Improved Differential Attacks for ECHO and Grøstl

Thomas Peyrin

Ingenico, France
thomas.peyrin@ingenico.com

Abstract. We present improved cryptanalysis of two second-round SHA-3 candidates: the AES-based hash functions ECHO and Grøstl. We explain methods for building better differential trails for ECHO by increasing the granularity of the truncated differential paths previously considered. In the case of Grøstl, we describe a new technique, the *internal differential attack*, which shows that when using parallel computations designers should also consider the differential security between the parallel branches. Then, we exploit the recently introduced start-from-the-middle or Super-Sbox attacks, that proved to be very efficient when attacking AES-like permutations, to achieve a very efficient utilization of the available freedom degrees. Finally, we obtain the best known attacks so far for both ECHO and Grøstl. In particular, we are able to mount a distinguishing attack for the full Grøstl-256 compression function.

Keywords: hash function, cryptanalysis, ECHO, Grøstl, AES, internal differential attack.

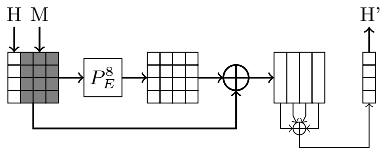
1 Introduction

Cryptographic hash functions are very important tools in cryptography, used in many applications such as digital signatures, authentication schemes or message integrity. Informally, a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. The classical security requirements for such a function are collision resistance and (second)-preimage resistance. Namely, it should be impossible for an adversary to find a collision (two distinct messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. Moreover, those primitives are traditionally used to simulate the behavior of a random oracle [2] and while the community is divided on such a requirement, in the ideal case an attacker should not be able to distinguish a hash function from a random oracle.

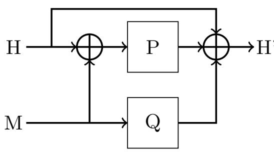
As many standardized hash functions [41, 31] have been broken a few years ago [45, 44], the NIST launched in 2008 the SHA-3 competition [33] that will lead to the future hash function standard. 14 candidates among 51 have been selected for the second round and many of them (like ECHO [3], Grøstl [14] or SHAvite-3 [5]) are actually using some parts of the standardized block cipher

`AES` [32, 10] as internal primitives or mimicking the structure of this cipher. While `AES`-256 can no more be considered as secure in the related-key model [7], the cryptography community has made important progresses concerning the evaluation of `AES`-based hash functions security [35, 19, 27, 25, 23, 26, 15]. Those attacks make an extensive use of the freedom degrees that are available in a hash function and even provides the best known distinguishing attack against `AES`-128 [15] in the known-key model [21, 30]. Much recent analysis of `AES`-based hash functions has helped to identify the limits of current techniques, but as we show in this paper, it is possible to improve the differential path building methods used so far.

Our contributions. In this paper, we improve the best known cryptanalysis results [1, 18, 27, 26, 15] on two second round `SHA-3` candidates: the hash functions `ECHO` [3] and `Grøstl` [14]. While we do not provide advances regarding the freedom degrees optimization, we use the recently introduced Super-Sbox techniques [15, 28] in order to find pairs of inputs verifying a given differential path. We then exploit some specific properties of `ECHO` and `Grøstl` to derive very good differential paths. More precisely, we improve the previously known truncated differential paths for `ECHO` by reducing the size of the truncated words considered. This allows us to broaden the differential trail search space, therefore increasing our probability to find a good path, but also augmenting the search complexity. We circumvent this constraint by giving a heuristic method to prune the potential candidates. Concerning `Grøstl`, we describe a novel yet simple cryptanalysis technique: the *internal differential attack*. It may be applied for functions using parallel branches that are not sufficiently made distinct. In that case, the attacker can find input instances (where a classical differential attack exhibits pairs of inputs) verifying non random properties on the output.



`ECHO`



`Grøstl`

As a result, we improve the complexity for distinguishing the internal permutation of `ECHO` from a random 2048-bit permutation for a number of rounds corresponding to the full 256-bit version. Because of the folding phase after the permutation application at the end of the `ECHO` compression function, this attack does not translate into a distinguishing attack for the full `ECHO` compression function, nor the hash function itself. We provide also the first distinguishing attack on the full internal permutations for the 256-bit version of `Grøstl`, which can be directly derived into a distinguisher on the full `Grøstl`-256 compression function. Structural distinguishers (independent of the number of rounds) were already described in the original submission document [14]. For example, it was already identified that one can find fixed points or build a distinguisher for the

Table 1. Summary of results for ECHO, ECHO-SP and Grøstl compression functions. ECHO-256, ECHO-SP-256, ECHO-512 and ECHO-SP-512 compression functions have 8, 8, 10 and 10 rounds respectively, while Grøstl-256 and Grøstl-512 compression functions have 10 and 14 rounds respectively. All details of these attacks are given in the extended version of this article [36].

target	rounds	computational complexity	memory requirements	type	section
ECHO-256 comp. function	3	2^{64}	2^{64}	semi-free-start collision[1]	this paper
	4	2^{64}	2^{64}	distinguisher	this paper
ECHO-512 comp. function	3	2^{96}	2^{64}	semi-free-start collision[1]	this paper
	6	2^{96}	2^{64}	distinguisher	this paper
ECHO-SP-256 comp. function	3	2^{64}	2^{64}	semi-free-start collision	this paper
	3	2^{64}	2^{64}	distinguisher	this paper
ECHO-SP-512 comp. function	3	2^{64}	2^{64}	semi-free-start collision[2]	this paper
	4	2^{64}	2^{64}	distinguisher	this paper
Grøstl-256 comp. function	7	2^{56}		distinguisher	see [26]
	8	2^{112}	2^{64}	distinguisher	see [15]
	9	2^{80}	2^{64}	distinguisher	this paper
	10	2^{192}	2^{64}	distinguisher	this paper
Grøstl-512 comp. function	11	2^{640}	2^{64}	distinguisher	this paper

compression function with the generalized birthday paradox [43]. However, our results also allow to distinguish the Grøstl compression function from the same construction when assuming the two internal permutations P and Q as ideal. This is not the case for the known structural distinguishers since they already consider the two internal permutations as ideal. Our results are also interesting because they exploit the specificities of P and Q which is essential in order to really evaluate the security margin of this hash function in terms of number of rounds. All the results and the corresponding computation/memory complexities for ECHO, ECHO-SP (the simple-pipe version of ECHO) and Grøstl are summarized in Table 1 and available in the extended version of this article [36]. Note that none of the results described in this article seem to endanger the security of the ECHO compression function or the Grøstl hash function.

2 Previous Cryptanalysis

In this section, we recall the recent advances regarding cryptanalysis of AES-like permutations and their specificities. In the rest of the paper, we will use the start-from-the-middle and Super-Sbox attacks as basic tool for finding input pairs verifying a given differential path.

[1] A semi-free-start collision can be found for the 4-round reduced ECHO-256 or ECHO-512 compression functions with complexity 2^{224} computations and 2^{64} memory, if the salt value can be controlled by the attacker.

[2] A semi-free-start collision can be found for the 4-round reduced ECHO-SP-512 compression function with complexity 2^{224} computations and 2^{64} memory, if the salt value can be controlled by the attacker.

2.1 Building Differential Trails with Truncated Differences

Cryptanalysis of AES-based hash functions began with the hash family proposal Grindahl [20] for which collision attacks have been found [35, 19]. This showed that truncated differentials [22, 20] are very useful when cryptanalyzing a byte-oriented primitive such as the AES. Namely, instead of looking at the actual difference value of a byte, one only checks if a byte contains a difference (active byte) or not (inactive byte). In particular, this allows the attacker to handle the non-linear Sboxes quite nicely when building differential trails. On the other hand, the differential transitions through the linear MixColumns layer will now be verified probabilistically.

