Improved Cryptanalysis of the Reduced `Grøstl` Compression Function, `ECHO` Permutation and AES Block Cipher

Florian Mendel[1], Thomas Peyrin[2], Christian Rechberger[1], and Martin Schläffer[1]

[1] IAIK, Graz University of Technology, Austria
[2] Ingenico, France
{thomas.peyrin@gmail.com,martin.schlaeffer@iaik.tugraz.at}

Abstract. In this paper, we propose two new ways to mount attacks on the SHA-3 candidates `Grøstl`, and `ECHO`, and apply these attacks also to the AES. Our results improve upon and extend the rebound attack. Using the new techniques, we are able to extend the number of rounds in which available degrees of freedom can be used. As a result, we present the first attack on 7 rounds for the `Grøstl`-256 output transformation[3] and improve the semi-free-start collision attack on 6 rounds. Further, we present an improved known-key distinguisher for 7 rounds of the AES block cipher and the internal permutation used in `ECHO`.

Keywords: hash function, block cipher, cryptanalysis, semi-free-start collision, known-key distinguisher

1 Introduction

Recently, a new wave of hash function proposals appeared, following a call for submissions to the SHA-3 contest organized by NIST [26]. In order to analyze these proposals, the toolbox which is at the cryptanalysts' disposal needs to be extended. Meet-in-the-middle and differential attacks are commonly used. A recent extension of differential cryptanalysis to hash functions is the rebound attack [22] originally applied to reduced (7.5 rounds) Whirlpool (standardized since 2000 by ISO/IEC 10118-3:2004) and a reduced version (6 rounds) of the SHA-3 candidate `Grøstl`-256 [14], which both have 10 rounds in total.

Many hash functions [1, 2, 6, 12, 14, 16, 17] use concepts or parts of the block cipher AES [25] as basic primitives, and research on AES-related hash functions is ongoing [15, 22, 27]. In this direction, a new attack model has been recently introduced for block ciphers [18]. In this model, the secret key is known to the adversary and the goal is to distinguish the block cipher from a random permutation. In particular, reduced versions of the AES have been studied in this setting [18,24] and recently, an attack on full AES-256 has been published [5].

[3] Note that the 7-round semi-free-start collision attack on `Grøstl`-256 in the preproceedings version of this paper does not have enough freedom to succeed, see Sect. 6.1.

Appeared in M.J. Jacobson Jr, V. Rijmen and R. Safavi-Naini (Eds.): SAC 2009, LNCS 5867, pp. 16–35.

In the rebound attack [22], two rounds of the state update transformations are bypassed by a match-in-the-middle technique using the available degrees of freedom in the state. The characteristic used in the attack is then constructed by moving the "most expensive" parts into these two rounds. The "cheaper" parts are then covered in an inside-out manner, called outbound phase. Other work in parallel to this explores the application of the rebound idea to other SHA-3 candidates [21, 28]. Recently, the rebound attack has been extended to attack the full compression function of Whirlpool [20] and LANE [19] by using the additional freedom of the key schedule (Whirlpool) or other parts of the state (LANE).

In this work, we present improved techniques, to use the freedom available in only a single state. The effect of both techniques we present are an extension of the number of rounds in which degrees of freedom can be used to improve the work from the two to four rounds. As a result, we present the best known attacks on the reduced Grøstl-256 permutation and output transformation (up to 7 out of 10 rounds), and also significantly improve upon the first known-key distinguisher [18] for 7-round AES and 7 rounds of the internal permutation used in ECHO. A summary of our results is given in Table 1.

Table 1. Summary of results for Grøstl, ECHO and AES.

target	rounds	computational complexity	memory requirements	type	section
Grøstl-256	6	2^{112}	2^{64}	semi-free-start collision	see [22]
	6	2^{64}	2^{64}	semi-free-start collision	Sect. 6.1
	7	2^{55}	-	permutation distinguisher	Sect. 6.1
	7	2^{56}	-	output transf. distinguisher	Sect. 6.1
ECHO	7	2^{896}	-	permutation distinguisher	see [2]
	7	2^{384}	2^{64}	permutation distinguisher	Sect. 6.3
AES	7	2^{56}	-	known-key distinguisher	see [18]
	7	2^{24}	2^{16}	known-key distinguisher	Sect. 6.2

2 Description of AES-based Primitives

In this paper, we show improved attack strategies for AES based cryptographic primitives. We apply the ideas to the Grøstl and ECHO hash function, and to the block cipher AES. In the following, we describe these functions in more detail.

2.1 AES

The block cipher Rijndael was designed by Daemen and Rijmen and standardized by NIST in 2000 as the Advanced Encryption Standard (AES) [25]. The AES follows the wide-trail design strategy [7, 8] and consists of a key schedule and state update transformation. Since we do not use the key schedule in our attack, we just describe the state update here.

In the AES, a 4×4 state of 16 bytes is updated using the following 4 round transformations, with 10 rounds for AES-128 and 14 rounds for AES-256:

- the non-linear layer SubBytes (SB) applies a S-Box to each byte of the state independently
- the cyclical permutation ShiftRows (SR) rotates the bytes of row j left by j positions
- the linear diffusion layer MixColumns (MC) multiplies each column of the state by a constant MDS matrix
- AddRoundKey (AK) adds the 128-bit round key K_i to the state

Note that a round key is added prior to the first round and the MixColumns transformation is omitted in the last round of AES. For a detailed description of the AES we refer to [25].

