Improved Cryptanalysis of AES-like Permutations

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

[1] École Normale Supérieure, France
[2] INRIA Paris-Rocquencourt, France
[3] Nanyang Technological University, Singapore

Abstract. AES-based functions have attracted of a lot of analysis in the recent years, mainly due to the SHA-3 hash function competition. In particular, the rebound attack allowed to break several proposals and many improvements/variants of this method have been published. Yet, it remained an open question whether it was possible to reach one more round with this type of technique compared to the state-of-the-art. In this article, we close this open problem by providing a further improvement over the original rebound attack and its variants, that allows the attacker to control one more round in the middle of a differential path for an AES-like permutation. Our algorithm is based on lists merging as defined in [21] and we generalized the concept to non-full active truncated differential paths [25].
As an illustration, we applied our method to the internal permutations used in Grøstl, one of the five finalist hash functions of the SHA-3 competition. When entering this final phase, the designers tweaked the function so as to thwart attacks from Peyrin [24] that exploited relations between the internal permutations. Until our results, no analysis was published on Grøstl and the best results reached 8 and 7 rounds for the 256-bit and 512-bit version respectively. By applying our algorithm, we present new internal permutation distinguishers on 9 and 10 rounds respectively.

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

1 Introduction

Hash functions are one of the most important primitives in symmetric-key cryptography. They are simply functions that, given an input of variable length, produce an output of a fixed size. They are needed in several scenarios, like integrity check, authentication, digital signatures, so we want them to verify some security properties, for instance: preimage resistance, collision resistance (i.e., for a n-bit hash function, finding two distinct inputs mapping to the same output should require at least $2^{n/2}$ computations), second preimage resistance, and so on.

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. [26, 27]. Due to the resemblance of the standard SHA-2 with SHA-1, the confidence in the former was also somewhat undermined. This is why the American National Institute of Standards and Technology (NIST) decided to launch in 2008 a competition in order to find a new hash standard, SHA-3. This competition received 64 hash function submissions and accepted 51 to enter the first round. Three years and two rounds later, only 5 hash functions remained in the final phase of the competition.

Among the candidates, many functions were AES-based (they reuse some AES components or the general AES design strategy), like the SHA-3 finalist Grøstl [6]. This design trend is at the origin of the introduction of the rebound attack [17], 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 AES-based constructions (such as block ciphers in the known/chosen-key model). It 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.

The rebound attack was proposed as a method to derive a pair of internal states that verifies some truncated differential path with lower complexity than a generic attack. It was formed by two steps: a first one, *the controlled part (or inbound)*, where solutions for two rounds of an unkeyed AES-like permutation were found with negligible complexity, and a second one *uncontrolled part (or outbound)*, where the solutions found during the inbound phase were used to verify probabilistically the remaining differential transitions. Assuming an AES-like internal state composed of a $t \times t$ matrix of c-bit cells, the rebound attack was then extended to three rounds by the start-from-the-middle [18] and the **SuperSBox** variants [7, 14] for a negligible average complexity per found pair, but with a higher minimal complexity of $2^{t \cdot c}$ computations. Since most rebound-based attacks actually required many such pairs, this was not much of a constraint. In parallel, other improvements on the truncated differential paths utilized [25] or on methods to merge lists [21] were proposed.

In this article, we describe a method based on lists merging in order to control truncated differences over four rounds of an unkeyed AES-like permutation [11] with complexity $2^{t \cdot c \cdot x}$ computations, where x is a parameter depending on the differential path considered. While the cost per pair found in the controlled part is much increased, solving four rounds directly allows to handle much better truncated differential paths for the uncontrolled part. Note that whether it was possible or not to reach four rounds remained an open problem among the research community. We also generalize the global reasoning by considering as well non-fully-active truncated differential paths [25] during both the controlled and uncontrolled phases, eventually obtaining the best known results for many attack scenarios of an AES-like permutation.

As an application, we concentrated our efforts on the Grøstl internal permutation. Rebound-like attacks on this function have already been applied and improved in several occasions [7, 18, 19, 21, 24], Grøstl being one of the most studied SHA-3 candidates. When entering the final round, a tweak of the function was proposed, which prevents the application of the attacks from [24]. We denote Grøstl-0 the original submission [5] of the algorithm and Grøstl its tweaked version [6]. Apart from the rebound results, the other main analysis communicated on Grøstl is a higher order property on 10 rounds of its internal permutation [3] with a complexity of 2^{509} computations. In Table 1, we give a summary of the best known results on both the 256 and 512-bit tweaked versions of Grøstl, including the ones that we present in this article.

Namely, we provide the best known rebound distinguishers on 9 rounds of the internal permutation and we show how to make some nontrivial observations on the corresponding compression function, providing the best known analysis of the Grøstl compression function exploiting the properties of the internal permutations. For Grøstl-512, we considerably increase the current largest number of analyzed rounds, from 7 to 10. Additionally, we provide in Appendix the direct application of our new techniques to the AES-based hash function PHOTON [8].

These results do not threaten the security of Grøstl, but we believe they will have an important role in better understanding its security, 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 this type of constructions.

Target	Subtarget	Rounds	Time	Memory	Ideal	Reference
Grøstl-256	Permutation	8 (dist.)	2^{112}	2^{64}	2^{384}	[7]
		8 (dist.)	2^{48}	2^{8}	2^{96}	[25]
		9 (dist.)	$\mathbf{2^{368}}$	$\mathbf{2^{64}}$	$\mathbf{2^{384}}$	**Section 3**
		10 (zero-sum)	2^{509}	–	2^{512}	[3]
Grøstl-512	Permutation	7 (dist.)	2^{152}	2^{56}	2^{512}	[25]
		8 (dist.)	$\mathbf{2^{280}}$	$\mathbf{2^{64}}$	$\mathbf{2^{448}}$	**Appendix A**
		9 (dist.)	$\mathbf{2^{328}}$	$\mathbf{2^{64}}$	$\mathbf{2^{384}}$	**Appendix A**
		10 (dist.)	$\mathbf{2^{392}}$	$\mathbf{2^{64}}$	$\mathbf{2^{448}}$	**Appendix A**
PHOTON-224/32/32	Permutation	8 (dist.)	2^{8}	2^{4}	2^{10}	[8]
		9 (dist.)	$\mathbf{2^{184}}$	$\mathbf{2^{32}}$	$\mathbf{2^{192}}$	**Appendix B**

Table 1: Best attacks on targets where our analysis is applicable. By best analysis, we mean the ones on the highest number of rounds.

2 Generalities

In this section, we start by describing a generic view of an AES-like permutation to capture various cryptographic primitives such as AES [4], Grøstl [5], ECHO [2], Whirlpool [1], LED [9], PHOTON [8].

2.1 Description of AES-like Permutations

We define an AES-like permutation as a permutation that applies N_r rounds of a round function to update an internal state viewed as a square matrix of t rows and t columns, where each of the t^2 cells has a size of c bits. As we will show later, our techniques can also be adapted when the matrix is not square (as it is the case for Grøstl-512), but we focus on square matrices for ease of description.

The round function (Figure 1) starts by xoring a round-dependent constant to the state in the **AddRoundConstant** operation (AC). Then, it applies a substitution layer **SubBytes** (SB) which relies on a $c \times c$ non-linear bijective SBox. Finally, the round function performs a linear layer, composed of the **ShiftRows** transformation (SR), that moves each cell belonging to the x-th row by x positions to the left in its own row, and the **MixCells** transformation (MC), that linearly mixes all the columns C of the matrix separately by multiplying each one with a matrix $\mathbf{M}$ implementing a Maximum Distance Separable (MDS) code: $C \leftarrow \mathbf{M} \times C$.

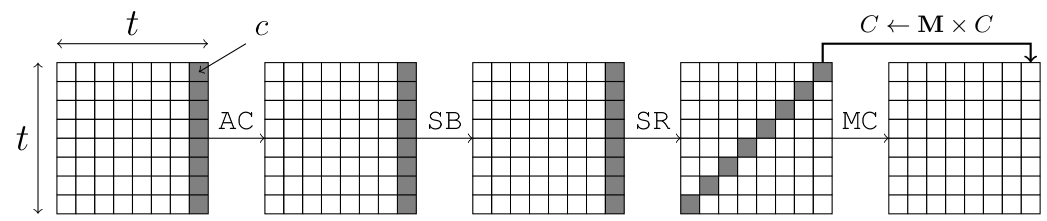


Figure 1: One round of the AES-like permutation instantiated with $t = 8$.

Note that this description encompasses permutations that really follow the AES design strategy, but very similar designs (for example with a slightly modified **ShiftRows** function or with a **MixCells** layer not implemented with an MDS matrix) are likely to be attacked by our techniques as well. In the case of `AES`-like block ciphers analyzed in the known/chosen-key model, the subkeys generated by the key schedule are incorporated into the known constant addition layer **AddRoundConstant**. We note that all the rounds considered in this article are *full* rounds: they all have the **MixCells** transformation, even the last one as opposed to the full version of the `AES`.

2.2 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 one. For the final round of the competition, the candidate has been tweaked to `Grøstl`, with corresponding versions `Grøstl-256` and `Grøstl-512`.

The `Grøstl` hash function handles messages[1] by dividing 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 are built upon 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 state is ensured to be at least twice as large as the final digest.