The matrix multiplication underlying the MixColumns transformation on a r-byte column for AES or Grøstl presents the interesting property of being a Maximum-Distance Separable (MDS) mapping: the number of active input and output bytes is always greater or equal to $r+1$ (unless there is no active input and output byte at all). When picking random inputs, the probability of success of a differential transition that meets the MDS constraints through a MixColumns layer is determined by the number of active bytes in the output. More precisely, if such a differential transition contains k active bytes in one column of the output, its probability of success will approximatively be equal to $2^{-8\times(r-k)}$. For example, a $4 \mapsto 1$ transition for one column of the AES MixColumns layer has success probability of approximatively 2^{-24}. Note that the same reasoning applies when dealing with the invert function of the MixColumns layer as well.

2.2 Rebound Attacks

The rebound attack [27] is a new technique for using efficiently the available freedom degrees. The authors utilize truncated differential paths in which most of the cost lies in the middle rounds. Then, by using a local meet-in-the-middle-like technique, the freedom degrees are consumed in the middle part of the differential path, right where they can improve at best the overall complexity. More precisely, some rounds in the middle (the *controlled rounds*) will be verified with only a few operations on average, while the rest of the path both in forward and backward direction (the *uncontrolled rounds*) is fulfilled probabilistically. This cryptanalysis provides good results [25, 23] and can work without any special constraint on the differential path. However, the controlled part is limited to two rounds.

2.3 Start-from-the-Middle Attacks

In [26], the start-from-the-middle attack for AES-like permutations is introduced. It can be seen as a generalization of the previous technique in the sense that the idea is simply to use the freedom degrees for AES-like permutations in the "most expensive" part of the differential trail, without setting any constraint in the way this is handled. The "cheaper" parts are then covered in an inside-out manner in both forward and backward directions. The authors describe a freedom degrees use example that can control 3 rounds in the middle part, without increasing the complexity (i.e. with only a few operations). However, the depicted technique only works for specific differential paths, in which the number of active bytes in the controlled rounds is not too important. We refer to the original publication [26] for more details.

2.4 The Super-Sbox Cryptanalysis Technique

Finally, another example of start-from-the-middle attacks is the Super-Sbox cryptanalysis ([15] and independently published in [28]). The idea is that one can view two rounds of an AES-like permutation as the parallel application of a layer of big Sboxes, named Super-Sboxes, preceded and followed by simple affine transformations. This technique can control 3-rounds in the middle of the differential trail with only a few operations on average, but works especially when the number of active bytes in the controlled rounds is important (this allowed to use longer differential paths which generally contain more active bytes). Because of some local precomputation steps, the drawback of this technique is its memory requirement when the size of the internal state of the scheme is too big. In the case of Grøstl this remains acceptable with a 2^{64} memory requirement, but in the case of ECHO as much as 2^{512} memory is required, making this tool unsuitable for this hash proposal. We refer to the original article [15] for more details.

3 Improved Differential Attack for ECHO

3.1 Description of ECHO

ECHO is a double-pipe hash function using HAIFA [4] as chaining iteration mode. The message to hash is first padded and divided into fixed-length blocks M_i which are used to update iteratively the chaining variable H_i (originally initialized with an initial vector $H_0 = IV$) thanks to the compression function h: $H_i = h(H_{i-1}, M_i)$. Finally, the hash output is obtained by truncating the last chaining variable. The compression function is built upon a 2048-bit AES-like permutation P_E^R composed of R rounds and its internal state can be viewed as a 4×4 matrix of 128-bit words (or cells). A cell will be denoted by $C_{i,j}$, where i is its row position and j its column position in the matrix, starting the counting from 0. One round of P_E^R is composed of three layers: the "BIG SubBytes" layer (big Sbox or B.SB), the "BIG ShiftRows" layer (B.ShR) and the "BIG MixColumns" layer (B.MC).

The BIG SubBytes layer is a non-linear function defined by the application of a big Sbox S on each 128-bit cell and this big Sbox is made of 2 AES rounds. The classical AddRoundKey part from the AES is not present in P_E^R and in order to avoid trivial symmetric vulnerabilities that would occur, each big Sbox in ECHO is distinct thanks to different subkey additions in each of the 2-round AES uses. The first round subkey depends on the value of a 64-bit internal counter K that is different at each use, while the second round subkey is set to the 128-bit salt value and thus always remains the same during the whole ECHO computation. So, for each cell $C_{i,j}$ of the internal state, we compute

$$C'_{i,j} = S[C_{i,j}] = AES_{salt}(AES_{0||K}(C_{i,j})).$$

where AES_{sk} denotes the application of one AES round with the subkey sk. As for the AES, the BIG ShiftRows transformation permutes the position of each cell in its own row: for each cell $C_{i,j}$ of the internal state, we compute $C'_{i,j} = C_{i,Sub_i(j)}$ where $Sub_i(j) = (j - i) \bmod 4$. Finally, the BIG MixColumns function is a linear function that mixes all the columns of the internal state separately. More precisely, the 32-bit AES MixColumns function is reused: if $C^b_{i,j}$ denotes the b-th byte of the cell $C_{i,j}$, then we compute

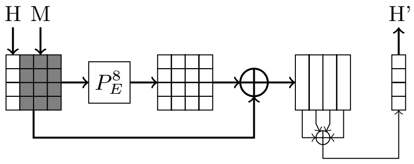
$$({C'}^b_{0,j}, {C'}^b_{1,j}, {C'}^b_{2,j}, {C'}^b_{3,j}) = AESMixColumns(C^b_{0,j}, C^b_{1,j}, C^b_{2,j}, C^b_{3,j})$$

for all $0 \leq j \leq 3$ and $1 \leq b \leq 16$. The round function on an internal state C can thus be defined as:

$$MixColumns \circ ShiftRows \circ SubBytes(C).$$

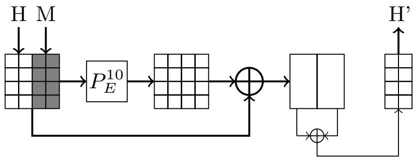
In the case of the ECHO-256 compression function, 8 rounds of the permutation are applied and a folding phase is processed after the final feedforward. Namely, the folding phase (denoted fold_{256}) xors all the four 512-bit columns together. Finally, the compression function takes a 1536-bit message input M (12 words) and a 512-bit chaining variable input H (4 words) and outputs a new 512-bit chaining variable H' with

$$H' = \mathsf{fold}_{256}(P_E^8(H||M) \oplus H||M)$$



In the case of the ECHO-512 compression function, 10 rounds of the permutation are applied in order to turn a 1024-bit message input M (8 words) and a 1024-bit chaining variable input H (8 words) onto a new 1024-bit chaining variable H'. A different folding phase is processed after the final feedforward. Namely, the folding phase (denoted fold_{512}) xors the two first and the two last 512-bit columns together.

$$H' = \mathsf{fold}_{512}(P_E^{10}(H||M) \oplus H||M)$$



Since ECHO is a nested design of AES-like permutations, we will always use the prefix "BIG" when referring to one of the three layers of the 2048-bit permutation. When not using a prefix, we will refer to the layers of the 2-round AES permutation in the big Sboxes of ECHO.

