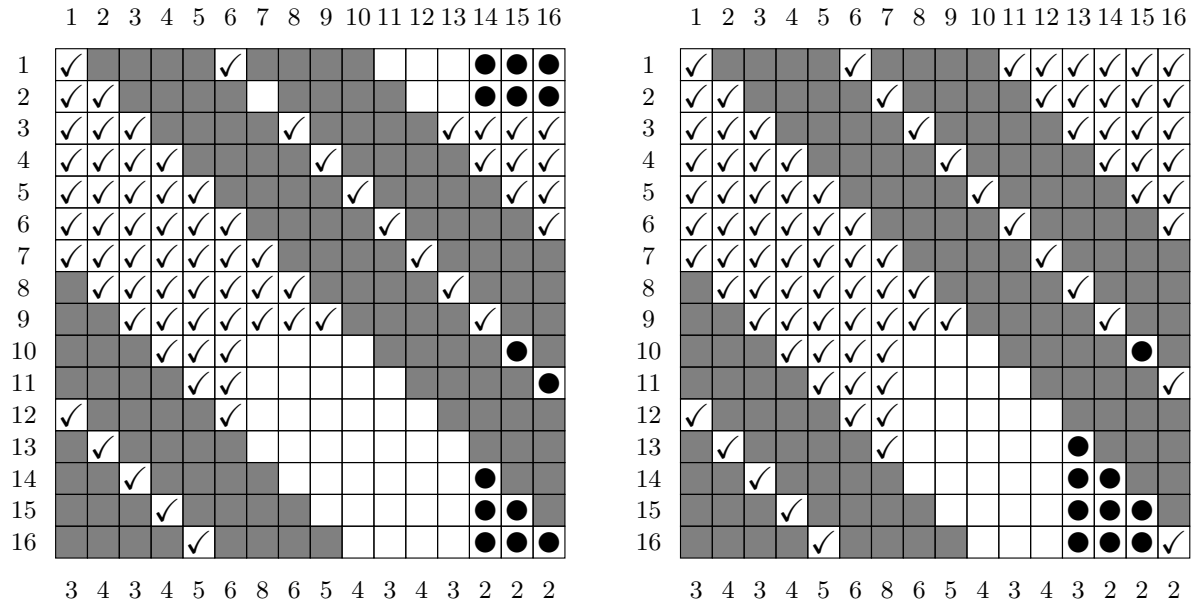


Figure 5: A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

differences of the elements associated to the lists L'_2, L'_3, L'_4 and L'_5. For this, we will try all the possible combinations of their elements, there are $2^{4\times 64} = 2^{256}$ in total. For each one of the 2^{256} tries, all the checked cells ✓ now have known value and difference. From here, 8 bytes are known in each of the four lists L_5, L_6, L_7 and L_8: this imposes a 64-bit constraint on those lists, which filter out a single element in each. Thereby, we determined the value and difference in the other 16 bytes marked by ✓ in Figure 5. In lists L'_1 and L'_{16}, we have reached the maximum number of independent differences (three and two, respectively), so we can determine the differences for the other bytes of those columns: we mark them by ●. In L_4, the 8 constraints (three ✓ and two ●) filter out one element; then, we deduce the correct element in L_4 and mark it by ✓. We can now determine the differences in L'_{15} since the corresponding subspace has a dimension equals to two.

At this point, no more byte can be determined based on the information propagated so far. We continue by guessing the elements remaining in L'_6. Since there are already six byte-constraints on that list (three ✓), only 2^{16} elements conform to the conditions. The time complexity until now is thus $2^{256+16} = 2^{272}$.

Guessing the list L'_6 implies a 64-bit constraint of the list L_9 so that we get a single element out of it and determine four yet-unknown other bytes. This enables to learn the independent differences in L'_{14} and therefore, we filter an element from L_3 (two ✓ and four ●). At this stage, the list L'_1 is already fully constrained on its differences, so that we are left with a set of



(a) End of the second guess.

(b) Near the end.

Figure 6: A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

$2^{64-3\times8} = 2^{40}$ values constrained on five bytes (five ✓). Hence, we are able to determine all the unset values in L'_1 (Figure 6a).

Again, the lack of constraints prevent us to determine more bytes. We continue by guessing the 2^8 elements left in L_1 (two ✓ and three ●), which makes the time complexity increase to 2^{280}. The list L_1 being totally known, we derive the vector of differences in L'_{13}, which adds an extra byte-constraint on L_2 where only one element was left, and so fully determines it. From here, L'_7 becomes fully determined as well (four ✓) and so is L_{16}. In the latter, the differences being known, we were left with a set of $2^{64-2\times8} = 2^{48}$ values, which are now constrained on six bytes (six ✓).