The compression function f_{256} of `Grøstl-256` uses two 256-bit permutations P_{256} and Q_{256}, which are similar to the two 512-bit 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 2) produces the output ($\oplus$ denotes the `XOR` operation):

$$f_{256}(h, m) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus h, \tag{1}$$

$$\text{or:} \quad f_{512}(h, m) = P_{512}(h \oplus m) \oplus Q_{512}(m) \oplus h. \tag{2}$$

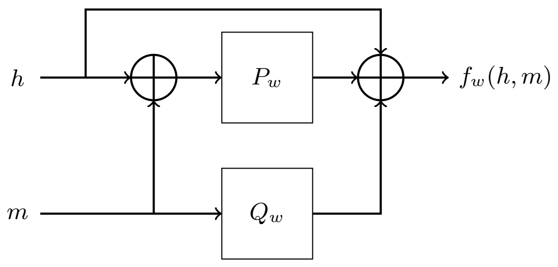


Figure 2: The compression function of `Grøstl` using the permutations P_w and Q_w, with $w \in \{256, 512\}$.

The internal states are viewed as matrices of bytes 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:

$$R \overset{\text{def}}{:=} \textbf{MixCells} \circ \textbf{ShiftBytes} \circ \textbf{SubBytes} \circ \textbf{AddRoundConstant}. \tag{3}$$

[1] Messages are of maximal bit-length $2n \cdot (2^{64} - 1) - 64 - 1$ for `Grøstl`-n, with $n \in \{256, 512\}$.

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 [24] that takes advantage of the similarities between 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 cells of the internal state. The **ShiftBytes** (Sh) transformation shifts cells 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, **MixCells** (MC) is implemented in Grøstl by the **MixBytes** (Mb) operation that applies a circulant MDS 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 $h' = \mathrm{trunc}_n(P(h) \oplus h)$, where trunc_n only keeps the last n bits.

2.3 Distinguishers

In this article, we describe algorithms that find input pairs (X, X') for an AES-like permutation P, 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 [14] where linear subspaces are considered) in order to solve this problem, known as limited-birthday problem, has been given in [7] and later a very close lower bound has been proven in [22]. For a randomly chosen n-bit permutation π, the generic algorithm can find such a pair with complexity

$$\max\left\{\min\left\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\right\}, 2^n/(IN \cdot OUT)\right\}. \tag{4}$$

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:

$$f(h,m) \oplus f(m,h) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus P_{256}(m \oplus h) \oplus Q_{256}(h) \oplus h \oplus m, \tag{5}$$
$$f(h,m) \oplus f(m,h) = Q_{256}(m) \oplus Q_{256}(h) \oplus h \oplus m. \tag{6}$$

Hence, it follows that:

$$f(h,m) \oplus f(m,h) \oplus h \oplus m = Q_{256}(m) \oplus Q_{256}(h). \tag{7}$$

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.

2.4 Truncated differential characteristics

In the following, we will consider truncated differential characteristics, originally introduced by Knudsen [13] for block cipher analysis. With this technique, already proven to be efficient for `AES`-based hash functions cryptanalysis [10, 12, 16, 17, 23], the attacker only checks if there is a difference in a cell (active cell, denoted by a black square in the Figures) or not (inactive cell, denoted by an empty square in the Figures) without caring about the actual value of the difference.

In this model, all **AddRoundConstant** and **SubBytes** layers can be ignored since they have no impact on truncated differences. **ShiftBytes** will only move the difference positions and the diffusion will come from the **MixCells** layers. More precisely, we denote $x \rightarrow y$ a non-null truncated differential transition mapping x active cells to y active cells in a column through a **MixCells** (or **MixCells^{-1}**) layer, and the MDS property ensures $x + y \geq t + 1$. Its differential probability is determined by the number $(t - y)$ of inactive cells on the output: $2^{-c(t-y)}$ if the MDS property is verified, 0 otherwise.

3 Distinguishers for `AES`-like permutations

In this section, we describe a distinguisher for 9 rounds of an `AES`-like permutation with certain parameters t and c. For the sake of clarity, we first describe the attack for a truncated differential characteristic with three fully active states in the middle, but we will generalize our method in the next section by introducing a characteristic parameterized by variables controlling the number of active cells in some particular states.

Let us remark that before our work, the best known such distinguishers on this type of constructions could only reach 8 rounds, being an open problem whether reaching more rounds would be possible.

3.1 A first truncated differential characteristic

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

$$t \xrightarrow{R_1} 1 \xrightarrow{R_2} t \xrightarrow{R_3} t^2 \xrightarrow{R_4} t^2 \xrightarrow{R_5} t^2 \xrightarrow{R_6} t \xrightarrow{R_7} 1 \xrightarrow{R_8} t \xrightarrow{R_9} t^2, \qquad (8)$$

where the size of the input and output difference subsets are both $IN = OUT = 2^{ct}$, since there are t active c-bit cells in the input of the truncated characteristic, and the t^2 active cells in the output are linearly generated from only t active cells. The actual truncated characteristic instantiated with $t = 8$ is described in Figure 3.

Note that we have three fully active internal states in the middle of the differential characteristic, and this kind of path is impossible to solve with previous rebound or **SuperSBox** techniques since the number of controlled rounds would be too small and the cost for the uncontrolled part would be extremely high.

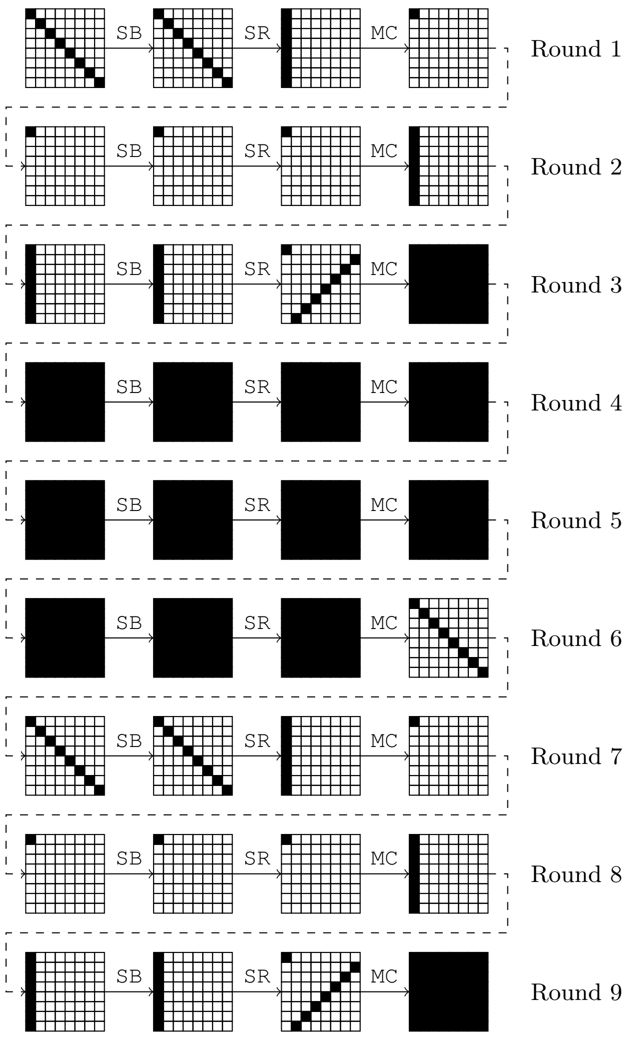


Figure 3: The 9-round truncated differential characteristic used to distinguish an AES-like permutation from an ideal permutation.

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 (beginning of round 3 to the end of round 6) and then we filter them out during the outwards probabilistic transitions through the **MixCells** layers (round 2 and round 7). Since in our case we have two **MixCells** transitions $t \to 1$ (see Figure 3), the outbound phase has a success probability of $2^{-2c(t-1)}$ 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 4), we propose an algorithm inspired by the ones in [20, 21]. As in [7, 14], instead of dealing with the classical t^2 parallel c-bit **SubBytes** SBox applications, one can consider t parallel tc-bit SBoxes (named **SuperSBoxes**) each composed of two SBox layers surrounding one **MixCells** and one **AddRoundConstant** function. Indeed, the **ShiftBytes** can be taken out from the **SuperSBoxes** since it commutes with **SubBytes**. The part of the internal state modified by one **SuperSBox** is a **SuperSBox** set. The total state is formed by t such sets, and their particularity is that their transformation through the **SuperSBox** can be computed independently.

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 **MixCells** layer in state `S12`. Both δ_{IN} and δ_{OUT} are exact differences, not truncated ones, but they are chosen so that they are compliant with the truncated characteristic in `S0` and `S12`. Since we have t active cells in `S1` and `S12`, there are as many as 2^{2ct} different ways of choosing $(\delta_{IN}, \delta_{OUT})$. Note that differences in `S1` can be directly propagated to `S3` since **MixCells** is linear. We continue by computing the t forward **SuperSBox** sets independently by considering the 2^{ct} possible input values for each of them in state `S3`. This generates t independent lists, each of size 2^{ct} and composed by paired values in `S3` (that can be used to compute the corresponding paired values in `S8`). Doing the same for the t backwards **SuperSBox** sets from state `S12`, we again get t independent lists of 2^{ct} elements each, and we can compute for each element of each list the pair of values of the **SuperSBox** set in state `S8`, where the t forward and the t backward lists overlap. In the sequel, we denote L_i the ith forward **SuperSBox** list and L'_i the ith backward one, for $1 \leq i \leq t$.