In the following, $\mathsf{B.SB}_R^{in}$ (respectively $\mathsf{B.SB}_R^{out}$) will denote the whole internal state just before (respectively just after) application of the BIG SubBytes layer during round R (starting the counting from 0). Similarly, $\mathsf{B.MC}_R^{in}$ and $\mathsf{B.MC}_R^{out}$ will stand for the input and output internal states of the BIG MixColumns layer during round R. Of course, we have $\mathsf{B.SB}_R^{in} = \mathsf{B.MC}_{R-1}^{out}$. We refer to [3] for the full specifications.

3.2 Generic Differential Paths

In order to fully use the power of recent freedom degrees optimization techniques, the core of the differential path we use will not differ from the ones described in [27, 26, 15]. The reason here is that this core characteristic is perfectly fit for using the available freedom degrees in the middle: it is computationally very costly in its middle part, but quite cheap on its side parts. This core truncated differential path is 7 rounds long and is depicted in Figure 1. Of course, when attacking a smaller number of rounds than 7, one can use this core to build a further reduced path by cutting off some of the first and/or last rounds.

The second advantage of this core characteristic is that the relatively low number of active cells in the controlled rounds makes it usable with the start-from-the-middle technique, as it is described in [26]: one can find a pair of internal states verifying the 128-bit truncated differential trail from the beginning of round 2 ($\mathsf{B.SB}_2^{in}$) up to the end of round 4 ($\mathsf{B.MC}_4^{out}$) with only one operation on average (and 2^{64} memory). Note that another view of the attack is to say that with one operation the attacker can find a pair of internal states such

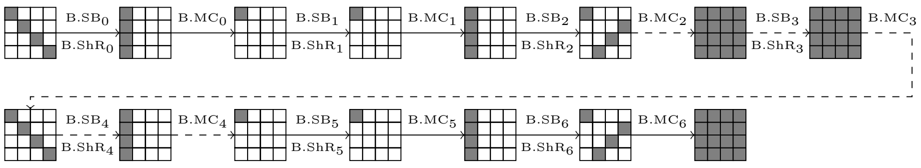


Fig. 1. Core of the truncated differential paths for 7-round reduced ECHO internal permutation. Each cell represents a 128-bit word and a gray cell stands for an active 128-bit word. The controlled rounds are depicted with dashed lines.

that the difference on $\mathsf{B.SB}_2^{out}$ and on $\mathsf{B.MC}_4^{out}$ are chosen (no more truncated differentials). Therefore, for `ECHO` we consider that the controlled rounds go from $\mathsf{B.SB}_2^{out}$ up to $\mathsf{B.MC}_4^{out}$.

One can easily check that the rest of the path (the uncontrolled rounds) is fulfilled with probability one, except round 1. Indeed, in round 1, a $4 \Rightarrow 1$ truncated differential transition is expected through the backward computation of the BIG MixColumns layer $\mathsf{B.MC}_1$. When dealing with 128-bit truncated differentials, this will happen with approximate probability $2^{-24\times16} = 2^{-384}$ (i.e. a $4 \Rightarrow 1$ byte-wise truncated differential transition is expected through sixteen parallel *AESMixColumns* functions) and this probability sets the overall 2^{384} complexity for finding a valid pair for the core path from Figure 1. We will see in the next section that by looking at byte-wise truncated differentials (instead of word-wise), one can sharpen the differential path and improve the success probability of this BIG MixColumns layer. On the other side, in order to be able to use the byte-wise truncated differentials at this stage and since he can control the difference only in $\mathsf{B.SB}_2^{out}$ (and not in $\mathsf{B.SB}_2^{in}$), the attacker will have to handle the backward computation of the BIG SubBytes layer of round 2 ($\mathsf{B.SB}_2$) as well. He then hopes that controlling both $\mathsf{B.SB}_2$ and $\mathsf{B.MC}_1$ with byte-wise truncated differentials will cost less than 2^{384} operations. Not controlling $\mathsf{B.SB}_2$ would lead us back to the 128-bit truncated differential cryptanalysis, as each active 128-bit word of $\mathsf{B.SB}_2^{in}$ will very likely contain 16 active bytes (i.e. fully active word) since full diffusion is achieved with only two `AES` rounds.

3.3 Differential Transitions for 2 `AES` Rounds

Now that we introduced the core of the differential path, we need to study the word-wise differential transitions. That is, instead of looking for 128-bit truncated differentials, we will look at byte-wise truncated differentials. Of course, we still fully leverage the previous works on start-from-the-middle attacks [26]: the attacker can find a valid candidate pair verifying the controlled rounds and fully control the differences in $\mathsf{B.SB}_2^{out}$ and $\mathsf{B.MC}_4^{out}$ with one operation on average. Sharpening the differential path will improve the results since our scope is now wider, but it will also greatly increase the number of potential trails and complicate the analysis. For that reason, we need to heuristically filter them so that we place our search into a good subspace. First, we restrict ourselves to four types of byte-wise truncated differential words `F`, `C`, `D` and `1`, all depicted in Figure 2. Due to symmetry and diffusion considerations, we believe that analyzing other differentials would not provide better results, while it would greatly increase the search space. Secondly, we add the constraint that all the active 128-bit words in an internal state will present the same byte-wise truncated differential (all words have the same truncated differential types `F`, `C`, `D` or `1`). This seems a sound constraint as the processing of the BIG MixColumns layer on one word column of the internal state can be seen as the parallel application of sixteen *AESMixColumns* functions (one for each byte position). Thus, for each word column, instead of analyzing the behavior of sixteen parallel *AESMixColumns*

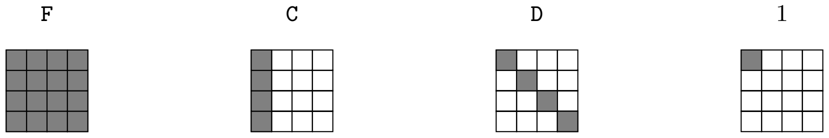


Fig. 2. Byte-wise truncated differential states for one word of ECHO. Each cell represents a byte and a gray cell stands for an active byte.

functions one conceptually only handles a single function that will do for all the 16. Those two filters will really simplify the analysis.

Since the attacker will have to control the behavior of BIG SubBytes layer $\mathsf{B.SB}_2$, we have to study the success probability for each possible transition for 2 AES rounds between the four bit-wise truncated differentials F, C, D and 1, especially in backward direction. First, we can compute the approximate probability of success for a one-round transition between those four 128-bit differential states and this is given in Table 2 for both forward and backward directions. Those probabilities are simply obtained by studying the *AESMixColumns* transitions for one AES round (since we are dealing with byte-wise truncated differentials, all the probabilities comes only from the *AESMixColumns* transitions, see [35]).

When computing backward through $\mathsf{B.SB}_2$, the *AESMixColumns* function from the second AES round is the first function to invert. But since this layer is fully linear, one can verify the expected backward transitions by carefully choosing the differences in $\mathsf{B.SB}_2^{out}$ beforehand. Since the start-from-the-middle attack allows us such a liberty, the second AES round in $\mathsf{B.SB}_2$ comes for free (one only has to check that the transition is not impossible, i.e. the probability in Table 2 is not null). Finally, having set all the constraints and the cost evaluation, we only have to pick the best backward differential transition through $\mathsf{B.SB}_2$ in terms of probability and active byte weight: $\mathtt{D} \Leftarrow \mathtt{1} \Leftarrow \mathtt{C}$. The transition $\mathtt{D} \Leftarrow \mathtt{1}$ is free as showed by Table 2, while the 2^{-24} probability for the transition $\mathtt{1} \Leftarrow \mathtt{C}$ is not taken in account since we can avoid it by carefully choosing the byte-wise truncated differences in $\mathsf{B.SB}_2^{out}$ beforehand. Therefore, controlling $\mathsf{B.SB}_2$ is now completely free for the attacker.