2.2 The `Grøstl` Hash Function

`Grøstl` was proposed by Gauravaram *et al.* as a candidate for the SHA-3 competition [14]. It is an iterated hash function with a compression function built from two distinct permutations P and Q. `Grøstl` is a wide-pipe design with proofs for the collision and preimage resistance of the compression function [13]. A t-block message M (after padding) is hashed using the compression function $f(H_{i-1}, M_i)$ and output transformation $g(H_t)$ as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, M_i) = H_{i-1} \oplus P(H_{i-1} \oplus M_i) \oplus Q(M_i) \quad \text{for } 1 \le i \le t \\ h &= g(H_t) = trunc(H_t \oplus P(H_t)), \end{aligned}$$

The two permutations P and Q are constructed using the wide trail design strategy. The design of the two permutations is very similar to AES, instantiated with a fixed key input. Both permutations of `Grøstl`-256 update an 8×8 state of 64 bytes in 10 rounds each. The round transformations of `Grøstl`-256 are briefly described here:

- AddRoundConstant (AC) adds different one-byte round constants to the 8×8 states of P and Q
- the non-linear layer SubBytes (SB) applies the AES S-Box to each byte of the state independently
- the cyclical permutation ShiftBytes (ShB) rotates the bytes of row j left by j positions
- the linear diffusion layer MixBytes (MB) multiplies each column of the state by a constant MDS matrix

2.3 The `ECHO` Hash Function

The `ECHO` hash function is a SHA-3 proposal submitted by Benadjila *et al.* [2]. It is also a wide-pipe iterated hash function and uses the HAIFA [3] domain extension algorithm. Its compression function uses an internal 2048-bit permutation that can be seen as a big AES manipulating 128-bit words instead of bytes. More precisely, an appropriately padded t-block message M and a salt s are hashed using the compression function $f(H_{i-1}, M_i, c_i, s)$ where c_i is a bit counter:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, M_i, c_i, s) \quad \text{for } 1 \leq i \leq t \\ h &= trunc(H_t), \end{aligned}$$

The compression function of ECHO is built upon a 2048-bit permutation F, composed of 8 rounds (resp. 10 rounds) in the case of a 256-bit output (resp. 512-bit output). Its internal state can be modeled as a 4×4 matrix of 128-bit words. The concatenation of the input chaining variable and the incoming message block are the plaintext input of the permutation F which is tweaked by the input counter c_i and the salt s. A feed-forward of the plaintext is then applied to the internal state V:

$$V = F_{c_i,s}(H_{i-1}||M_i) \oplus (H_{i-1}||M_i)$$

and the 512-bit output chaining variable H_i is the xor of all the 128-bit word columns of V for a 256-bit hash digest size. In the case of a 512-bit hash value, the 1024-bit output chaining variable H_i is the xor of the two left and the two right 128-bit word columns of V.

A permutation round is very similar to an AES round, except that 128-bit words are manipulated. One round is the composition of three sub-functions $BigMC \circ BigShR \circ BigSW$:

- the non-linear layer BigSubWords (BigSW) applies two AES rounds to each of the 16 128-bit words of the internal state. The round keys, always different, are composed of the salt value and a counter value initialized by c_i.
- the cyclical permutation BigShiftRows (BigShR) rotates the location in the matrix of the 128-word of row j left by j positions
- the linear diffusion layer BigMixColumns (BigMC) multiplies each column of the state by a constant MDS matrix

3 Basic Attack Properties

Before describing attacks for Grøstl, ECHO and AES in detail, we give an overview of the round transformation properties used by the attacks. Since we mostly use Grøstl to describe the attacks and the properties of MixColumns and MixBytes are rather similar, we use MixBytes to describe their common properties in the following. We will use differential properties of the SubBytes and MixBytes transformation and exploit the diffusion property of both, ShiftBytes (ShiftRows) and MixBytes. Together, this leads to an efficient guess-and-determine strategy for both, differences and values at the input and output of SubBytes and MixBytes.

Since we exploit the differential properties of SubBytes and MixBytes, we define the notation and state variables according to these two transformations.

We denote the SubBytes layer of round i by $\mathsf{SB_i}$, its input state by $\mathsf{SB_i^{in}}$ and the output state by $\mathsf{SB_i^{out}}$. An equivalent notation is used for the MixBytes transformation. The MixBytes transformation of round i is denoted by MB_i, its input state by $\mathsf{MB_i^{in}}$ and the output state by $\mathsf{MB_i^{out}}$. We will use MC_i for the MixColumns transformation of `ECHO` and AES. Further, counting from 0, we denote the byte in row r and column c by $[r, c]$, i.e. $\mathsf{SB_i^{in}[r, c]}$ for the input of the S-box in round i.

3.1 Improving on the Rebound Attack

The main idea of the rebound attack [22] is to start close to the middle of a (truncated) differential path, connect using the available degrees of freedom in the middle and finally propagate outwards. Our attack works rather similar for `Grøstl`-256, `ECHO` and AES, and in the following we use `Grøstl`-256 to describe the attacks and discuss then, the application to `ECHO` and AES. Similar to the rebound attack, we start with a truncated differential path with a full active state in the middle of the trail. Fig. 1 shows the truncated differential path used in both permutations P and Q of our improved attack on `Grøstl`-256. In the rebound attack, the middle part of the differential trail is solved first for both differences and values by exploiting the available degrees of freedom (inbound phase). Then, differences and values are propagated outwards probabilistically (outbound phase) to find semi-free-start collisions, free-start collisions, or non-random properties of the permutations or compression function.