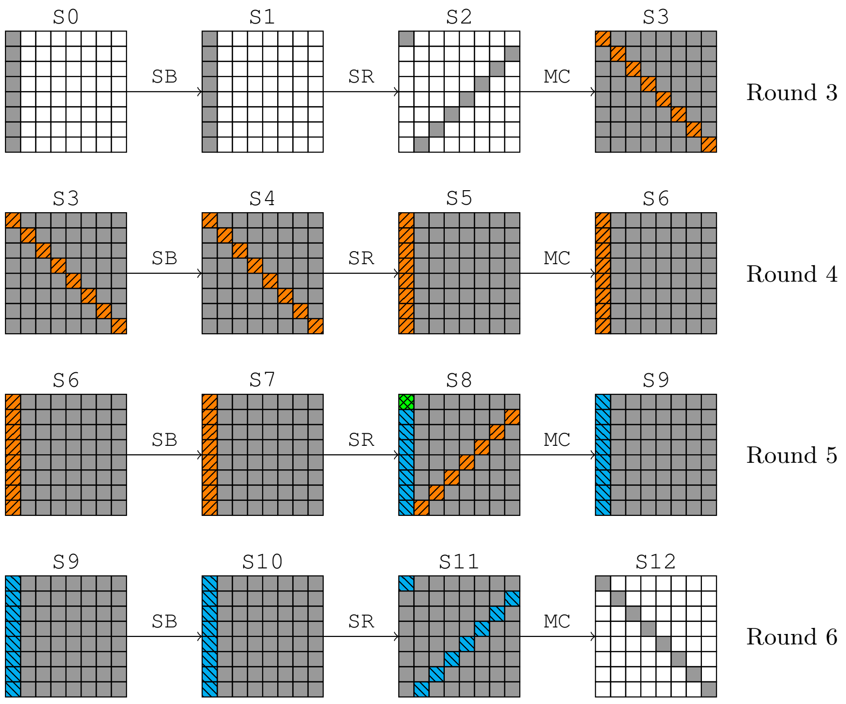


Figure 4: Inbound phase for the 9-round distinguisher attack on an AES-like permutation instantiated with $t = 8$. The four rounds represented are the rounds 3 to 6 from the whole truncated differential characteristic. A gray cell indicates an active cell; hatched and colored cells emphasize one **SuperSBox** set: there are seven similar others for each one of the two hatched senses.

In terms of freedom degrees in state `S8`, we want to merge $2t$ lists of 2^{ct} elements each for a merging condition on $2 \times ct^2$ bits, where we use the definition of list merging from [21], (ct^2 for values and ct^2 for differences) since the merging state is fully active: we then expect $2^{2t \times ct}\, 2^{-2ct^2} = 1$ solution as a result of the merging process on average.

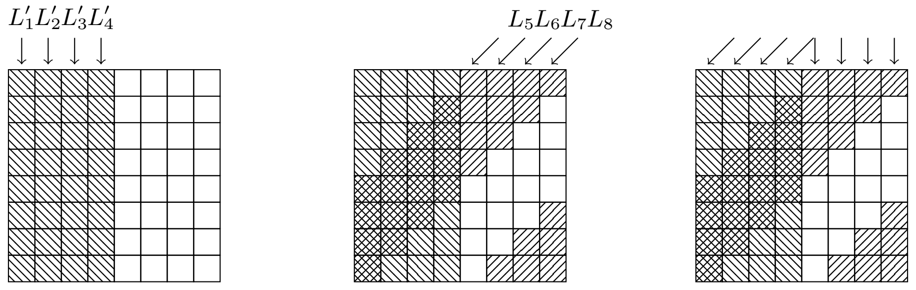
In the following, we describe a method to find this solution and compute its complexity afterwards. In comparison to the algorithm suggested in [11] where the case $t = 8$ is treated, we generalize the concept to any t, even odd ones where the direct extension of [11] is not applicable. To detail this algorithm, we use a temporary parameter $t' \in [1, t]$ such that the time complexity will be written in terms of t'. In the end, we give the best choice for t' such that the time complexity is minimal for any t.

Step 1. We start by considering every possible combination of elements in each of the t' first lists $L'_1, \ldots, L'_{t'}$. There are $2^{c \cdot t \cdot t'}$ possibilities.

Step 2. Each choice in Step 1 fixes the first t' columns of the internal state (both values and differences) and thus forces $2c$ constraints on t' cells in each of the t lists L_i, $1 \leq i \leq t$. For each list L_i, we then expect on average $2^{ct}\, 2^{-2ct'} = 2^{\,c(t-2t')}$ elements to match this constraint for each choice in Step 1, and these elements can be found with one operation by sorting the lists L_i beforehand[2].

Step 3. We continue by considering every possible combination of elements in each of the $t - t'$ last lists $L_{t-t'+1}, \ldots, L_t$. Depending on the value of t', we may have different scenarios at this point: if $t - 2t' \geq 0$, then the time complexity is multiplied by $2^{\,c(t-2t')(t-t')}$, which is the number of expected elements in the lists. Otherwise, the probability of success decreases from 1 to $2^{\,c(t-2t')(t-t')}$, as the constraints imposed by the previous step are too strong and elements in those lists would exist only with probability smaller than 1.

Step 4. We now need to ensure that the t' first lists $L_1, \ldots, L_{t'}$ and the $t - t'$ last lists $L'_{t-t'+1}, \ldots, L'_t$ contain a candidate fulfilling the constraints deduced in the previous steps. In the L'_i lists, we already determined $2c(t - t')$ bits so that there are $2^{ct-2c(t-t')}$ elements remaining in each of those. Again, we can check if these elements exist with one operation by sorting the lists beforehand. Finally, the value and difference of all the cells have been determined, which leads to a probability $2^{ct-2ct} = 2^{-ct}$ of finding a valid element in each of the t' first lists L_i.



(a) Step 1. (b) Steps 2 and 3. (c) Step 4.

Figure 5: In the case where $t = 8$, the figure shows the steps to merge the $2 \times t$ lists. Grey cells denote cells fully constrained by a choice of elements in $L'_1, \ldots, L'_{t/2}$ during the first step.

All in all, trying all the $2^{c \cdot t \cdot t'}$ elements in Step 1, we find

$$2^{\,c \cdot t \cdot t' + c(t-2t')(t-t') + (ct-2c(t-t'))(t-t') - ct \cdot t'} = 1$$

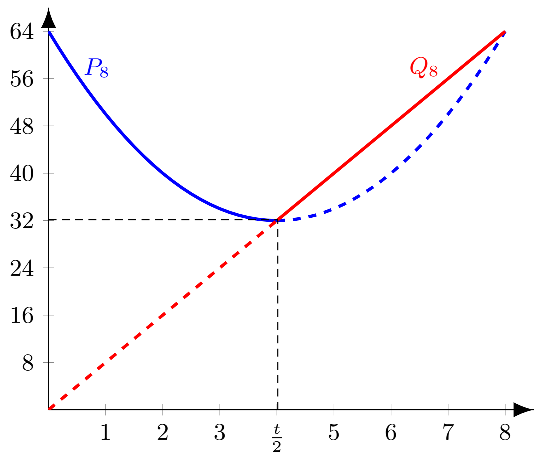
solution during the merge process. We find this solution in time T_m operations, with two cases to consider. Either $t - 2t' \geq 0$, in which case we enumerate $2^{\,c \cdot t \cdot t'}$ elements in

[2] We consider lists for the sake of clarity, but we can reach the constant-time access of elements using hash tables. Otherwise, it would introduce a logarithmic factor.

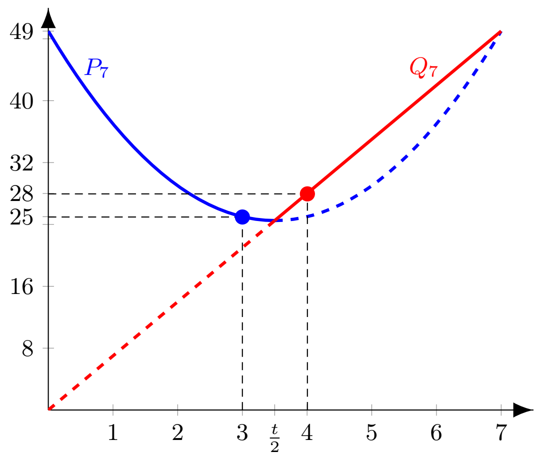
Step 1 followed by the enumeration of $2^{c(t-2t')}$ elements in Step 2. In that case, we have $\log_2(T_m) = ctt' + c(t-2t')(t-t') = 2t'^2 - 2tt' + t^2$. If $t - 2t' \leq 0$, the conditions imposed by the elements enumerated in the first steps make the lists from Step 2 to be nonempty with probability smaller than 1. Hence, we simply have $\log_2(T_m) = ctt'$. This can be summarized by:

$$\log_2(T_m) = \begin{cases} c \cdot P_t(t') & \text{if } t - 2t' \geq 0 \text{ with } P_t = 2X^2 - 2tX + t^2, \\ c \cdot Q_t(t') & \text{if } t - 2t' \leq 0 \text{ with } Q_t = tX. \end{cases} \tag{9}$$

To find the value t' that minimizes the time complexity, we need to determine for which value the minimum of both polynomials P_t and Q_t is reached. For P_t, we get $\frac{t}{2}$ and the nearest integer value satisfying $t - 2t' \geq 0$ is $\lceil \frac{t}{2} \rceil$. For Q_t, we get $\lfloor \frac{t}{2} \rfloor$. For example, see Figure 6a and Figure 6b, when t equals 8 and 7 respectively.