Now that we controlled the differential behavior of $\mathsf{B.SB}_2$, what is the improvement obtained for the BIG MixColumns layer $\mathsf{B.MC}_1$? Since we only have four active bytes in D, we can focus on controlling 4 parallel *AESMixColumns* transitions instead of 16. We are looking for $4 \Rightarrow 1$ transitions, each happening with probability 2^{-24}. Thus, for the whole BIG MixColumns layer, we get a probability of $2^{-24\times 4} = 2^{-96}$ and this has to be compared to the previous $2^{-24\times 16} = 2^{-384}$ probability.

Overall the whole 7-round differential path is depicted in Figure 3 and a valid candidate can be found with complexity 2^{96} operations and 2^{64} memory. Since the internal permutation of ECHO is much bigger than its hash output size, it should be easy to distinguish it from a random 2048-bit permutation. Note that our solution pair has four active 128-bit words in the input and four active 128-bit words in the output (the last BIG MixColumns call is not taken in account

Table 2. Byte-wise truncated differential transition approximated probabilities for one round of `AES`. The left table shows forward transitions, while the right one gives backward transitions.

Forward				
in \ out	`F`	`C`	`D`	1
`F`	1	0	2^{-96}	0
`C`	1	0	0	0
`D`	0	1	0	2^{-24}
1	0	1	0	0

Backward				
in \ out	`F`	`C`	`D`	1
`F`	1	2^{-96}	0	0
`C`	0	0	1	2^{-24}
`D`	1	0	0	0
1	0	0	1	0

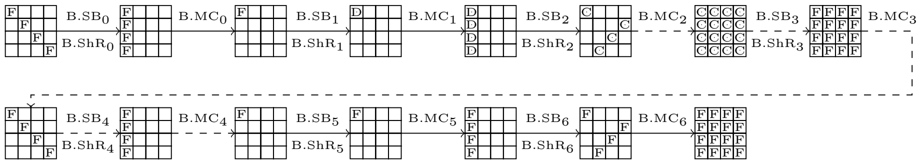


Fig. 3. 7-round differential path for the `ECHO` internal permutation. The controlled rounds are depicted with dashed lines.

since it is fully linear). A naive analysis would conclude that for a random 2048-bit permutation, finding such a pair with a birthday paradox technique should require at least $2^{(2048-512)/2} = 2^{768}$ operations. However, since the input and output amount of differences is low, the attacker can not fully leverage the power of the birthday paradox. We conclude by reusing the concept of *limited birthday distinguishers* [15] that for a random 2048-bit permutation, finding such a pair should require at least 2^{1024} operations.[1] Finally, 7 rounds of the internal permutation of `ECHO` can be distinguished from a random 2048-bit permutation with 2^{96} operations and 2^{64} memory. The amount of freedom degrees available during the attack is discussed in the Appendix A and a costly distinguisher for 8 rounds of the `ECHO` internal permutation is given in the extended version of this article [36].

4 Internal Differential Attack: Application to `Grøstl`

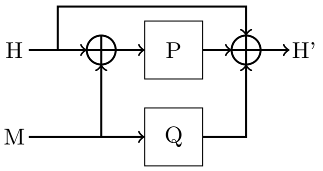
4.1 Description of `Grøstl`

We give in this section the description of `Grøstl` and refer to the submission document [14] for more details. `Grøstl` is a double-pipe hash function that uses a chaining mode similar to the Merkle-Damgård [29, 11] iteration. More

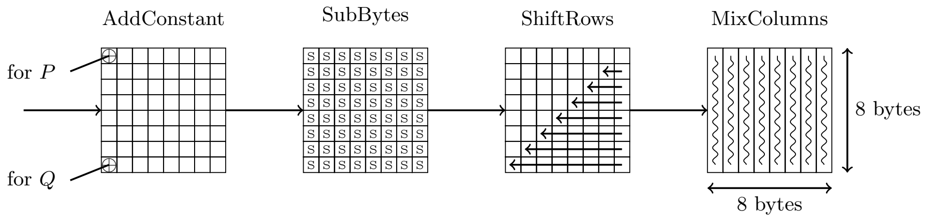
[1] The generic attack complexity for mapping through a permutation a fixed difference on i bits on the input and j bits on the output with $i \geq j$ is given by the formula $\max\{2^{j/2}, 2^{i+j-t}\}$, where t is the size of the permutation.

precisely, after having initialized the internal state H_0 and padded the input message string, the iteration i updates the $2n$-bit chaining variable H_i with the $2n$-bit incoming message block M_i by applying the compression function h: $H_i = h(H_{i-1}, M_i)$. After having processed all the t message blocks, an output function is applied to the last chaining variable to obtain the n-bit hash result: $hash = trunc_n(P(H_t) \oplus H_t)$, where $trunc_n$ is the truncation function of the n first bits and P is an `AES`-based permutation. The double-pipe compression function h is built upon two similar parallel `AES`-based permutations P and Q (that only differ by the constants addition layers) to update chaining variable H with message block M:

$$H' = P(H \oplus M) \oplus Q(M) \oplus H$$



In the case of `Grøstl`-256, the 512-bit internal state of both permutations can be viewed as a 8×8 matrix of bytes. A byte for permutation P is denoted by $CP_{i,j}$ (resp. $CQ_{i,j}$ for permutation Q), where i is its row position and j its column position in the matrix, starting the counting from 0. P and Q are both 10-round long and each round is composed of 4 layers. The first layer (AddConstant or AC) is a constant addition function. More precisely, for the round number i (starting the counting from 0), in P the byte $CP_{0,0}$ is xored with i and in Q the byte $CQ_{7,0}$ is xored with $i \oplus$ `0xff`. **Note that this layer is the only difference between permutations P and Q.** The second layer (SubBytes or SB) is a non-linear function defined by the application of the `AES` Sbox S to each byte. The third layer (ShiftRows or ShR) cyclically rotates to the left the position of each byte in its own row with the following constants: $(0, 1, 2, 3, 4, 5, 6, 7)$. Finally, the last layer (MixColumns or MC) is a linear function that mixes all the columns of the internal state separately. As for *AESMixColumns*, the matrix multiplication underlying this transformation is a Maximum-Distance Separable mapping. In order to avoid overweighting the notations, we used the same notations for the `ECHO` and `Grøstl` subfunctions, but their meaning is implicit depending on which scheme we are dealing with. The round function on an internal state C can thus be defined as $MixColumns \circ ShiftRows \circ SubBytes \circ AddConstant(C)$:



4.2 The Internal Differential Attack

In this section, we will show that very good differential trails can be found for `Grøstl`. Our new technique, ***the internal differential attack***, may apply when a function is built upon parallel computation branches that are not distinct enough. The trick is **to devise a differential path representing the differences between the branches and not between two inputs of the function**. Usually this is avoided by a forcing strong separation between the two parallel branches. For example, for all steps of the hash function `RIPEMD` [39, 12], very distinct constants are used in the left and right branches. However, in the case of `Grøstl`, this separation is thin between permutations P and Q, and we will describe in the next sections how to exploit this property in order to mount for example a distinguishing attack against the full `Grøstl`-256 compression function.

All the previous analysis of `Grøstl` studied the differential behavior of the permutations in a classic way. That is, they derived differential trails by dealing with two different inputs for each of the permutations P and Q (the two permutations were attacked separately). Those permutations mimicking the `AES` block cipher, the best usable differential paths naturally reached 8 rounds [15], but we argue that much more interesting trails can be built. We do not analyze the two permutations separately, but we build a differential path **between them**: we keep track of the differences ongoing between branch P and branch Q (see Figure 4). We compute two internal states A and B, such that $A \oplus B = \Delta_{IN}$ and such that $P(A) \oplus Q(B) = \Delta_{OUT}$.