We describe in Figure 6b the knowledge propagated so far, with time complexity 2^{280} and probability 1. We observe that L_{10} is overdetermined (four ✓ and one ●) by one byte. This means that we get the correct value with probability 2^{-8}, whereas L_{11} is filtered with probability 1. Similarly, the element of L'_8 happens to be correctly defined with probability 2^{-16}; as for L'_9 and L'_{15}, with probability 1. We continue in L'_{11} by learning the full vector of differences, which constraints L_{12} on 11 bytes (five ✓ and one ●) so that we get a valid element with probability 2^{-24}. Finishing the guess and determine technique is done by filtering L'_{10} and L_{12} with probability 1, L_{16} with probability 2^{-40} and L_{13}, L_{14} and L_{15} with probability 2^{-64} each.

In total, for each guess, we successfully merge the 32 lists with probability

$$2^{-8-16-24-40-64-64-64} = 2^{-280},$$

but the whole procedure is repeated $2^{64\times4+16+8} = 2^{280}$ times, so we expect to find the one existing solution. All in all, we described a way to do the merge with time complexity 2^{280} and memory complexity 2^{64}. The final complexity to find a valid candidate for the whole characteristic is then 2^{392} computations and 2^{64} memory.

4.3 Comparison with ideal case

In the ideal case, obtaining a pair whose input difference lies in a subset of size $IN = 2^{512}$ and whose output difference lies in a subset of size $OUT = 2^{64}$ for a 1024-bit permutation requires 2^{448} computations. We can directly conclude that this leads to a distinguishing attack on the 10-round reduced version of the Grøstl-512 permutation with 2^{392} computations and 2^{64} memory. Similarly, as explained in Section 2.2, this results also induces a nontrivial observation on the 10-round reduced version of the Grøstl-512 compression function with identical complexity.

One can also derive slightly cheaper distinguishers by aiming less rounds while keeping the same generic complexity: instead of using the 10-round truncated characteristic from Appendix C, it is possible to remove either round 3 or 9 and spare one $8 \rightarrow 1$ truncated differential transition. Overall, this gives a distinguishing attack on the 9-round reduced version of the Grøstl-512 permutation with 2^{336} computations and 2^{64} memory. By removing both rounds 3 and 9, we achieve 8 rounds with 2^{280} computations.

One can further gain another small factor for the 9-round case by using a $8 \rightarrow 2$ truncated differential transition instead of $8 \rightarrow 1$, for a final complexity of 2^{328} computations and 2^{64} memory. Indeed, the generic complexity drops to 2^{384} because we would now have $OUT = 2^{128}$.

5 Conclusion

In this paper, we have provided new and improved cryptanalysis results on the building blocks of both 256 and 512-bit versions of the finalist Grøstl. This is done by using a rebound-like approach as well as an algorithm that allows us to pass three fully active states in the middle of the differential characteristic with lower complexity than a general probabilistic approach. To the best of our knowledge, all previously known methods only manage to control two fully active states in the middle of the differential characteristic.

On Grøstl-256, we could provide the best known rebound distinguishers on 9 rounds of the permutation. For Grøstl-512, we have considerably increased the number of analyzed rounds, from 7 to 10, providing the best analysis known the permutation.

These results do not threaten the security of Grøstl, but we believe they will have an important role in better understanding AES-based functions in general. In particular, we believe that our work will help determining the bounds and limits of rebound-like attacks in these types of constructions. Future works could include the study of more AES-like functions in regards to this new cryptanalysis method.

References

1. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order Differential Properties of Keccak and Luffa. In: FSE. Volume 6733 of LNCS., Springer (2011) 252–269
2. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate
3. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong, S., Iwata, T., eds.: FSE. Volume 6147 of Lecture Notes in Computer Science., Springer (2010) 365–383
4. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In Rogaway, P., ed.: CRYPTO. Volume 6841 of Lecture Notes in Computer Science., Springer (2011) 222–239
5. Jean, J., Fouque, P.A.: Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function. In Joux, A., ed.: FSE. Volume 6733 of Lecture Notes in Computer Science., Springer (2011) 107–127
6. Jean, J., Naya-Plasencia, M., Schläffer, M.: Improved Analysis of ECHO-256. In Miri, A., Vaudenay, S., eds.: Selected Areas in Cryptography. Volume 7118 of Lecture Notes in Computer Science., Springer (2011) 19–36
7. Knudsen, L.R.: Truncated and Higher Order Differentials. In Preneel, B., ed.: FSE. Volume 1008 of Lecture Notes in Computer Science., Springer (1994) 196–211

8. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. [9] 126–143
9. 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. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009)
10. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full LANE Compression Function. [9] 106–125
11. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Fast Software Encryption - FSE 2009. Volume 1008 of Lecture Notes in Computer Science., Springer (5665)
12. 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. Volume 5867 of Lecture Notes in Computer Science., Springer (2009) 16–35
13. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound Attacks on the Reduced Grøstl Hash Function. In Pieprzyk, J., ed.: CT-RSA. Volume 5985 of Lecture Notes in Computer Science., Springer (2010) 350–365
14. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010) (extended version). urlhttp://eprint.iacr.org/.
15. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Cryptology: CRYPTO 2011. Volume 6841 of Lecture Notes in Computer Science., Springer (2011) 188–205
16. Nikolic, I., Pieprzyk, J., Sokolowski, P., Steinfeld, R.: Known and Chosen Key Differential Distinguishers for Block Ciphers. In Rhee, K.H., Nyang, D., eds.: ICISC. Volume 6829 of Lecture Notes in Computer Science., Springer (2010) 29–48
17. Peyrin, T.: Cryptanalysis of Grindahl. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 551–567
18. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO. Volume 6223 of Lecture Notes in Computer Science., Springer (2010) 370–392
19. 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. Volume 6477 of Lecture Notes in Computer Science., Springer (2010) 38–55
20. Schläffer, M.: Updated Differential Analysis of Grøstl. Grøstl website (January 2011)
21. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
22. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 19–35

A Distinguishers for other AES-like permutations

Using the same cryptanalysis technique, it is possible to study other AES-like schemes using permutations similar to the Grøstl ones. For example, the recent lightweigth hash function family PHOTON [4] is based on five different versions of AES-like permutations. We denote s the size of the cells ($s = 8$ for AES) and c the size of the square matrix representing the internal state ($c = 4$ for AES), the five versions (s, c) for PHOTON are then $(4, 5)$, $(4, 6)$, $(4, 7)$, $(4, 8)$ and $(8, 6)$ for increasing versions. All versions are defined to apply 12 rounds of an AES-like process, where the subkey additions are replaced by constant additions. Since the internal state is always square, by trivially adapting the method from Section 3 to the specific parameters of PHOTON, one can hope to obtain distinguishers for 9 rounds of the PHOTON internal permutations. However, we are able to do so only for the parameters $(4, 8)$ used in PHOTON-224/32/32 (see Table 2 with the comparison to previously known results). Indeed, the size c of the matrix plays an important role in the gap between the complexity of our algorithm and the generic one. The bigger is the matrix, the better will be the gap between the algorithm complexity and the generic one.

The same effect applies on AES in the known-key model, for which distinguishers on only 8 rounds are known as of today [3]. When attacking 9 rounds with the method from Section 3, the middle rounds will cost about 2^{64} operations per solution, while the two $4 \rightarrow 1$ truncated differential transitions during the outbound will be verified with probability $(2^{-24})^2 = 2^{-48}$.

Target	Subtarget	Rounds	Time	Memory	Ideal	Reference
PHOTON-224/32/32	permutation	8 (dist.)	2^8	2^4	2^{10}	[4]
		9 (dist.)	2^{184}	2^{32}	2^{192}	Section A

Table 2: Distinguishers on PHOTON internal permutation when applying the method from Section 3.

Overall, one solution for the whole characteristic is found with 2^{112} computation and 2^{32} memory, but the generic algorithm can find such a pair with only 2^{64}.

B 9-round Grøstl-256 permutation truncated characteristic

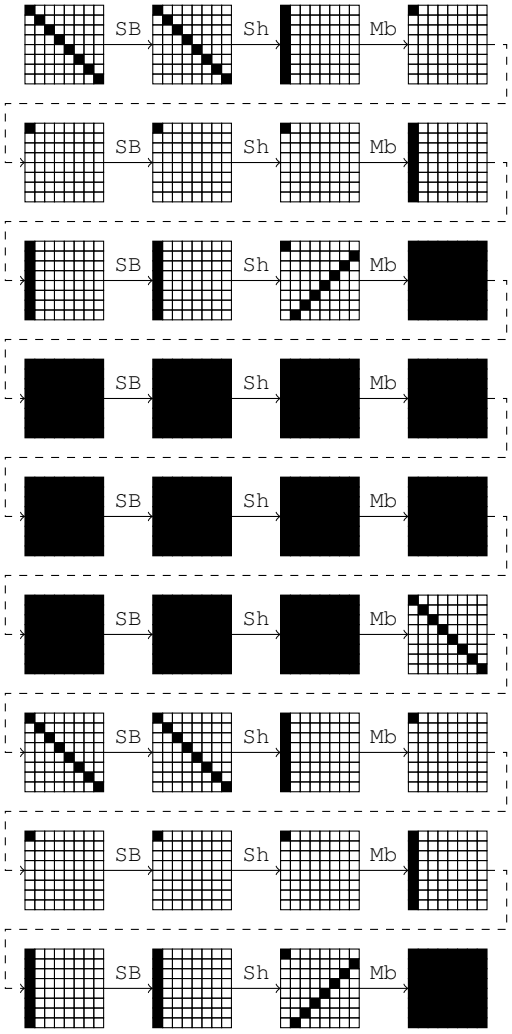


Figure 7: The 9-round truncated differential characteristic used to distinguish the permutation P of Grøstl-256 from an ideal permutation.