(a) $t = 8$



(b) $t = 7$

Figure 6: Plot of the two polynomials P_t and Q_t in two cases: $t = 8$ and $t = 7$.

Consequently, if t is even we set $t' = \frac{t}{2}$, which leads to an algorithm running in $2^{ct^2/2}$ operations and $t' \cdot 2^{ct}$ memory. If t is odd, then we need to decide whether t' should be $\lfloor \frac{t}{2} \rfloor$ or $\lceil \frac{t}{2} \rceil$. If we write $t = 2k+1$, this is equivalent to find the smallest value between $P_t(k)$ and $Q_t(k+1)$. We find $P_t(k) = 2k^2 + 2k + 1$ and $Q_t(k+1) = 2k^2 + 3k + 1$ so that $P_t(k) < Q_t(k+1)$ (see for example Figure 6b when $t = 7$). Hence, when t is odd, we fix $t' = \lceil \frac{t}{2} \rceil$. Note that $\frac{t}{2} = \lceil \frac{t}{2} \rceil$ if t is even.

Summing up, for any t, our algorithm performing the merge runs in T_m operations, with:

$$\log_2(T_m) = c \cdot P_t\left(\left\lceil \frac{t}{2} \right\rceil\right) = ct^2 - 2c \left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil \tag{10}$$

and a memory requirement of $2t' \cdot 2^{ct}$.

Hence, from a pair of random fixed differences $(\delta_{IN}, \delta_{OUT})$, we show how to find a pair of internal states of the permutation that conforms to the middle rounds. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging $2^{2c(t-1)}$ 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

$T_9 = 2^{2c(t-1)} \cdot T_m$ operations, that is:

$$\log_2(T_9) = ct(t+2) - 2c\left(\left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{11}$$

with a memory requirement of $t \cdot 2^{ct}$.

3.3 Comparison with the ideal case

In the ideal case [7], obtaining a pair whose input and output differences lie in a subset of size $IN = OUT = 2^{ct}$ for a ct^2-bit permutation requires

$$2^{\max\left\{\frac{ct(t-1)}{2},\, ct^2 - ct - ct\right\}} = 2^{ct(t-2)}, \tag{12}$$

computations (assuming $t \geq 3$). Therefore, our algorithm distinguishes an AES-like permutation from a random one if and only if its time complexity is smaller than the generic one. This occurs when $\log_2(T_9) \leq ct(t-2)$, which happens as soon as $t \geq 8$. Note that for the AES in the known-key model, we have $t = 4$ and thus our attack does not apply.

One can also derive slightly cheaper distinguishers by aiming at less rounds: instead of using the 9-round truncated characteristic from Figure 3, it is possible to remove either round 2 or 8 and spare one $t \to 1$ truncated differential transition. Overall, the generic complexity remains the same and this gives a distinguishing attack on the 8-round reduced version of the AES-like permutation with T_8 computations, with:

$$\log_2(T_8) = \log_2(T_m) + c(t-1) = ct(t+1) - c\left(2\left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{13}$$

and still 2^{ct} memory provided that $t \geq 6$. If we spare both $t \to 1$ transitions, we end up with a 7-round distinguishing attack with time complexity $T_7 = T_m$ and $t \cdot 2^{ct}$ memory for any $t \geq 4$. Note that those reduced versions of this attack can have a greater time complexity than other techniques: we provide them only for the sake of completeness.

4 Using non-fully-active characteristics

4.1 The generic truncated characteristic

In [25], Sasaki et al. present new truncated differential characteristics that are not totally active in the middle. Their analysis allows to derive distinguishers for 8 rounds of AES-like permutations with no totally-active state in the middle, provided that the state-size verifies $t \geq 5$. In this section, we reuse their idea by introducing an additional round in the middle of their trail, which is the unique fully active state of the characteristic. With a similar algorithm as in the previous section, we show how to find a pair conforming to that case.

To keep our reasoning as general as possible, we parameterize the truncated differential characteristic by four variables (see Figure 7) such that trade-offs will be possible by finding the right values for each one of them. Namely, we denote n_B the number of active diagonals in the plaintext (alternatively, the number of active cells in the second round), n_F the number of active diagonals in the ciphertext (alternatively, the number of active cells in the eighth round), m_B the number of active cells in the third round and m_F the number of active cells in the seventh round.

Hence, the sequence of active cells in the truncated differential characteristic becomes:

$$t\,n_B \xrightarrow{R_1} n_B \xrightarrow{R_2} m_B \xrightarrow{R_3} t\,m_B \xrightarrow{R_4} t^2 \xrightarrow{R_5} t\,m_F \xrightarrow{R_6} m_F \xrightarrow{R_7} n_F \xrightarrow{R_8} t\,n_F \xrightarrow{R_9} t^2, \qquad (14)$$

with the constraints $n_F + m_F \geq t+1$ and $n_B + m_B \geq t+1$ that come from the MDS property. The amount of solutions that can be generated for the differential path equals to ($\log_2$):

$$ct^2 + ctn_B - c(t-1)n_B - c(t-m_B) - ct(t-m_F) - c(t-1)m_F - c(t-n_F) \qquad (15)$$

$$= c(n_B + n_F + m_B + m_F - 2t). \qquad (16)$$

From the MDS constraints $m_B + n_B \geq t+1$ and $m_F + n_F \geq t+1$, we can bound the amount of expected solutions by $2^{c(t+1+t+1-2t)} = 2^{2c}$. This means that, there will always be at least 2^{2c} freedom degrees, independently of t.

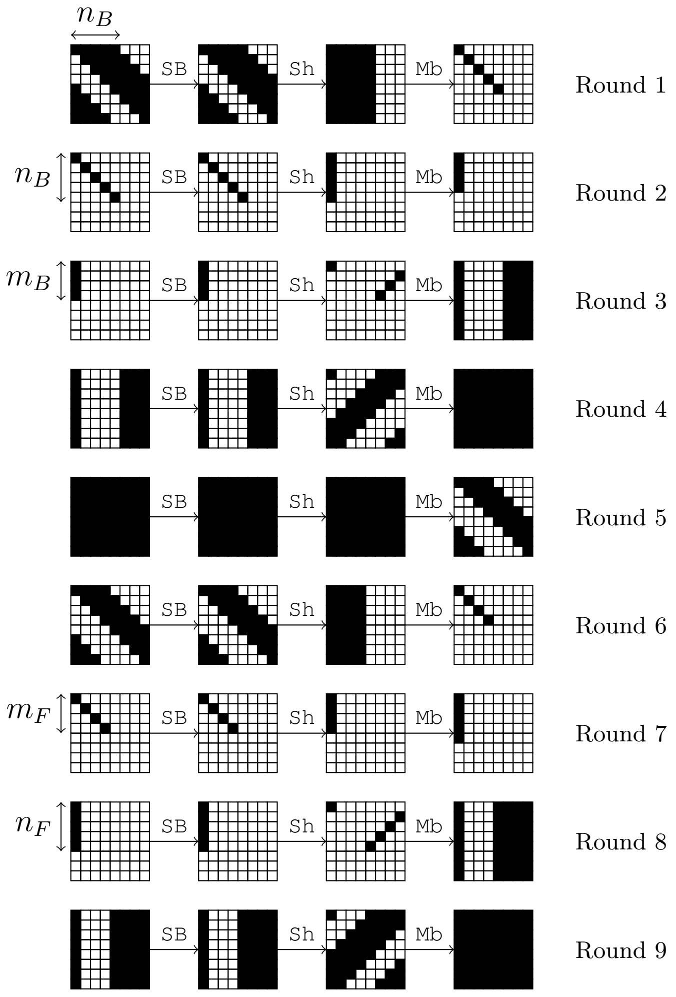


Figure 7: Non-fully-active truncated differential characteristic on 9 rounds of an AES-like permutation instantiated with $t = 8$.

4.2 Finding a conforming pair

As in the previous case, the algorithm that finds a pair of inputs conforming to this characteristic first produces many pairs for the middle rounds and then exhausts them outwards until one passes the probabilistic filter. The cost of those uncontrolled rounds is given by:

$$2^{c(t-n_B)}\,2^{c(t-n_F)} = 2^{c(2t-n_B-n_F)}, \tag{17}$$

since we need to pass one $n_B \leftarrow m_B$ transition in the backward direction and one $m_F \rightarrow n_F$ in the forward direction.

We now detail a way to find a solution for the middle rounds (Figure 8) when the input difference δ_{IN} after the first **SubBytes** layer in state S1 and the output difference δ_{OUT} after the last **MixCells** layer in state S12 are fixed in a way that the truncated characteristic holds in S0 and S12. The beginning of the attack is exactly the same as before in the sense that once the output differences have been fixed, we generate the $2t$ lists that contains the paired values of the t forward **SuperSBox** sets and the t backward **SuperSBox** sets. Again, the same $2t$ lists overlap and we show how to find the solution of the merging problem in $2^{ct\cdot\min(m_F,m_B,\lceil\frac{t}{2}\rceil)}$ operations and $m_B\cdot 2^{ct}$ memory. We recall that L_i is the ith forward **SuperSBox** list (orange) and L'_i is the ith backward one (blue), for $1 \le i \le t$.