This idea comes naturally after having noticed that permutations P and Q are really similar, since their only distinction is the constant addition phase. Even in that step, the distinction is really thin: a different constant is added on only two different bytes. Thus, we can hope that the amount of differences will remain low when setting a differential trail.

Since using truncated differentials is very handy when attacking `AES`-like permutations, we will only keep track of active and inactive bytes through the path. Also, preparing for the utilization of Super-Sbox attacks, we aim for a differential path in which the costly part lies in the middle, and the cheap parts on the

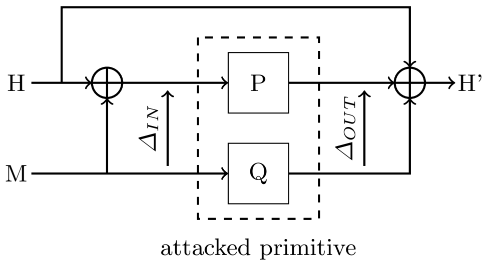


Fig. 4. The differential path keeps track of the differences between permutations P and Q

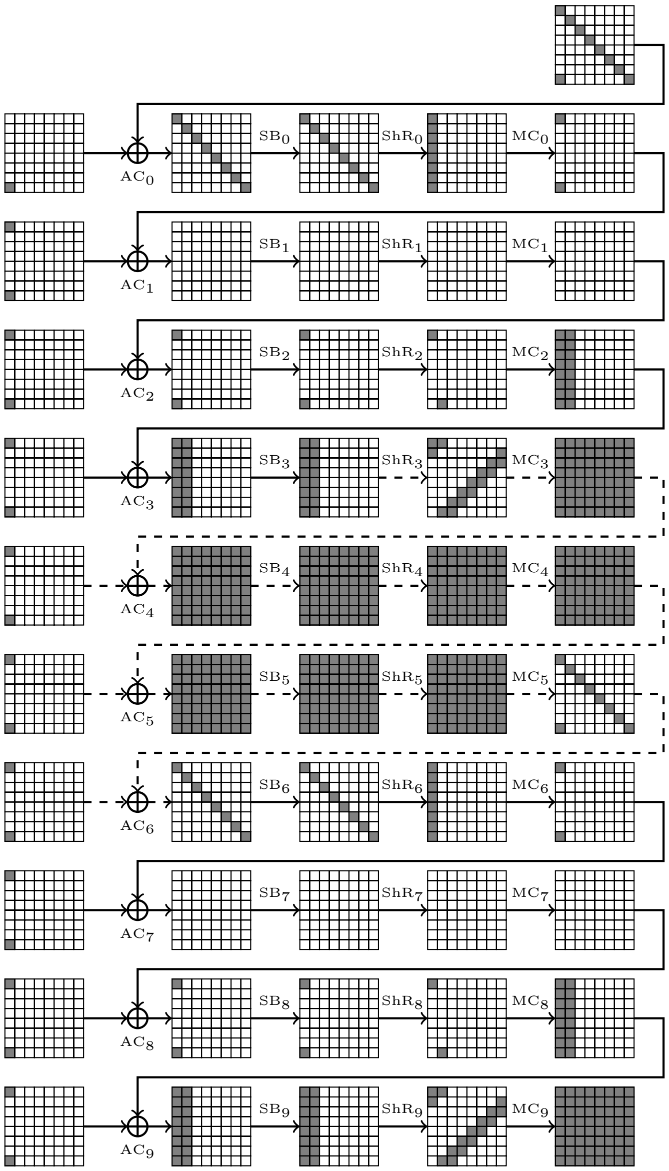


Fig. 5. 10-round differential path between P and Q for Grøstl-256. Each cell represents a byte and a gray cell stands for an active byte. The controlled rounds are depicted with dashed lines. The matrices on the left represent the differences incorporated during the AC layers.

sides. In Figure 5, we provide a differential path between the permutations P and Q of the `Grøstl`-256 compression function for the 10-round version. Note that only one difference is incorporated during AC_0 since the constant added in P is 0.

4.3 Deriving a Distinguisher for `Grøstl`

In the following, our goal is to distinguish the `Grøstl` compression function from an ideal primitive on the same domain. As shown in Figure 4, once the differential path settled, we find a valid pair of internal states (A, B) such that

$$A \oplus B = \Delta_{IN}$$
$$P(A) \oplus Q(B) = \Delta_{OUT}$$

where Δ_{IN} and Δ_{OUT} are respectively the input and output truncated differences. We then set $H = A \oplus B$ and $M = B$ and we obtain

$$h(H, M) = P(A) \oplus Q(B) \oplus A \oplus B = \Delta_{IN} \oplus \Delta_{OUT}.$$

We will show that Δ_{IN} and Δ_{OUT} are always maintained in a small subspace of x and y elements respectively. As a consequence, $\Delta_{IN} \oplus \Delta_{OUT}$ will also belong to a small subspace of the full output domain. Said in other words, we will be able to compute outputs of the $2n$-bit compression function that always belong to a predetermined set of at most $k = x \cdot y$ elements. In the ideal case, one such input/output property should not be obtained with substantially less than $2^{2n}/k$ compression function calls. Unlike the previously known distinguishers that find partially colliding outputs for `AES`-like permutations, the one we describe here is more "preimage" oriented.

One can go further and even try to distinguish the `Grøstl` compression function from its internal construction

$$h(H, M) = P(H \oplus M) \oplus Q(M) \oplus H = (P(A) \oplus A) \oplus (Q(B) \oplus B)$$

assuming P and Q as ideal permutations. We will compute pairs (H, M) such that H belong to a small subspace of x elements and H' to a small subspace of $k = x \cdot y$ elements. In the ideal case, one may think that the best attack can obtain such a property this with $\sqrt{2^{2n}/k}$ computations by performing a birthday method with the two branches. However, this is not the case here because a strong constraint on the input H exists (see the limited birthday distinguishers [15]) and the best known complexity to obtain the input/output property with ideal permutations P and Q is $2^{2n}/(k \cdot x)$ computations. It is important to remark that this type of distinguisher is new since the already known ones are structural, i.e. they already consider P and Q as ideal permutations.

While formally defining a distinguisher for a keyless primitive is difficult [40], we argue that the property we exhibit here works for any choice of Sbox, MixColumns function or AddConstant positions for example. Note that such keyless primitive distinguishers have already been utilized in [26, 15].

Let Grøstl(a) denote the Grøstl hash function for which the constant addition in Q is $i \oplus a$ instead of $i \oplus \texttt{0xff}$. Clearly, when choosing $a > \texttt{0x1a}$, we ensure that the constant values added in P and Q are always distinct and each member of this family of Grøstl hash functions should have the same security as Grøstl(0xff). Overall, for each member of the family, the attacker can exhibit with good probability an output maintained in the set of k elements, while the input H belongs to the subspace of x elements. Thus, if we are queried to distinguish the Grøstl compression function instantiated with permutations corresponding to Grøstl(a) from the same construction with random permutations P and Q, we have a very good probability to succeed. It shows a weakness in the Grøstl design philosophy.

5 Results

In this section we present some of our results on the compression functions of ECHO and Grøstl-256. For the complete results, and the differential paths concerning the internal permutation of ECHO, the single-pipe version ECHO-SP, or Grøstl-512 compression function, we refer to the extended version of this article [36]. Moreover, we also provide in the Appendix A a study of the amount of freedom degrees available during the attacks.