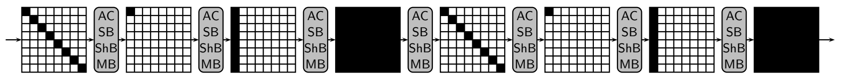

Fig. 1. The position of active bytes of the 7 round differential path for both permutations P and Q.

In this work, we improve on the middle part of the attack where we exploit the available degrees of freedom of the state values and differences. The idea is to first find differences and values for the middle (4-round) part of the differential trail, with the following number of active bytes at SubBytes:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8 \xrightarrow{r_4} 1$$

3.2 Exploiting Properties of the Round Transformations

In this section, we briefly describe which properties of the round transformations are used for the attacks in the following sections. Note that some used properties, especially those of MixBytes, are specific to a truncated differential path with a minimum number of active S-boxes such as the one given in Fig. 1.

SubBytes. In our attacks, we use some differential properties of the AES S-box. Most of these properties can simply be verified by computing the differential distribution tables (DDT) [4] of the S-box (or inverse S-box).

– for a given input (or output) difference of the AES S-box, the number of possible output (or input) differences is restricted to 127.
– for a given input and output difference, the number of possible input values is limited to either 2 or 4 values.
– for a given input and output difference, the AES S-box behaves linear due to the fact that there are only 2 or 4 solutions per S-box (see Section 4.2 for more details).

In the following sections, we use some differential S-box tables to efficiently carry out the attacks. We call S_F^δ the table that contains all input byte pairs (a, b), such that we get the difference δ at the output of the AES S-box, i.e. such that $Sbox(a) \oplus Sbox(b) = \delta$. Each table S_F has 256 entries with 127 possible input differences $a \oplus b$ of the S-box. More precisely, for any difference $\delta \neq 0$ on the output of the S-box, 129 input differences are not possible, 126 differences have two candidates and 1 difference has 4 candidates. The table S_B^δ contains all the output byte pairs (a, b), such that we get the difference δ at the input of the S-box, i.e. after the application of the inverse AES S-box. For a fast implementation of the attacks, these tables are precomputed and sorted accordingly.

ShiftBytes. The ShiftBytes (or ShiftRows) transformation moves bytes and thus, differences to different positions but does not change their value. The good diffusion property of ShiftRows allows us to choose certain differences and values of the subsequent MixBytes layer independently. Assume we have one active column in MixBytes. Then, we get after the adjacent ShiftRows application one active byte in each new column. Hence, we can determine these single active bytes by the subsequent MixBytes transformation independently.

MixBytes. In the case of MixBytes (or MixColumns), we use the property of an $n \times n$ MDS matrix that, given any n bytes of input and output, the other n bytes can be uniquely determined. Since MixBytes is linear, this also holds for differences. In the following attacks, we use differential paths with a minimum number of active S-boxes. Hence, also the number of differences in the MixBytes transformation is minimal and every active MixBytes operation contains zero differences in exactly 7 (3 for MixColumns) input/output bytes. It follows, that choosing a single byte difference uniquely determines all other 8 (4 for MixBytes) differences.

Hence, for a fixed position of active bytes, we get 255 possibilities for the difference propagation of MixBytes (bundles in [9]). These cases can be precomputed and stored in tables. We call M_F^i the table that contains all possible input differences of MixBytes (or MixColumns), such that we get only one non-zero byte at row i in the output. We call M_B^i the same table but for the inverse of the MixBytes (or MixColumns) transformation. Since the MixBytes transformation is linear, the same tables can be used for values and differences.

3.3 Known-key Distinguishers

In the following, we will describe known-key distinguisher attacks against AES and the internal permutations used in `Grøstl` and `ECHO`. We refer to [18] for the details of this setting. However, in this paper, our distinguishers will consist in finding a pair of plaintext for the keyed permutation (when the key is randomly chosen but known by the attacker) such that some plaintext and ciphertext words contain no difference. For the distinguishing attack to be valid, the complexity should be lower than expected in the case of a random permutation. Assume an n-bit permutation with differences in i bytes of the plaintext and in i bytes of the ciphertext. Then, assuming that the positions of the byte-differences are fixed, the complexity of the generic attack is greater or equal (depending on the values of i and n) to $2^{(n-8\cdot i)/2}$.

4 A Linearized Match-in-the-Middle Attack

In this section, we present a method which allows us to find a state pair with differences according to the truncated differential path of Fig. 1 with a complexity of about 2^{48}. The main idea is to first search for differences according to the 4-round middle part ($1 \to 8 \to 64 \to 8 \to 1$) of the path. We can find such differences with a complexity of about 1 by guess and determine (see Section 4.1). In the second phase, we try to solve for the corresponding values of the state. The main idea is that we can do this *linearly*. Since the differential of each S-box is fixed we get either 2 or 4 possible values for the AES S-box (see Section 3.2). In these cases, the S-box behaves linearly and hence, we can find the correct values by solving a linear system of equations (see Section 4.2). Note that we need to repeat the solving phase with new differences if no solution was found.