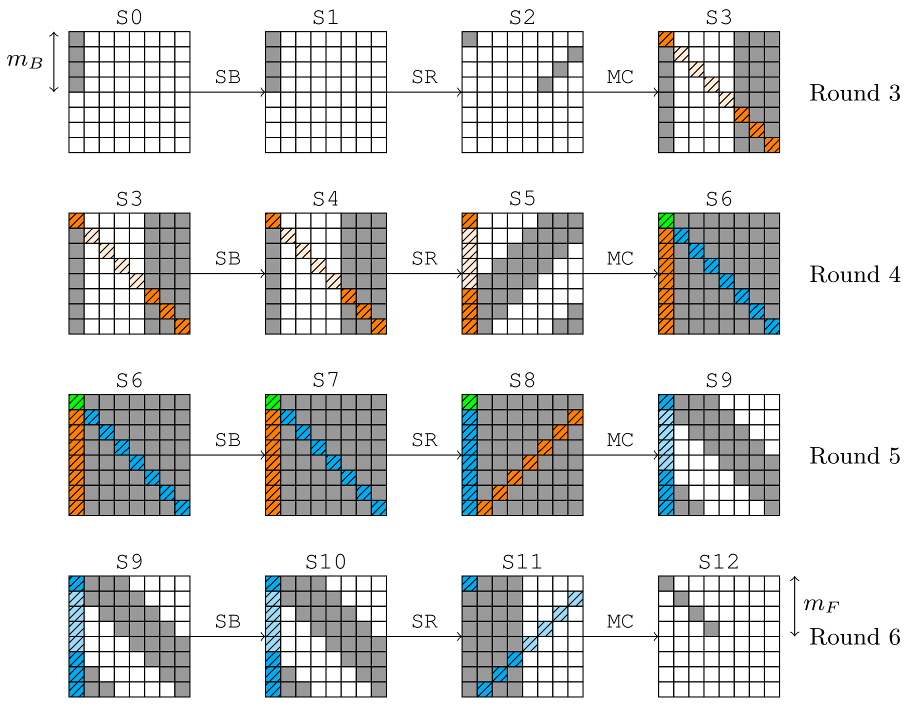


Figure 8: Inbound phase for the 9-round distinguisher attack on an AES-like permutation instantiated with $t = 8$ with a single fully-active state in the middle. A gray cell indicates an active cell; hatched and colored cells emphasize one **SuperSBox** set: there are seven similar others.

We proceed in three steps, where the first guesses the elements from some lists, this determines the remaining cells and we finish by checking probabilistic events. Without loss of generality, we assume in the sequel that $m_B \le m_F$; if this is not the case, then we start Step 1 by guessing elements of lists L_i in S8. We split the analysis into two cases, whether $m_B \le \lceil\frac{t}{2}\rceil$ or $m_B > \lceil\frac{t}{2}\rceil$.

First case: $m_B \leq \lceil \frac{t}{2} \rceil$ — In this case, we use the strong constraints on the vector spaces spanned by the m_B differences on each columns to find a solution to the merge problem.

Step 1. We start by guessing the elements of the m_B lists $L'_1, \dots, L'_{m_B}$ in state `S6`. There is a total of 2^{ctm_B} possible combinations.

Step 2. In particular, the previous step sets the differences of the m_B first diagonals of `S6` such that there are exactly m_B known differences on each of the t columns of the state. This allows to determine all the differences in `S5` since there are exactly m_B independent differences in each column of that state. Consequently, we linearly learn all the differences of `S6`.

Step 3. Since all differences are known in `S6`, we determine 1 element in each of the $t - m_B$ remaining L'_i lists: they are of size 2^{ct} and we count ct bits of constraints coming from t differences. From the known differences, we also get a suggestion of 2^{ct-cm_B} values for the cells of each column. Indeed, the elements of the t lists L_i in `S5` can be represented as disjointed sets regarding the values of the differences, since the differences can only take 2^{cm_B} values per column. Assuming that they are uniformly distributed[3], we get $2^{ct}/2^{cm_B} = 2^{ct-cm_B}$ elements per disjointed set for each list: they all share the same value of the differences, but have different values. Additionally, the ct-bit constraints of each list L_i allows to find one element in each, and therefore a solution to the merge problem, with probability $2^{((ct-cm_B)-ct)t} = 2^{-ctm_B}$.

Step 4. Finally, trying all the 2^{ctm_B} elements in $(L'_1, \dots, L'_{m_B})$, we expect to find $2^{ctm_B} \, 2^{-ctm_B} = 1$ solution that gives a pair of internal states conforming to the four middle rounds with a few operations.

Second case: $m_B > \lceil \frac{t}{2} \rceil$ — The columns of differences are less constrained, and it is enough to guess $\lceil \frac{t}{2} \rceil$ lists in the first step to find a solution to the merge problem.

Step 1. We start by guessing the elements of the $\lceil \frac{t}{2} \rceil$ lists $L'_1, \dots, L'_{m_B}$ in state `S6`. There is a total of $2^{ct\lceil \frac{t}{2} \rceil}$ possible combinations.

Step 2. The previous step allows to filter $2^{c(t-2\lceil \frac{t}{2} \rceil)}$ elements in each of the t lists L_i. Depending of the parity of t, we get 1 element per list for even t, and 2^{-c} for odd ones[4]. In the latter case, there are then a probability 2^{-ct} that the t elements are found in the t lists L_i.

Step 3. In the event that elements have been found in the previous step, we determine completely the remaining $2ct(t - \lceil \frac{t}{2} \rceil)$ values and differences of the remaining $t - \lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor$ lists L'_i. We find a match in those lists with probability $2^{-ct} \times 2^{(ct-2ct)(t-\lceil \frac{t}{2} \rceil)} = 2^{-ct(1+\lfloor \frac{t}{2} \rfloor)}$.

Step 4. Finally, trying all the $2^{ct\lceil \frac{t}{2} \rceil}$ elements in $(L'_1, \dots, L'_{\lceil \frac{t}{2} \rceil})$, we expect to find $2^{ct\lceil \frac{t}{2} \rceil} \, 2^{-ct(1+\lfloor \frac{t}{2} \rfloor)} = 1$ solution that gives a pair of internal states that conforms to the four middle rounds with a few operations.

Hence, in any case, 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 $2^{ct \min\left(m_B, \lceil \frac{t}{2} \rceil\right)}$ and memory $m_B 2^{ct}$. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging $2^{c(2t-n_B-n_F)}$ times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation conforming to the truncated differential

[3] This is a classical assumption, and here it is due to the non-linear SBox.

[4] Indeed, $t - 2\lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor - \lceil \frac{t}{2} \rceil$ equals 0 when t is even, and -1 when t is odd.

characteristic in time complexity $2^{ct\min(m_B,\lceil\frac{t}{2}\rceil)}\,2^{c(2t-n_B-n_F)}=2^{c(t(\min(m_B,\lceil\frac{t}{2}\rceil)+2)-n_B-n_F)}$ and memory complexity $m_B\cdot 2^{ct}$.

Finally, without assuming $m_B \leq m_F$, the time complexity T of the algorithm generalizes to:

$$\log_2(T) = c\left(t\cdot\min\left\{m_B, m_F, \left\lceil\frac{t}{2}\right\rceil\right\} + 2t - n_B - n_F\right), \tag{18}$$

with $n_F+m_F \geq t+1$ and $n_B+m_B\geq t+1$, and memory requirements of $m_B\cdot 2^{ct}$.

4.3 Comparison with ideal case

In the ideal case, the generic complexity $C(a,b)$ is given by the limited birthday distinguisher:

$$\log_{2^c}\Big(C(a,b)\Big) = \max\left\{\min\left\{\frac{t^2-a}{2},\frac{t^2-b}{2}\right\}, t^2-a-b\right\}, \tag{19}$$

since we get an input space of size $IN = 2^{c\cdot a}$ and output space of size $OUT = 2^{c\cdot b}$. Without loss of generality, assume that $a\leq b$: this only selects whether we attack the permutation or its inverse. In that case, we have:

$$\log_{2^c}\Big(C(a,b)\Big) = \begin{cases} C_1(a,b) := (t^2-b)/2, & \text{if: } t^2 < 2a+b,\\ C_2(a,b) := a, & \text{if: } t^2 = 2a+b,\\ C_3(a,b) := t^2-a-b, & \text{if: } t^2 > 2a+b.\end{cases} \tag{20}$$

In the case of the 9-round distinguisher, the generic complexity equals $C(t\cdot n_B, t\cdot n_F)$ since there are n_B active diagonals at the input, and n_F active diagonals at the output. Let us compare T and the case of $C_3(t\cdot n_B, t\cdot n_F)$ where $t > 2n_B+n_F$ corresponding to the limited birthday distinguisher. We want to find set of values for the parameters (t, n_F, n_B, m_F, m_B) such that our algorithm runs faster that the generic one, that is T is smaller than $C_3(t\cdot n_B, t\cdot n_F)$. In the event that $\min\left(m_F, m_B, \lceil\frac{t}{2}\rceil\right)$ is either m_F or m_B, we can show that T is always greater than $C_3\left(t\cdot n_B, t\cdot n_F\right)$, and so are the cases involving $C_2\left(t\cdot n_B, t\cdot n_F\right)$ and $C_1\left(t\cdot n_B, t\cdot n_F\right)$.