5.1 ECHO

Compression function distinguishers. We provide here the first distinguishers for reduced ECHO compression functions. In the case of ECHO-256, we use the 4-round differential path from Figure 6 which is derived from the 7-round core path. One can find a solution with 2^{64} computations and memory (2^{39} valid candidates can be generated by the attacker and 2^{167} if the salt is controlled as well). In the case of ECHO-512, we use the 6-round differential path from Figure 7 for which a solution can be found with 2^{96} computation and 2^{64} memory (2^{71} valid candidates can be generated by the attacker and 2^{199} if the salt is controlled as well). In both cases, we obtain compression function outputs colliding on 2 predetermined words (i.e. 256 bits) and this should require 2^{128} computations in the ideal case.

Collision attacks. We provide here the first collision attacks for reduced ECHO compression functions. In the case of ECHO-256, we use a special 3-round differential path depicted in Figure 8. In this trail, the start-from-the-middle technique can still be used and the only part uncontrolled is the first AES round of the very first BIG SubBytes layer. However, since we use the backward transition $\texttt{D} \Leftarrow \texttt{1} \Leftarrow \texttt{C}$, this layer will behave as expected with probability 1. Then, the feedforward is applied and only four 128-bit words will be active, each containing 4 active bytes at the exact same positions (truncated differential of type D). Finally, since the four columns of 128-bit words are xored together to obtain the output chaining variable, a collision can occur if the truncated differences are

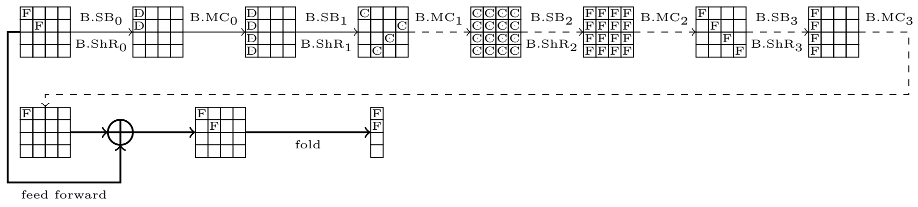


Fig. 6. 4-round differential path for the ECHO-256 compression function distinguisher. The controlled rounds are depicted with dashed lines.

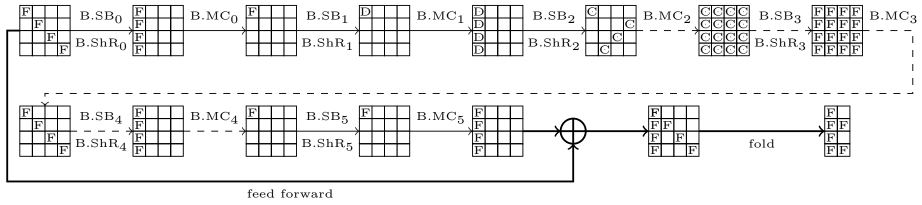


Fig. 7. 6-round differential path for the ECHO-512 compression function distinguisher. The controlled rounds are depicted with dashed lines.

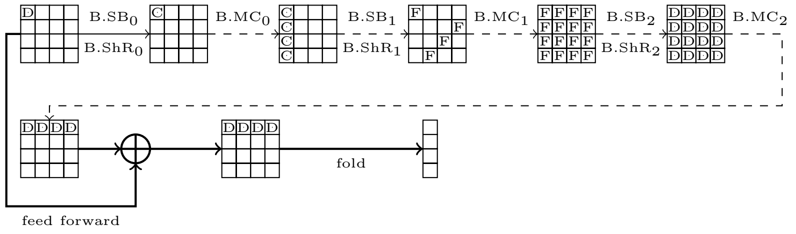


Fig. 8. 3-round differential path for the ECHO-256 compression function semi-free-start collision attack. The controlled rounds are depicted with dashed lines.

erased on the 4 byte positions. Thus, in order to get a semi-free-start collision, one should therefore test 2^{32} candidates (we have enough freedom degrees since one can generate 2^{143} valid candidates for the whole trail and 2^{271} if the salt is chosen by the attacker). However, the minimum cost for using the start-from-the-middle attack for ECHO is 2^{64} memory and precomputation. Thus, the overall cost is 2^{64} computations and memory in order to find one single semi-free-start collision for 3 rounds.

In the case of ECHO-512, we use the 4-round differential path from Figure 9 for which a solution can be found with 2^{96} computations and 2^{64} memory (2^{71} valid candidates can be generated by the attacker and 2^{199} if the salt is controlled as well). Then, before the feedforward is applied, one active 128-bit word remains

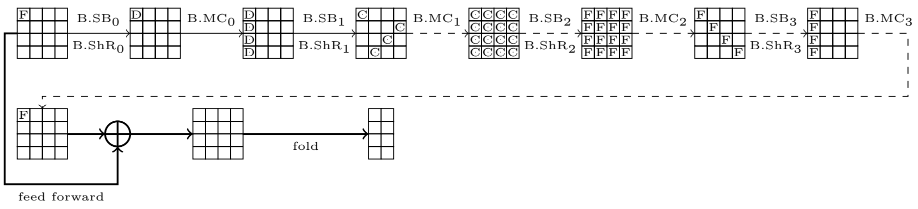


Fig. 9. 4-round differential path for the ECHO-512 compression function semi-free-start collision attack. The controlled rounds are depicted with dashed lines.

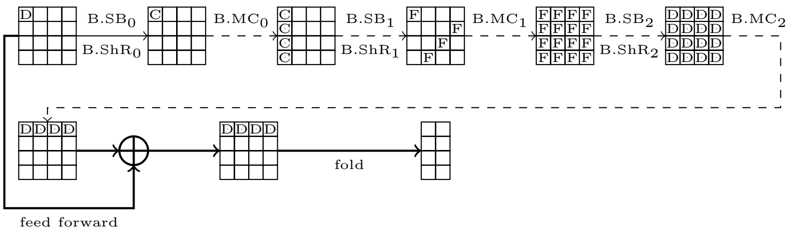


Fig. 10. 3-round differential path for the ECHO-512 compression function semi-free-start collision attack. The controlled rounds are depicted with dashed lines.

in the output of the permutation. In order to erase this difference and obtain a semi-free-start collision, this should be repeated 2^{128} times and the total cost of the attack is then 2^{224} computations and 2^{64} memory. Thus, this attack is valid only in the chosen-salt attacker model (otherwise the number of available freedom degrees is not sufficient). Since the collision happens before the final compression phase, this semi-free-start collision attack applies with the same complexity to ECHO-256 compression function.

When the attacker can not control the salt value, the 3-round attack from Figure 10 applies. Namely the reasoning is exactly the same as for the 256-bit case with Figure 8, except that we have 4 additional bytes to collide during the feed-forward phase. Finally, finding a semi-free-start collision for the 3-round reduced ECHO-512 compression function requires 2^{96} computations and 2^{64} memory.

5.2 Grøstl

We use the Super-Sbox technique to find two 512-bit internal states such that the 10-round differential path from Figure 5 between permutations P and Q is verified. Namely, one can find internal state values for P and Q verifying the truncated differential trail from the output of SB_3 up to the input of SB_6 with one computation on average. However, the two $8 \mapsto 1$ MixColumns transitions through MC_2 and the $8 \mapsto 2$ transition through MC_6 during the uncontrolled rounds happen with probability $2^{-2\times56} = 2^{-112}$ and 2^{-48} respectively. Also, 2 byte differences must be erased during both AddConstant functions AC_2 and AC_7 which adds another $2^{-4\times8} = 2^{-32}$ factor. Overall, one can find a valid candidate for the whole path with only $2^{112+48+32} = 2^{192}$ computations (an amount of 2^{64} memory is required by the utilization of the Super-Sbox technique).