4.1 Filtering for Differential Paths

In this section, we filter for candidate differences which follow the middle part ($1 \to 8 \to 64 \to 8 \to 1$) of the differential path of Fig. 1 with a high probability. Fig. 2 shows the corresponding round transformations and the differential path in detail. In the attack, we use properties of the SubBytes (SB) and MixBytes (MB) transformations to filter for differential paths. Hence, we are interested in the input and output of these transformations. The first and second column show differences at the input and output of the S-boxes ($\mathsf{SB}_\mathsf{i}^\mathsf{in}$ and $\mathsf{SB}_\mathsf{i}^\mathsf{out}$), and column three and four show differences at the input and output of the MixBytes transformations ($\mathsf{MB}_\mathsf{i}^\mathsf{in}$ and $\mathsf{MB}_\mathsf{i}^\mathsf{out}$). To determine possible input and output differences of these two transformations, we use their corresponding lookup tables M_F^j, M_B^j, S_F^δ and S_B^δ (see Section 3.2).

Column 1. We start with the differences of the first column (marked by "1" in state $\mathsf{MB}_2^\mathsf{in}$ and $\mathsf{MB}_2^\mathsf{out}$) of the MixBytes operation of round 2 (MB_2). Since 7 input byte differences are required to be zero, choosing one of the remaining

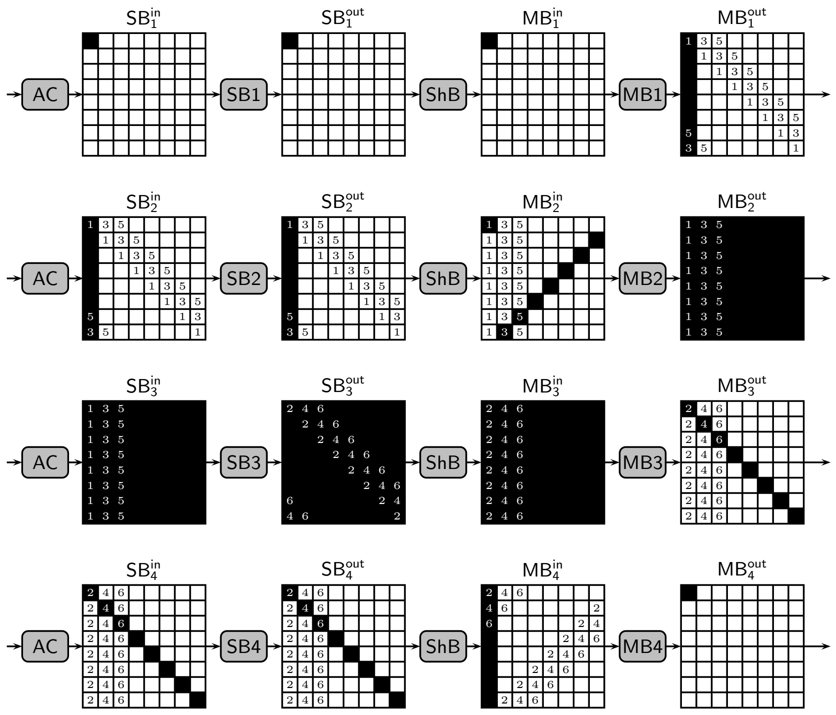

Fig. 2. Filtering for differential paths.

9 non-zero differences, uniquely determines all other differences of MB_2. Since the ShiftBytes and AddRoundConstant operations are linear, we get the same differences for the bytes marked by "1" in states $\mathsf{SB}_2^{\mathsf{out}}$ and $\mathsf{SB}_3^{\mathsf{in}}$. It follows that we can choose from 255 non-zero differences for the first byte of $\mathsf{SB}_3^{\mathsf{in}}$, and this choice determines all differences marked by "1" between state $\mathsf{SB}_2^{\mathsf{out}}$ and $\mathsf{SB}_3^{\mathsf{in}}$.

Column 2. Next, we continue with the differences of the first column of MB_3 (marked by "2" in states $\mathsf{MB}_3^{\mathsf{in}}$ and $\mathsf{MB}_3^{\mathsf{out}}$). Again, 7 differences of MB_3 are zero and choosing one byte determines all differences of the first column of MB_3. Note that the input of the first column of SB_3 and thus, the difference of $\mathsf{SB}_3^{\mathsf{in}}[0,0]$, has already been fixed in the previous step. Due to the differential behavior of the AES S-box (see Section 3.2), we can choose from only 127 differences for the corresponding output byte of SB_3 ($\mathsf{SB}_3^{\mathsf{out}}[0,0]$). Choosing one of these possible 127 differences uniquely determines all differences marked by "2" between states $\mathsf{SB}_3^{\mathsf{out}}$ and $\mathsf{SB}_4^{\mathsf{in}}$.

Column 3. Then, we continue with the second column of MB_2 (marked by "3" in states $\mathsf{MB}_2^{\mathsf{in}}$ and $\mathsf{MB}_3^{\mathsf{out}}$). Again, 7 bytes of the input differences are required to be zero. Additionally, one output difference of SB_3 ($\mathsf{SB}_3^{\mathsf{out}}[1,1]$) has already been fixed due to **Column 2**. Again, we can only choose from 127 possible input differences for SB_3 ($\mathsf{SB}_3^{\mathsf{in}}[1,1]$) and get 127 possible differences for the bytes marked by "3" between $\mathsf{SB}_3^{\mathsf{in}}$ and $\mathsf{SB}_2^{\mathsf{out}}$.