We consider the case $\min\left(m_F, m_B, \lceil\frac{t}{2}\rceil\right) = \lceil\frac{t}{2}\rceil$:

$$\log_{2^c}\Big(C_3\left(t\cdot n_B, t\cdot n_F\right)\Big) - \log_{2^c}\Big(T\Big) = t\left(t-n_F-n_B\right) - t\left\lceil\frac{t}{2}\right\rceil - 2t + n_B + n_F. \tag{21}$$

With t as a parameter and $n_F, n_B \in \{1,\ldots,t\}$, our algorithm turns out to be a distinguisher when the quantity from (21) is positive, which is true as soon as:

$$(n_B+n_F)(1-t) + t\left(t-2-\left\lceil\frac{t}{2}\right\rceil\right) \geq 0. \tag{22}$$

Since $t-\lceil\frac{t}{2}\rceil = \lfloor\frac{t}{2}\rfloor$, we can show that if $n_F\in\{1,\ldots,t\}$ and $n_B\in\{1,\ldots,t\}$ are chosen such that

$$2 \leq n_F+n_B \leq \frac{t}{t-1}\left(\left\lfloor\frac{t}{2}\right\rfloor - 2\right), \tag{23}$$

then our algorithm is more efficient than the generic one. Note that this may happen only when $t\geq 8$ and that m_F and m_B are still constrained by the MDS bound: $n_F+m_F\geq t+1$ and $n_B+m_B\geq t+1$.

We can also consider an 8-round case by considering the characteristic from Figure 7 where the last round is removed[5]: the generic complexity becomes $C(t\cdot n_B, n_F)$. Note

[5] We still assume that $n_B \leq n_F$. If not, then the generic complexity becomes $C(n_B, t\cdot n_F)$ by removing the first round.

that the complexity of our algorithm remains unchanged: there are still two probabilistic transitions to pass. For $t \geq 4$, we can show that there are many ways to set the parameters (n_F, n_B, m_F, m_B) so that $T \geq C(t \cdot n_B, n_F)$, and the best choice providing the most efficient distinguisher happens when the MDS bounds are tight, i.e.: $n_F + m_F = t + 1$ and $n_B + m_B = t + 1$.

For the sake of completeness, we can also derive distinguishers for 7-round of the permutation by considering the characteristic from Figure 7 where the first and last rounds are removed, as soon as $t \geq 4$. The generic complexity in that scenario is $C(n_B, n_F)$. Again, there are several ways to set the parameters, but the one that minimizes the runtime T of our algorithm also verifies the MDS bounds: $n_B = 1$, $m_B = t$, $m_F = 1$ and $n_F = t$.

We give examples of more different cases in Table 2, which for instance match `AES` and `Grøstl` instantiation. We note that the complexities of our algorithm may be worse that other published results.

Rounds	**Cipher**		**Parameters**				**Complexities**	
	t	**c**	$\mathbf{n}_B$	$\mathbf{m}_B$	$\mathbf{m}_F$	$\mathbf{n}_F$	$\log_2(\mathbf{T})$	$\log_2(\mathbf{C})$
9	8	8	1	8	8	1	368	$\log_2 C(t \cdot n_B, t \cdot n_F) = 384$
8	8	8	8	1	4	5	88	$\log_2 C(n_B, t \cdot n_F) = 128$
8	8	8	5	4	1	8	88	$\log_2 C(t \cdot n_B, n_F) = 128$
7	8	8	8	1	1	8	64	$\log_2 C(n_B, n_F) = 384$
8	7	8	7	1	4	4	80	$\log_2 C(n_B, t \cdot n_F) = 112$
8	7	8	4	4	1	7	80	$\log_2 C(t \cdot n_B, n_F) = 112$
7	7	8	7	1	1	7	56	$\log_2 C(n_B, n_F) = 280$
8	4	8	4	1	4	1	56	$\log_2 C(n_B, t \cdot n_F) = 64$
8	4	8	1	4	1	4	56	$\log_2 C(t \cdot n_B, n_F) = 64$
7	4	8	4	1	1	4	32	$\log_2 C(n_B, t \cdot n_F) = 64$

Table 2: Examples of reached time complexities for several numbers of rounds and different (t, c) scenarios.

5 Applications to `Grøstl-256` permutations

The permutations of the `Grøstl-256` hash function implement the previous generic algorithms will the following parameters: $t = 8$, $c = 8$ and $N_r = 10$.

Three fully-active states. From the analysis of Section 3, we can directly conclude that this leads to a distinguishing attack on the 9-round reduced version of the `Grøstl-256` permutation with $2^{c(t^2/2+2(t-1))} = 2^{368}$ computations and $2^{ct} = 2^{64}$ memory, when the ideal complexity requires $2^{ct(t-2)} = 2^{384}$ operations.

As detailed previously, we could derive distinguishers for 8-round `Grøstl-256` with $2^{c(t^2/2+t-1)} = 2^{312}$ operations and for 7-round `Grøstl-256` with $2^{ct^2/2} = 2^{256}$, but those results are more costly than previous known results.

Similarly, as explained in Section 2.3, this result also induces a nontrivial observation on the 9-round reduced version of the `Grøstl-256` compression function with identical complexity.

Non-fully-active characteristic. With the generic analysis of Section 4 that uses a single fully-active middle state, $t = 8$ only allows to instantiate the parameterized truncated differential characteristic with $n_F = n_B = 1$, which determines $m_F = m_B = 8$. Indeed, (23) imposes $2 \leq n_B + n_F \leq \frac{16}{7}$, which gives integer values $n_F = n_B = 1$. Note that it is exactly the case of the three fully-active states in the middle treated in Section 3, with the same complexities.

For 8-round distinguishers, the case $t = 8$ where $n_B \leq n_F$ may give the parameters $n_B = 5$, $m_B = 4$, $m_F = 1$ and $n_F = 8$ with the last round of the characteristic of Figure 7 is removed. If $n_B > n_F$, we instantiate the characteristic with the first round removed with the values $n_B = 8$, $m_B = 1$, $m_F = 4$ and $n_F = 5$. In both cases, the time complexity of the distinguishers are 2^{88} operations with 2^{64} of memory requirements, whereas the generic algorithm terminates in about 2^{128} operations. As for 7-round distinguishers, removing both first and last rounds of the characteristic of Figure 7 leads to an efficient distinguishers for Grøstl-256 when $n_B = 8$, $m_B = 1$, $m_F = 1$ and $n_F = 8$. The corresponding algorithm runs in 2^{64} operations with 2^{64} of memory requirements, when the corresponding generic algorithm needs 2^{384} operations to terminate. We note that those 8- and 7-round distinguishers are not as efficient as other available techniques: we provide them for the sake of completeness.

6 Conclusion

In this article, we have provided a new and improved cryptanalysis method for AES-like permutations, by using a rebound-like approach as well as an algorithm that allows us to control four rounds in the middle of a truncated differential path, with a lower complexity than a general probabilistic approach. To the best of our knowledge, all previously known methods only manage to control three rounds in the middle and we close the open problem whether this was possible or not.

We apply our algorithm on several algorithms and in particular on the building blocks of both the 256 and 512-bit versions of the SHA-3 finalist Grøstl. We could provide the best known distinguishers on 9 rounds of the internal permutations of Grøstl-256, while for Grøstl-512, we have considerably increased the number of analyzed rounds, from 7 to 10.

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.

Acknowledgements

We would like to thank the anonyous referees for their valuable comments on our paper. Jérémy Jean is partially supported by the French National Agency of Research through the SAPHIR2 project under Contract ANR-08-VERS-014 and by the French *Délégation Générale pour l'Armement* (DGA). Thomas Peyrin is supported by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06. This work was partially supported by the French National Agency of Research: ANR-11-INS-011.

References

1. Barreto, P.S.L.M., Rijmen, V.: Whirlpool. In van Tilborg, H.C.A., Jajodia, S., eds.: Encyclopedia of Cryptography and Security (2nd Ed.). Springer (2011) 1384–1385

2. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (updated) (2009)
3. 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
4. Daemen, J., Rijmen, V.: Rijndael for AES. In: AES Candidate Conference. (2000) 343–348
5. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submitted to the SHA-3 competition, NIST (2008)
6. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate (Updated version). Submitted to the SHA-3 competition (2011)
7. 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
8. 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
9. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In Preneel, B., Takagi, T., eds.: CHES. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 326–341
10. 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
11. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Rebound Attack on the Finalist Grøstl. In Canteaut, A., ed.: FSE. Volume 7549 of Lecture Notes in Computer Science., Springer (2012) 110–126
12. 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
13. 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
14. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. [15] 126–143
15. 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)
16. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full LANE Compression Function. [15] 106–125
17. 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 5665 of Lecture Notes in Computer Science., Springer (2009)
18. 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
19. 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
20. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010) (extended version).
21. 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
22. 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
23. Peyrin, T.: Cryptanalysis of Grindahl. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 551–567
24. 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
25. 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
26. 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
27. 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 Distinguish attack on 10-round `Grøstl-512`

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.

A.1 The truncated differential characteristic

To distinguish its permutation P_{512} [6] reduced to 10 rounds, we use the truncated differential characteristic with the sequence of active bytes

$$64 \xrightarrow{R_1} 8 \xrightarrow{R_2} 1 \xrightarrow{R_3} 8 \xrightarrow{R_4} 64 \xrightarrow{R_5} 128 \xrightarrow{R_6} 64 \xrightarrow{R_7} 8 \xrightarrow{R_8} 1 \xrightarrow{R_9} 8 \xrightarrow{R_{10}} 64. \tag{24}$$

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 represented on Figure 9. 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}.