The freedom degrees analysis from Appendix A shows that for the path from Figure 5, one can expect to obtain one solution with good probability. Indeed, when the success probability for a random input pair to verify the trail is 2^{-z}, we have 2^{z-1} freedom degrees available. We argue in the Appendix that it is sufficient for the attack to be considered as valid.

The distinguisher for `Grøstl`. In order to mount the distinguisher for `Grøstl`, one has to analyze the amount k of reachable output difference values. In the differential path from Figure 5, we have 16 active bytes just before applying the very last MixColumns layer MC_9. Since the MixColumns layer is fully linear, the amounts of reachable difference values on its input and on its output are equal. Thus, we can deduce that at most $y = 2^{16\times8} = 2^{128}$ distinct output differences can be reached on the output of the differential trail. Regarding the input of the path, the same reasoning gives us that at most $x = 2^{8\times8} = 2^{64}$ distinct input differences can be reached. Note that the difference inserted during AC_0 can be ignored since it is the last layer when computing backward (the difference value on that byte will always be equal to the constant added, i.e. `0xff`). Also, it is easy to verify that the differences on the output of SB_0 are always the same (since MixColumns is linear). Thus, since the inverse of the `AES` Sbox has the property that only 2^7 distinct output differences can be reached when the input difference is fixed, we can conclude that Δ_{IN} can go through a maximum of $x = 2^{8\times7} = 2^{56}$ distinct values.

To summarize, the output chaining variable $H' = h(H, M) = \Delta_{IN} \oplus \Delta_{OUT}$ is limited to a set of at most $k = 2^{128+56} = 2^{184}$ values, with H being limited to a set of at most $x = 2^{56}$ values. For an ideal 512-bit compression function, reaching any element of this set should require $2^{512-184} = 2^{328}$ operations. With 2^{80} and 2^{192} computations respectively (and 2^{64} memory), we finally conclude that our attack can distinguish 9-round reduced or the full 10-round compression function of `Grøstl`-256 from a random 512-bit compression function. One can even distinguish h from the compression function construction with P and Q assumed ideal since the best known attack requires $2^{512-184-56} = 2^{272}$ computations.

Note that structural distinguishers (i.e. working for randomly chosen permutations P and Q) already exist for `Grøstl`. For example, just like in the Davies-Meyer construction, one can very easily find fixed points for the compression function. Yet, as explained in Section 4.3, we believe that our distinguishers are very interesting because they exploit the real differential properties of the internal permutations P and Q, which is essential in order to appropriately evaluate the security margin in terms of number of rounds. Moreover, such structural attacks can not distinguish h from the compression function construction with P and Q assumed ideal.

6 Conclusion

In this article, based on recent advances on AES-like permutations studies, we provided a new cryptanalysis of ECHO and Grøstl, two second-round SHA-3 candidates. In particular, in the case of Grøstl, we introduce a new cryptanalysis technique: the internal differential attack. Overall, we obtain the best cryptanalysis results known so far for both ECHO and Grøstl. We are able to derive a distinguisher for the full (10 rounds) 256-bit version of the Grøstl compression function. This work also shows that designers must be careful when building a function with parallel branches computations as the internal differential paths may lead to unexpected attacks.

Acknowledgments

The author would like to thank the Grøstl team, Henri Gilbert, Yannick Seurin and the CRYPTO 2010 committee for their helpful comments. Also, many thanks to Elmar Tischhauser, Jorge Nakahara and Kota Ideguchi for pointing me an omission in the complexity computation for the full ECHO internal permutation distinguisher.

References

1. Barreto, P.S.L.M.: An observation on Grøstl. Comment submitted to the NIST hash function mailing list, hash-forum@nist.gov, `http://www.larc.usp.br/~pbarreto/Grizzly.pdf`
2. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
3. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (2008), `http://crypto.rd.francetelecom.com/echo/`
4. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions: HAIFA. In: Second NIST Cryptographic Hash Workshop (2006)
5. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (2008)
6. Biryukov, A. (ed.): FSE 2007. LNCS, vol. 4593. Springer, Heidelberg (2007) (revised selected papers)
7. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi (ed.) [16], pp. 231–249
8. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
9. Cramer, R. (ed.): EUROCRYPT 2005. LNCS, vol. 3494. Springer, Heidelberg (2005)
10. Daemen, J., Rijmen, V.: The Design of Rijndael. In: Information Security and Cryptography. Springer, Heidelberg (2002), ISBN 3-540-42580-2
11. Damgård, I.: A Design Principle for Hash Functions. In: Brassard (ed.) [8], pp. 416–427

12. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 71–82. Springer, Heidelberg (1996)
13. Dunkelman, O. (ed.): FSE 2009. LNCS, vol. 5665. Springer, Heidelberg (2009)
14. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (2008), `http://www.groestl.info`
15. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-like Permutations. In: FSE 2010. LNCS. Springer, Heidelberg (to appear 2010), `http://eprint.iacr.org/2009/531`
16. Halevi, S. (ed.): CRYPTO 2009. LNCS, vol. 5677. Springer, Heidelberg (2009)
17. Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.): SAC 2009. LNCS, vol. 5867. Springer, Heidelberg (2009)
18. Kelsey, J.: Some notes on Grøstl. Comment submitted to the NIST hash function mailing list, hash-forum@nist.gov, `http://ehash.iaik.tugraz.at/uploads/d/d0/Grostl-comment-april28.pdf`
19. Khovratovich, D.: Cryptanalysis of Hash Functions with Structures. In: Jocobson Jr., M.J., et al. (eds.) [17], pp. 108–125
20. Knudsen, L.R., Rechberger, C., Thomsen, S.S.: The Grindahl Hash Functions. In: Biryukov (ed.) [6], pp. 39–57
21. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 315–324. Springer, Heidelberg (2007)
22. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
23. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui (ed.) [24], pp. 126–143
24. Matsui, M. (ed.): ASIACRYPT 2009. LNCS, vol. 5912. Springer, Heidelberg (2009)
25. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In: Matsui (ed.) [24], pp. 106–125
26. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jocobson Jr., M.J., et al. (eds.) [17], pp. 16–35
27. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman (ed.) [13], pp. 260–276
28. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound Attacks on the Reduced Grøstl Hash Function. In: Pieprzyk (ed.) [37], pp. 350–365
29. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard (ed.) [8], pp. 428–446
30. Minier, M., Phan, R.C.-W., Pousse, B.: Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks. In: Preneel (ed.) [38], pp. 60–76
31. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard (April 1995), `http://csrc.nist.gov`
32. National Institute of Standards and Technology. FIPS PUB 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce (November 2001)

33. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register 27(212), 62212–62220 (November 2007), `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17)
34. Nguyên, P.Q. (ed.): VIETCRYPT 2006. LNCS, vol. 4341. Springer, Heidelberg (2006)
35. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
36. Peyrin, T.: Improved Differential Attacks for ECHO and Grostl. Cryptology ePrint Archive, Report 2010/223 (2010), `http://eprint.iacr.org/`
37. Pieprzyk, J. (ed.): CT-RSA 2010. LNCS, vol. 5985. Springer, Heidelberg (2010)
38. Preneel, B. (ed.): AFRICACRYPT 2009. LNCS, vol. 5580. Springer, Heidelberg (2009)
39. RIPE. Integrity Primitives for Secure Information Systems. In: Bosselaers, A., Preneel, B. (eds.) RIPE 1992. LNCS, vol. 1007. Springer, Heidelberg (1995)
40. Rogaway, P.: Formalizing Human Ignorance. In: Nguyen (ed.) [34], pp. 211–228
41. Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992), `http://www.ietf.org/rfc/rfc1321.txt`
42. Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621. Springer, Heidelberg (2005)
43. Wagner, D.: A Generalized Birthday Problem. In: Yung (ed.) [46], pp. 288–303
44. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup (ed.) [42], pp. 17–36
45. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer (ed.) [9], pp. 19–35
46. Yung, M. (ed.): CRYPTO 2002. LNCS, vol. 2442. Springer, Heidelberg (2002)

Appendix A: The Amount of Freedom Degrees

Once a differential path settled, a point has to be clarified: the amount of freedom degrees available to the attacker. Indeed, one has to evaluate how much valid pairs can be found for the whole differential trail. We want to ensure that enough solutions for the controlled rounds exist so that we have a good probability that at least one of them will fulfill the entire differential characteristic. Moreover, when searching for semi-free-start collisions, we may even go further since we may require an important amount of valid candidates for the entire differential path.