Column 4-5. We proceed with the second column of MB_3, marked by "4" in states $\mathsf{MB}_3^{\mathsf{in}}$ and $\mathsf{MB}_3^{\mathsf{out}}$. Note that the input bytes of two S-boxes ($\mathsf{SB}_3^{\mathsf{in}}[0,1]$ and $\mathsf{SB}_3^{\mathsf{in}}[7,0]$) have already been fixed due to **Column 1** and **Column 3**. These two input differences restrict the number of possible differences for the output of SB_3 (bytes marked by "4") to about $256/2^2 = 64$ values. We continue with the third column of MB_2 (marked by "5"). Two output differences of the corresponding S-box SB_3 have already been fixed and thus, we can choose from about 64 possible differences for the input bytes marked by "5" in $\mathsf{SB}_3^{\mathsf{in}}$ as well.

Column 6-16. This procedure continues for all 8 columns of each of the two MixBytes transformations MB_2 and MB_3. The approximate number of possible S-box differences for $\mathsf{SB}_3^{\mathsf{in}}$ and $\mathsf{SB}_3^{\mathsf{out}}$ are halved for each additional MixBytes column and are shown in Table 2.

MB_1 and MB_4. Until now, we have determined differences for the states $\mathsf{SB}_2^{\mathsf{out}}$, $\mathsf{SB}_3^{\mathsf{in}}$, $\mathsf{SB}_3^{\mathsf{out}}$ and $\mathsf{SB}_4^{\mathsf{in}}$. Since all differences in $\mathsf{SB}_2^{\mathsf{out}}$ and $\mathsf{SB}_4^{\mathsf{in}}$ have already been determined, we have only about $255/2^8 \sim 1$ difference left for $\mathsf{SB}_2^{\mathsf{in}}$ and $\mathsf{SB}_4^{\mathsf{out}}$. Note that choosing the difference for one byte determines all other differences as well due to the restrictions by MixBytes.

Note that we can find one possible differential characteristic with a complexity of about one, since we filter though each MixBytes and S-box transformation only once. The total number of possible differential paths can be determined by considering the number of choices we have at the input and output of S-box SB_3, the input of S-box SB_2 and the output SB_4. The approximate number of choices are listed in Table 2 and by multiplying these numbers we can get up to $\sim 2^{64}$ possible differential paths or starting points for the next phase.

Table 2. The approximate number of possible choices for the differences at the input and output of the 3 S-boxes SB_2, SB_3 and SB_4.

$\mathsf{SB}_2^{\mathsf{in}}$	$\mathsf{SB}_3^{\mathsf{in}}$	$\mathsf{SB}_3^{\mathsf{out}}$	$\mathsf{SB}_4^{\mathsf{out}}$
1	255	127	1
	127	64	
	64	32	
	32	16	
	16	8	
	8	4	
	4	2	
	2	1	

4.2 Solving for Conforming State Pairs

After we have found a differential path we need to search for a valid pair of the state. Since the differential of each active S-box is fixed there are only either 2 or 4 input pairs possible. In both cases, the S-box behaves linearly [10] and hence, we can easily solve the resulting linear system of equations. In the following description we assume that we have only 2 possible input pairs for each active S-box (note that in this case, all S-boxes behave linearly).

Consider the diagonal of $\mathsf{SB}_3^{\mathsf{out}}$ respectively the first column of $\mathsf{MB}_3^{\mathsf{in}}$ (denoted by "2" in Fig. 2). For each S-box we have 2 possible inputs k_i and k'_i for $0 \leq i < 8$ such that the differential path holds. In other words, we have 2^8 possible inputs for the diagonal of $\mathsf{SB}_3^{\mathsf{out}}$. Let $x \in \{0,1\}^8$ then the possible values for the diagonal of $\mathsf{SB}_3^{\mathsf{out}}$ are given by:

$$k \oplus x \cdot (k \oplus k')$$

where $k = [k_0, \ldots, k_7]$ and $k' = [k'_0, \ldots, k'_7]$.

Next, we compute the first byte of $\mathsf{SB}_4^{\mathsf{in}}$ by going forward ShiftBytes, MixBytes and AddRoundConstant.

$$\mathsf{SB}_4^{\mathsf{in}}[0,0] = (\mathsf{k} \oplus \mathsf{x} \cdot (\mathsf{k} \oplus \mathsf{k}')) \cdot \mathsf{L}$$

where L denotes composition of ShiftBytes, MixBytes and AddRoundConstant. Since these transformations are all linear L is a linear transformation as well.

Since we have 2 possible values a and a' for $\mathsf{SB}_4^{\mathsf{in}}[0,0]$ such that the differential trail holds, the following equation has to be fulfilled.

$$(k \oplus x \cdot (k \oplus k')) \cdot L = a \oplus y \cdot (a \oplus a')$$

where $y \in \{0,1\}$.

By doing the same for the other diagonals (corresponding to columns 2-8 of $\mathsf{MB}_3^{\mathsf{in}}$) we get a system of 64 equations in 64+8=72 variables which has to be fulfilled to guarantee that the differential trail holds in the forward direction. In a similar way we also get a system of 64 linear equations in 72 variables by going backward from $\mathsf{SB}_3^{\mathsf{in}}$ to $\mathsf{SB}_2^{\mathsf{out}}$. However, since the values of $\mathsf{SB}_3^{\mathsf{in}}$ and $\mathsf{SB}_3^{\mathsf{out}}$ are related, we get in total a system of 128 equations in 80 variables when we combine them. In other words, to find a valid pair, we have to backtrack and try about 2^{48} differential paths and thus, solve the linear system of equations 2^{48} times. Since we can start with up to 2^{64} differential paths, we can only find about $2^{64-48} = 2^{16}$ pairs after the linear solving step.