A.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 10) is similar to the ones presented in [20, 21] 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 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[7] 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 11: 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.

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

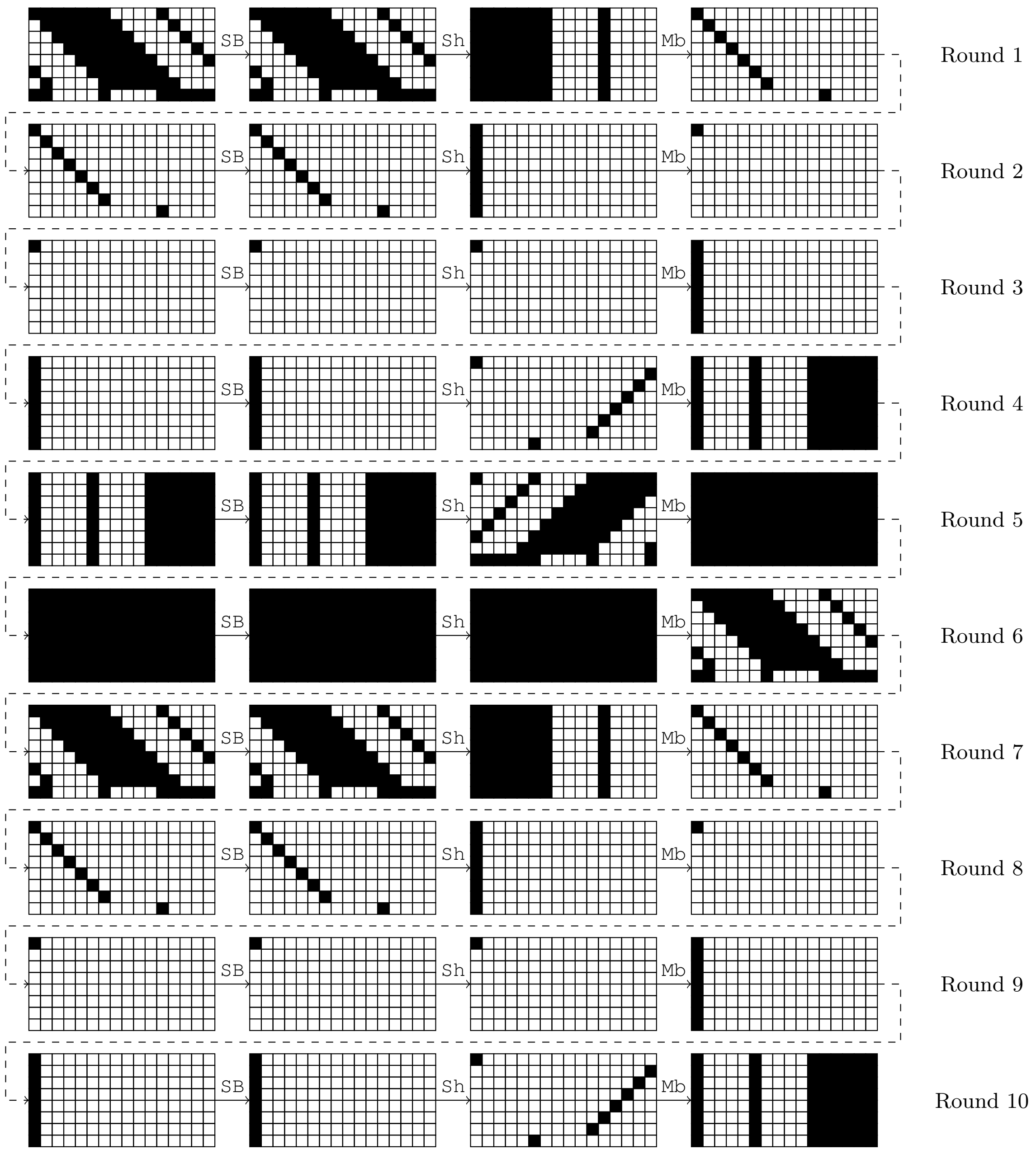


Figure 9: The 10-round truncated differential characteristic used to distinguish the permutation `P` of `Grøstl-512` from an ideal permutation.

Then, we note that the **MixCells** 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 an 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 11 reports the subspace dimensions for each L'_i.

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

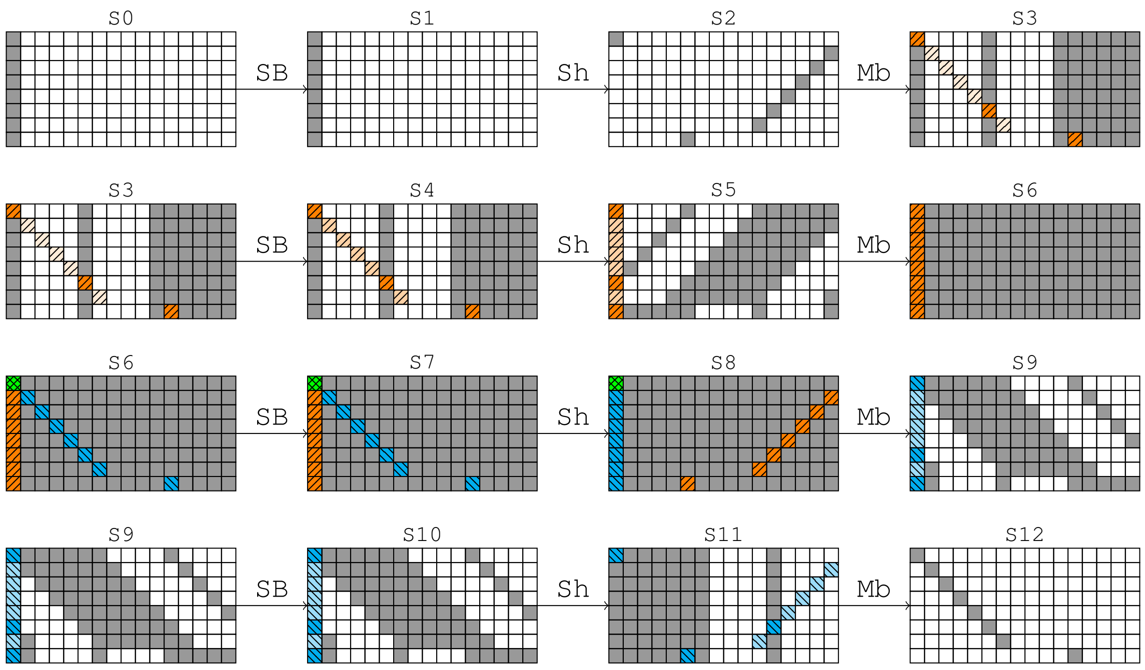
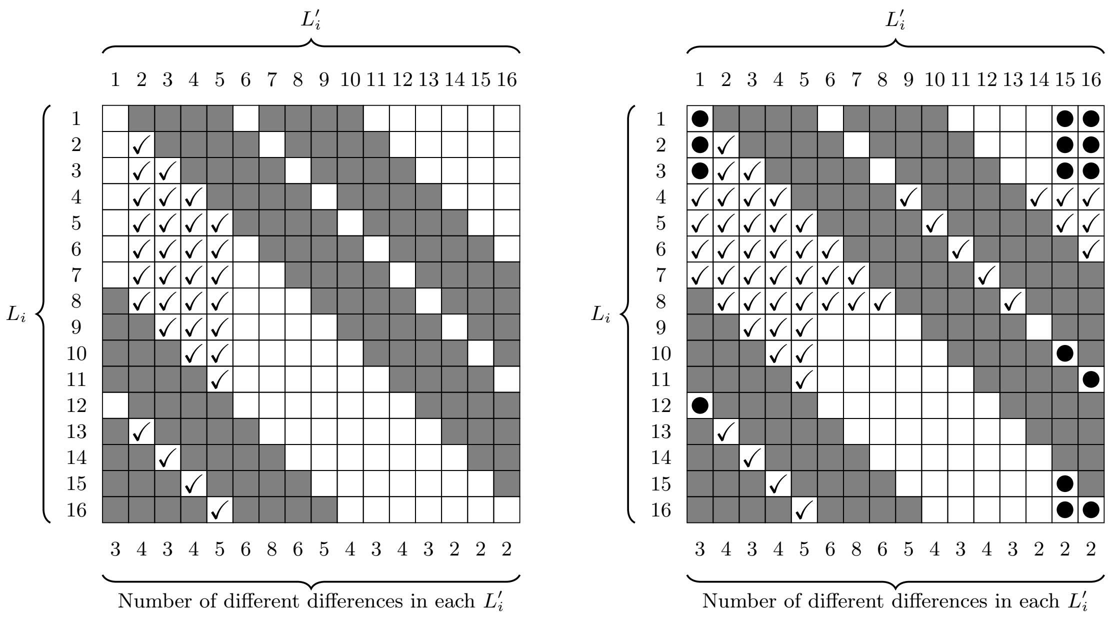


Figure 10: Inbound phase for the 10-round distinguisher attack on the `Grøstl-512` permutation P_{512}. The four rounds represented are the rounds 4 to 7 from the whole truncated differential characteristic (Figure 9). A gray byte indicates an active byte; hatched and coloured bytes emphasize the **SuperSBoxes**.

solution; that is, we want to find a single element in each of the 32 lists. In the sequel, we describe a sequence of 4 guess and determine steps illustrated by pictures before and after each *determine* phase.