C 10-round Grøstl-512 permutation truncated characteristic

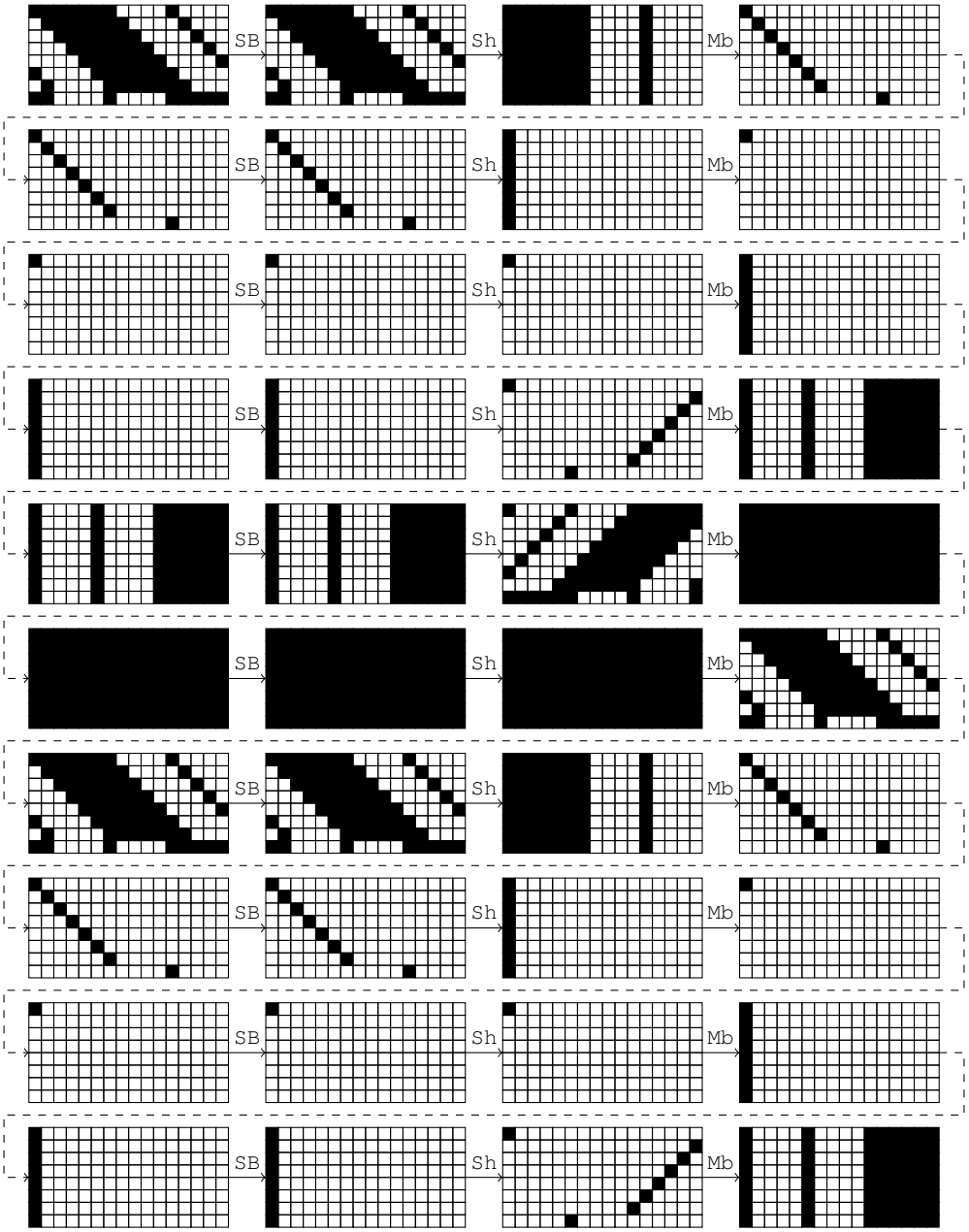


Figure 8: The 10-round truncated differential characteristic used to distinguish the permutation P of Grøstl-512 from an ideal permutation.