Note that the attack works similar if one has 4 possible input pairs for the S-box. By choosing the differences in the previous step (Section 4.1) in a way, to maximize the number of differentials with 4 possible pairs for the S-box, the overall complexity can be reduced slightly (by about 2^2 to 2^5). The total complexity of the attack is given by the number of times we need to solve the resulting linear system of equations (we assume here that this corresponds to about one compression function call).

4.3 Application to AES

The same technique applies to the block cipher AES as well. In this case, we start with a differential path with the following sequence of active S-boxes:

$$1 \rightarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1$$

Hence, we get 64 conditions (equations) for the S-box layers of round 2, 3 and 4. Since we have 64 equations in 24 variables, we need to repeat the attack $2^{64-24} = 2^{40}$ times to find a valid pair. Note that in the case of AES, we get a better complexity if we first fix the differential path for rounds 1-3 ($1 \rightarrow 4 \rightarrow 16 \rightarrow 4$) and then, solve for the pair. In this case, we get only 32 conditions and the complexity to solve for a pair is about 2^{12}. Since we need to repeat the attack only 2^{24} times to fulfill the last MixColumns operation we get a total complexity of only 2^{36} in this case.

5 A Start-From-the-Middle Technique

In this section, we describe another attack that uses the available freedom degrees in the middle. The truncated differential path considered here will be the same than in the previous section or in the rebound attack [22]: in the case of `Grøstl`, we use the one from Fig. 1. More precisely, the attack will first focus on a 3-round part of the middle of the path, the following sequence of active bytes:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8$$

We can find a conforming state pair according to this path with only a few operations by choosing "good" differences in advance and exploiting the available degrees of freedom. We start at the last MixBytes transformation of the 3-round trail (MB_3 in Fig. 2) and compute backwards. The attack can be divided into three main phases:

1. In Phase 1, we start with 1-byte differences at the output of each MixBytes column MB_3 ($\mathsf{MB}_3^{\mathsf{out}}$) and compute backwards to the input of SB_3 ($\mathsf{SB}_3^{\mathsf{in}}$). Each column of MixBytes MB_3 can be computed independently. Then, we maintain as much freedom as possible in the input difference of SB_3 ($\mathsf{SB}_3^{\mathsf{in}}$) by using the precomputed differential tables of the S-box.
2. In Phase 2, we have enough degrees of freedom to choose the differences for $\mathsf{SB}_3^{\mathsf{in}}$ such that each of the eight MB_2 MixBytes transitions from 8 to 1 active byte in backward direction is satisfied.
3. In Phase 3, we get more degrees of freedom since both (a, b) and (b, a) are valid solutions for each byte of $\mathsf{SB}_2^{\mathsf{in}}$. Hence, we can randomize each active byte of $\mathsf{SB}_2^{\mathsf{in}}$ and get enough pairs such that the last single MixBytes transformation MB_1 can be fulfilled as well.

At this point, all available degrees of freedom have been used and we rely on a probabilistic behavior for the remaining transitions in backward and forward direction.

5.1 Application to Grøstl-256

Phase 1. We randomly select non-zero differences for the eight active bytes of $\mathsf{SB}_4^{\mathsf{in}}$, i.e. for $\mathsf{SB}_4^{\mathsf{in}}[\mathsf{i},\mathsf{i}]$ with i ranging from 0 to 7. Those differences will remain unchanged when computing backward to $\mathsf{MB}_3^{\mathsf{out}}$. Since the MixBytes transformation is linear, we apply its inverse (Phase 1.A in Fig. 3) to $\mathsf{MB}_3^{\mathsf{out}}$ and deterministically get 64 byte differences for $\mathsf{MB}_3^{\mathsf{in}}$ and thus, for $\mathsf{SB}_3^{\mathsf{out}}$. We denote by $\delta[i,j]$ the byte difference of $\mathsf{SB}_3^{\mathsf{out}}[\mathsf{i},\mathsf{j}]$. For each output difference $\delta[i,j]$ in SB_3, we compute all valid byte pairs $\mathsf{SB}_3^{\mathsf{in}}[\mathsf{i},\mathsf{j}]$ such that the S-box differential holds (Phase 1.B in Fig. 3). As discussed in Section 3.2 we can choose from 127 possible input differences for $\mathsf{SB}_3^{\mathsf{in}}[\mathsf{i},\mathsf{j}]$ using the S-box differential table. For each of these XOR difference, we get two possible pairs (a,b) and (b,a). Hence, for each byte of $\mathsf{SB}_3^{\mathsf{in}}$, we get a list (denoted by capital letters in Fig. 3) of 254 valid candidate pairs which are sorted by input difference and stored in table $S_F^{\delta[i,j]}$. Note that *any* choice of these pairs will conform to the expected differential path from $\mathsf{SB}_3^{\mathsf{in}}$ up to $\mathsf{SB}_4^{\mathsf{in}}$.

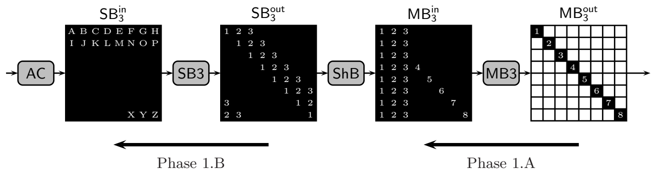

Fig. 3. Phase 1 of the attack.