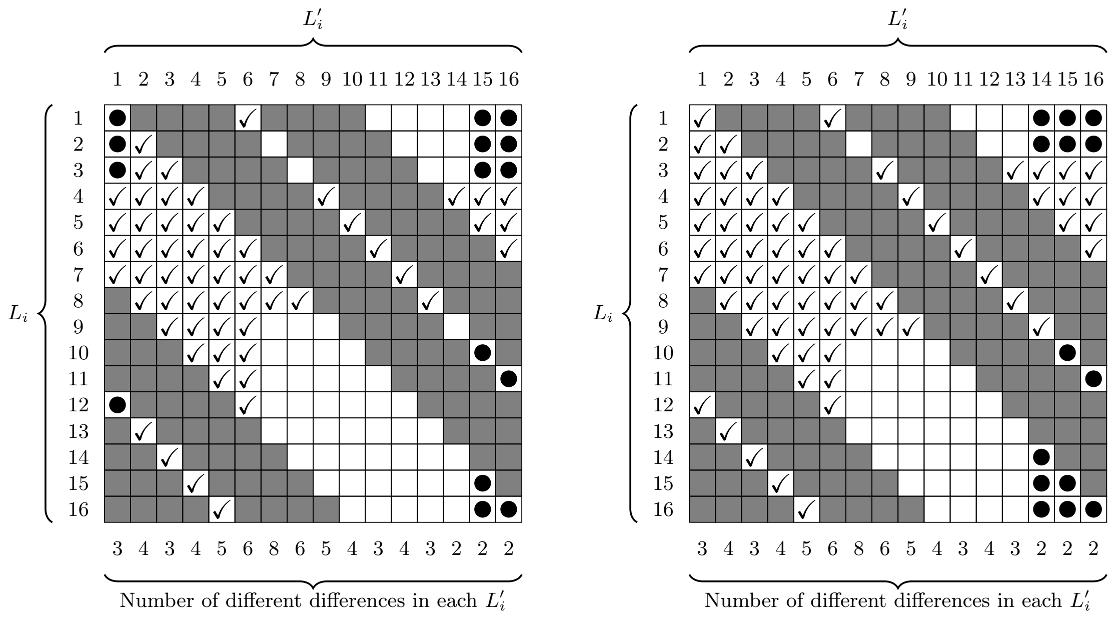
(a) Initial guesses.

(b) End of first determine step.

Figure 11: First guess on the algorithm. 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.

Step 1. We start by guessing the values and the 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 ✓ from Figure 11a 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 11b. 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. See Figure 11b for the current situation of the guess-and-determine algorithm.

Step 2. 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 (see Figure 12a). 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 $2^{256+16} = 2^{272}$.



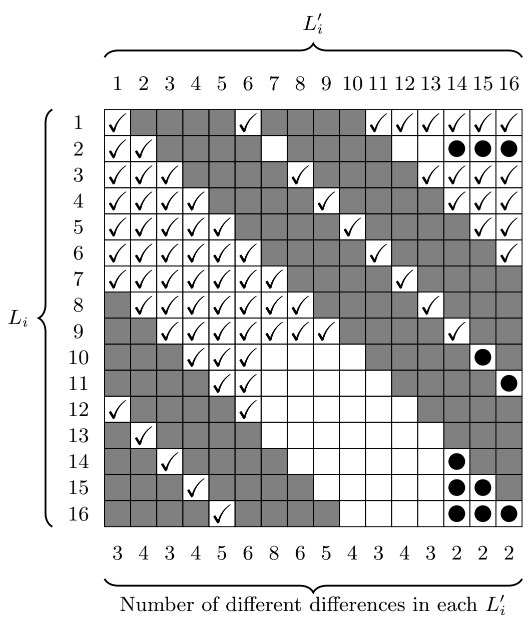
(a) Second guess: L'_6.

(b) End of the second guess (L'_6).

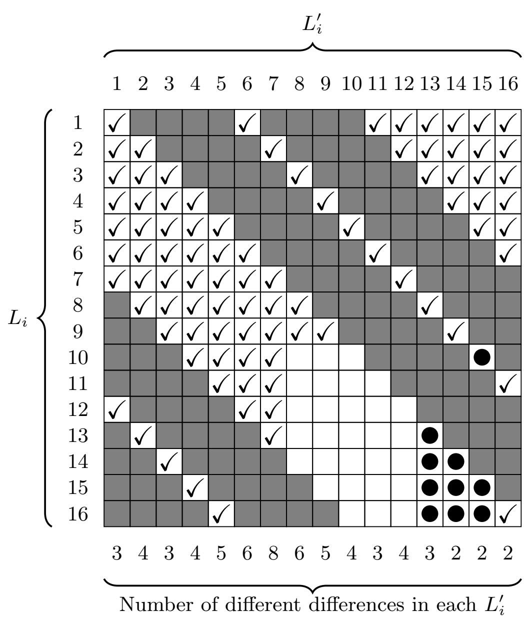
Figure 12: Second guess on the algorithm. A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

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 $2^{64-3\times 8} = 2^{40}$ values constrained on five bytes (five ✓). Hence, we are able to determine all the unset values in L'_1: see Figure 12b for the current situation.

Step 3. 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} (see Figure 13a).



(a) Third guess: L_1.



(b) End of the third guess (L_1).

Figure 13: Third guess on the algorithm. 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.

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 $2^{64-2\times 8} = 2^{48}$ values, which are now constrained on six bytes (six ✓).

Step 4. We describe in Figure 13b the knowledge propagated so far, with time complexity 2^{280} and probability 1. In this step, no new guess is needed, and we show how to end the algorithm by probabilistic filterings on the remaining unset lists.

First, 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 (four ✓). We assume the correct values are found, such that the element of L'_8 happens to be correctly defined with probability 2^{-16} (five ✓), L'_9 with probability 1 (four ✓) and L'_{15} also with probability 1 since we get 6 ✓ that complete the knowledge of the 2-dimensional subspace of differences (six ✓ and two •). We continue in L'_{11} by learning the full vector of differences (three independent ✓ for a subspace of dimension 3), which constraints L_{12} on 11 bytes (five ✓ and one ●) so that we get a valid element with probability 2^{-24}.

At this point, L_{16} is reduced to a single element with probability 2^{-8} (three ✓ and three •), which adds constraints on the three lists L'_{11}, L'_{13} and L'_{14}, where we already know all the differences (Figure 14). Consequently, we get respectively 5, 5 and 6 independent values (✓) on subspaces of respective dimensions 3, 3 and 2, which filter those three lists to a single element with probability 1. Finishing the guess and determine technique is done by filtering L'_{10} and L'_{12} with probability 1 (four ✓ in a subspace of dimension 4 for both lists), and then the three remaining lists L_{13}, L_{14} and L_{15} are all reduced to a single element which are the valid one with probability 2^{-64} for each (eight ✓). After this, if a solution is found, everything has been determined.

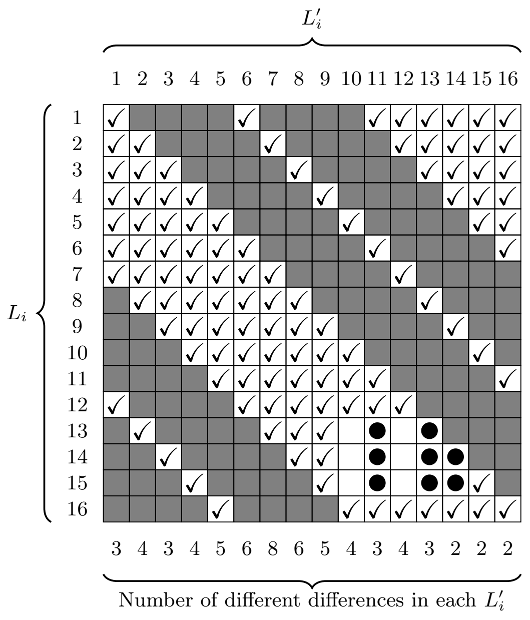


Figure 14: End of the guess-and-determine algorithm: after list L_{16} has been fully determined, we filter $L'_{10}, \ldots, L'_{14}$ with probability 1 and then $L_{13}, \ldots, L_{15}$ with probability 2^{-64}.

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

$$2^{-8-16-24-40-64-64-64} = 2^{-280}, \tag{25}$$

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.

A.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 2.3, 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 Figure 9, it is possible to remove either round 3 or 9 and spare one $8 \to 1$ 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 $8 \to 2$ truncated differential transition instead of $8 \to 1$, 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 $OUT = 2^{128}$.

B Distinguishers for reduced PHOTON permutations

Using the same cryptanalysis technique, it is possible to study the recent lightweight hash function family PHOTON [8], which is based on five different versions of AES-like permutations. Using the notations previously described in this article, the five versions (c, t) for PHOTON are $(4, 5)$, $(4, 6)$, $(4, 7)$, $(4, 8)$ and $(8, 6)$ for increasing versions. All versions are defined to apply $N_r = 12$ rounds of an AES-like process.

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 1 with the comparison to previously known results). Indeed, the size t 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.