Freedom Degrees for ECHO

We use the same counting reasoning than in [15], except that we have to precisely evaluate what is the freedom degrees consumption for the various 2 AES-round differential transitions as well (in [15] it was implicitly assumed that all the BIG SubBytes transitions were $\texttt{F} \rightarrow \texttt{F}$, thus happening with probability very close to 1 and consuming no freedom degrees). For example, let us take the $\texttt{D} \rightarrow \texttt{1}$ transition through the BIG SubBytes in the forward direction: we require one AES MixColumns transition $4 \rightarrow 1$ which happens with probability 2^{-24}. Thus, if we have k valid candidates on the input, we obtain $k \times 2^{-24}$ valid candidates on the output of this layer and we consumed 2^{24} freedom degrees. The amount of freedom degrees consumed during a transition is the invert of the probability of success of this transition. Thus, with Table 2, it is very easy to compute the freedom degrees consumption for all the AES round transitions considered so far.

We illustrate the counting method by applying it to the example of the 7-round path from Figure 3. First, note that the start-from-the-middle attack will find **all** the possible internal states such that the controlled rounds are verified. We start from state $\mathsf{B.MC}_3^{in}$ (located between $\mathsf{B.ShR}_3$ and $\mathsf{B.MC}_3$). This state is fully active which means that we can start with $2^{2048\times2-1} = 2^{4095}$ distinct pairs. When going forward, the $\mathsf{B.MC}_3$ transition happens with probability $2^{-4\times24\times16} = 2^{-1536}$ and the transition through $\mathsf{B.MC}_4$ happens with probability $2^{-24\times16} = 2^{-384}$. All the other layers are verified with probability one so the forward computation consumes $2^{1536+384} = 2^{1920}$ freedom degrees. Then, during the backward computation, the sixteen $\mathtt{C} \leftarrow \mathtt{F} \leftarrow \mathtt{F}$ transitions through $\mathsf{B.SB}_3$ happen with probability $2^{-16\times96} = 2^{-1536}$ according to Table 2 ($\mathtt{C} \leftarrow \mathtt{F}$ with probability 2^{-96} and $\mathtt{F} \leftarrow \mathtt{F}$ with probability 1). Also, the four $\mathtt{D} \leftarrow \mathtt{1} \leftarrow \mathtt{C}$ transitions through $\mathsf{B.SB}_2$ happen with probability $2^{-4\times24} = 2^{-96}$ ($\mathtt{D} \leftarrow \mathtt{1}$ with probability 1 and $\mathtt{1} \leftarrow \mathtt{C}$ with probability 2^{-24}). Then, the BIG MixColumns transitions through $\mathsf{B.MC}_2$ are verified with probability $2^{-4\times4\times24} = 2^{-384}$ and through $\mathsf{B.MC}_1$ with probability $2^{-4\times24} = 2^{-96}$. All the other layers in the backward direction are verified with probability one. Overall, the backward computation consumes $2^{1536+384+96+96} = 2^{2112}$ freedom degrees. We can finally conclude that we started with 2^{4095} pairs from which only a factor $2^{-1920-2112} = 2^{-4032}$ will be valid for the whole differential path. One can then generate 2^{63} distinct valid pairs for the 7-round path from Figure 3.

Note that the differential paths we use are just instances among a family of good differential trails. For example, in the case of the 7-round path from Figure 3, instead of placing the active word on the top left position of $\mathsf{B.MC}_0^{out}$ (between $\mathsf{B.MC}_0$ and $\mathsf{B.SB}_1$), one could place it in the 15 others locations. Those new paths present the same properties than the original one and this reasoning also applies to the active word located in $\mathsf{B.MC}_4^{out}$ (between $\mathsf{B.MC}_4$ and $\mathsf{B.SB}_5$). As a consequence, the attacker manages $16^2 = 2^8$ different core paths which provides him 2^8 additional freedom degrees.

Finally, some additional freedom degrees can be obtained if one considers that the salt value can be fully controlled by the attacker. While this scenario is not very relevant in practice, it is interesting to see what the attacker is able to do in such a situation. In the case of ECHO, the salt value is 128-bit long and it then directly adds 2^{128} supplementary freedom degrees. To conclude, the attacker can generate 2^{71} distinct valid pairs for 7-round paths like the one depicted in Figure 3, and 2^{199} if he controls the salt. The same method is used for all the differential trails for ECHO considered in this article.

Freedom Degrees for Grøstl

The case of Grøstl is easier to analyze since we don't have to handle word-wise and byte-wise truncated differentials at the same time. Yet, the same counting technique can be applied. Interestingly, for all the paths we provided concerning `Grøstl`, an attacker can expect only one solution for the whole trail with good probability. This explains why one can not really hope for a semi-free-start collision attack on reduced versions of `Grøstl` (such as 7 or 8-round versions) with the paths given. Or, said in other words, a semi-free-start collision attack may be mounted, but will only work with a low probability.

As an example, we provide here the freedom degrees analysis for the 10-round differential path from Figure 5. By starting from the fully active internal state located at the output of MC_4, we begin with about $2^{512\times 2-1} = 2^{1023}$ distinct pairs of internal state values. When going forward, the first freedom degrees consuming operation is the MC_5 transition which happens with probability $2^{-7\times 56-48} = 2^{-440}$. Then, one byte is erased during AC_6 while the transition through MC_6 happens with probability 2^{-48} and in total this round consumes $2^{8+48} = 2^{56}$ freedom degrees. Finally, the last consuming operation when computing forward is AC_7 for which two bytes have to be erased (2^{16}). When computing backward, the MixColumns functions MC_3 and MC_2 requires $2^{48\times 8} = 2^{384}$ and $2^{2\times 56} = 2^{112}$ freedom degrees respectively. Then, two bytes are erased through AC_2 and all the other differential transitions consume nothing since they are deterministic. Finally, we started with 2^{1023} freedom degrees from which only a fraction $2^{440+8+48+16+384+112+16} = 2^{1024}$ will verify the whole differential path. Thus, since this reasoning is done on average, an attacker has a good probability to obtain one single solution for the whole differential path.

Of course, one may argue that the attacker should have one more freedom degree to perform the attack. Yet, note that until really performed, most hash function attacks only have a certain success probability to actually find a solution. For example, in the case of `SHA-1`, even if very low, there is a probability that the known collision attacks eventually provide no solution. Therefore, with only a single freedom degree missing, we believe that the success probability is far sufficiently high in order to consider the attack as valid. Finally, if one really wants to increase this probability, additional freedom degrees could be found by defining a small family of `Grøstl` compression functions as explained in Section 4.3.