Phase 2. We now take care of the differential path from $\mathsf{SB}_3^{\mathsf{in}}$ to $\mathsf{SB}_2^{\mathsf{in}}$. Since we can choose a candidate pair for each byte $\mathsf{SB}_3^{\mathsf{in}}$ independently, we will process independently for each column of MB_2 as well. More precisely, for each column j of $\mathsf{SB}_3^{\mathsf{in}}$ (or $\mathsf{MB}_2^{\mathsf{out}}$), we will use the inverse MixBytes table M_B^j to choose each byte difference of $\mathsf{SB}_3^{\mathsf{in}}$, such that they result in only one active byte at the input of MB_2 ($\mathsf{MB}_2^{\mathsf{in}}$). For each of the 255 differences of M_B^j, we check if some candidate pairs of $\mathsf{SB}_3^{\mathsf{in}}$ (computed during Phase 1 and stored in $S_F^{\delta[i,j]}$) can fit the 8-byte difference of $\mathsf{MB}_2^{\mathsf{out}}$ (see Fig. 4). Since for each byte of $\mathsf{SB}_3^{\mathsf{in}}$ we can choose from 127 possible output differences of the S-box, the probability of success is $127/255 \simeq 1/2$.

Thus, for an entire column of MB_2 we get a probability of $(127/255)^8 \simeq 2^{-8}$ such that one valid candidate pair can be found. Since we can start with 255 input differences for each column of MB_2, we can find one solution for a column with probability $1-(1-(127/255)^8)^{255} \simeq 0,62$. We continue for all eight columns of $\mathsf{SB}_3^{\mathsf{in}}$. The probability of success is about $(0,62)^8 \simeq 2^{-5,5}$ and we have to restart at Phase 1 about $2^{5.5} = 46$ times to find a solution. At the end of Phase 2, we

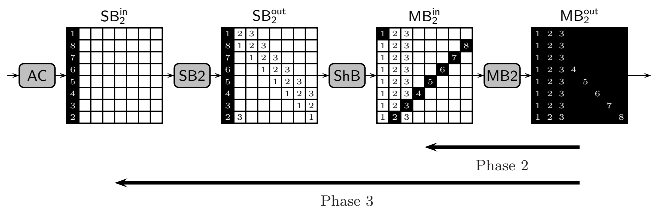

Fig. 4. Phase 2 and Phase 3 of the attack.

get a set of byte pairs for $\mathsf{SB}_3^{\mathsf{in}}$, which conforms to the differential path from $\mathsf{SB}_2^{\mathsf{in}}$ up to $\mathsf{SB}_4^{\mathsf{in}}$.

Note that these two first phases are doing essentially the same work as the rebound attack [22], but need fewer operations to complete (on average the rebound attack takes about one operations per valid candidate, but this whole step required 2^{64} operations). Here, we need to repeat the process $2^{5.5}$ times to find a solution, but compute only a few table lookups per iteration. Thus, we consider that we can find one solution for the truncated differential path $1 \rightarrow 8 \rightarrow 64 \rightarrow 8$ with about one computation of `Grøstl`-256 on average.

Phase 3. It seems that at this phase, the differences in $\mathsf{SB}_2^{\mathsf{in}}$ and $\mathsf{SB}_4^{\mathsf{out}}$ can not be chosen anymore. However, an observation allows us to actually get some control over the differences in $\mathsf{SB}_2^{\mathsf{in}}$. We denote by S a 64-byte solution of $\mathsf{SB}_3^{\mathsf{in}}$ (found at the end of Phase 2) and by $(a,b)^{[i,j]}$ the byte pair of row i and column j in S. In Fig. 4, we can see that the active bytes of $\mathsf{SB}_2^{\mathsf{in}}$ are located in the first column. By looking at this figure, it is easy to check that the differences of the active bytes located at row j of $\mathsf{SB}_2^{\mathsf{in}}$ depend only on the byte pairs of the j-th column of $\mathsf{MB}_2^{\mathsf{out}}$ (or $\mathsf{SB}_3^{\mathsf{in}}$). We know that $(a,b)^{[0,j]}, ..., (a,b)^{[7,j]}$ are valid solutions for this column, and switching a and b in any of the pairs actually maintains the validity of those candidates (the differences values of each byte will remain the same in MB_2 and MB_3).

Thus, one solution for each column of $\mathsf{SB}_3^{\mathsf{in}}$ provides us in fact 2^8 valid candidates[4]. Each of these solutions will lead to a random difference on the corresponding active byte $\mathsf{SB}_2^{\mathsf{in}}[\mathsf{j},0]$, independently of all other differences in SB_2. Now, if we can hit any of the elements of M_B^0 for MB_1 from the newly available differences in $\mathsf{SB}_2^{\mathsf{in}}$, we directly get a solution for the differential path from $\mathsf{SB}_1^{\mathsf{out}}$ to $\mathsf{SB}_4^{\mathsf{out}}$. Since we have 255 elements in M_B^0, we expect about 2^7 solutions on average (2^8 solutions, but half of them may be repeating, see footnote 4).

[4] We have 2^8 different combinations by switching a and b for each column. However, we must take in account that some repeating combinations are counted here (given a combination, inverting everything will obviously lead to exactly the same behavior in the differential path). Thus, instead of having 64 bits degrees of freedom left (8 for each column) we intrinsically loose one of them and get 63 degrees of freedom.

We did not succeed to control the differences in $\mathsf{SB}_4^{\mathsf{out}}$ as well. Thus, if the differences are uniformly distributed, the success probability for the 8 to 1 active byte transition from the MixBytes layer MB_4 is equal to $2^{-8\times 7} = 2^{-56}$.

5.2 Application to AES

Again, also this technique can be applied to the AES block cipher. We use the same differential path as in Fig. 1, except that we manipulate a 4×4 state and that no MixColumns transformation is applied in the last round:

$$4 \to 1 \to 4 \to 16 \to 4 \to 1 \to 4 \to 4$$

Phase 1. This step is analog to the `Grøstl`-256 case.

Phase 2. This step is similar to the case of `Grøstl`-256. However, the probability computation changes when looking for a match between $\mathsf{MB}_2^{\mathsf{out}}$ and $\mathsf{SB}_3^{\mathsf{in}}$. For each column, we now get a probability of $(127/255)^4 \simeq 2^{-4}$ such that at least one valid candidate pair can be found. Since we have 255 differences in M_B^i, we will immediately find one solution for each starting difference $\mathsf{SB}_4^{\mathsf{in}}$ of the attack. In fact, we expect up to about 2^4 solutions for each column.

Phase 3. Again, we try to control the differences in $\mathsf{SB}_2^{\mathsf{in}}$. We use the same technique as for `Grøstl`-256: for each active byte at row i in $\mathsf{SB}_2^{\mathsf{in}}$, we can randomize its difference by randomizing the solutions on the column i in $\mathsf{SB}_3^{\mathsf{in}}$. By switching a and b, we directly get 2^4 solutions per column. Moreover, we also have to consider the fact that for the AES case, we already had 2^4 solutions per column. Thus, we get in total about 2^8 solutions per column (see footnote 4). Each of those solutions will lead to a random difference on the corresponding active byte of $\mathsf{SB}_2^{\mathsf{in}}$, independently of the other active bytes of $\mathsf{SB}_2^{\mathsf{in}}$. Now, if we can hit any of the elements of M_B^0 using the available differences in $\mathsf{SB}_2^{\mathsf{in}}$, we get a solution for the differential path between $\mathsf{SB}_1^{\mathsf{out}}$ and $\mathsf{SB}_4^{\mathsf{out}}$. Since we have 255 elements in M_B^0, the whole attack will find about 2^7 solutions on average (2^8 solutions, but half of them may be fully repeating ones, see footnote 4).

Extending the Path. Propagating from $\mathsf{SB}_4^{\mathsf{in}}$ to $\mathsf{SB}_5^{\mathsf{in}}$ according to the truncated differential path has a success probability of $2^{-3 \cdot 8} = 2^{-24}$. Thus, we can find a pair corresponding to the path from $\mathsf{SB}_1^{\mathsf{in}}$ to $\mathsf{SB}_5^{\mathsf{in}}$ with about 2^{24} round computations on average.

6 Results

In the previous two sections, we have proposed two new techniques to find differences and values for a 4-round truncated differential path with $1 \to 8 \to 64 \to 8 \to 1$ active bytes for `Grøstl`-256. In the following, we apply these results to the permutation, compression function and output transformation of `Grøstl`-256, the AES block cipher and the ECHO permutation.

6.1 `Grøstl`-256

Both proposed techniques can be used to improve the complexity of the 6-round semi-free-start collision attack of [22]. However, due to the limited degrees of freedom, a semi-free-start collision attack on 7 rounds of the `Grøstl`-256 compression function is not possible.

6 Rounds. Both proposed techniques (described in Sect. 4 and Sect. 5) can be used to find a valid pair for the 6 round trail of P and Q, given in [22]:

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 8 \xrightarrow{r_6} 64$$

Using the linearized match-in-the-middle attack, we can omit the conditions on SB_4. Hence, the number of equations is reduced to 64 and we expect to find a solution (in fact 2^8 solutions) for already the first differential path. The complexity to find a match for the 8 active bytes (64 bits) at the input, and at the output prior to the (linear) MixBytes transformation is 2^{32} each. Hence, the total complexity to find a semi-free-start collision for 6 rounds of `Grøstl`-256 is about 2^{64} in time and memory.

Using the start-from-the-middle technique, we can construct a differential path with active bytes $8 \rightarrow 1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 8 \rightarrow 64$) with only a few operations. As a proof of concept, we give in Appendix A a valid input pair for the permutations P and Q on 6-rounds of `Grøstl`-256 which conforms to this truncated differential path. We get a final complexity of 2^{64} operations and memory for a semi-free-start collision on `Grøstl`-256 reduced to 6 rounds.

7 Rounds. Again, both techniques can be used to find a valid pair conforming to the 4-round part in the middle of Fig. 1 ($1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$) with a relatively low complexity (2^{48} and 2^{56}). This path can be extended by one round in backward and two rounds in forward direction to give a differential path of the form:

$$8 \rightarrow 1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1 \rightarrow 8 \rightarrow 64,$$

However, using both techniques we can only find 2^{16} pairs conforming to this truncated differential path and one can convince himself with a counting argument: In the middle of the differential path where all bytes of the state are active, one can start with approximatively $2^{512} \cdot 2^{512} = 2^{1024}$ different pairs. However, only a portion 2^{-56} will follow a MixBytes transition $8 \rightarrow 1$, and only a portion $2^{-56 \cdot 8} = 2^{-448}$ will follow a MixBytes transition $64 \rightarrow 8$ (because we have a probability of 2^{-56} for each column). Since we have two $64 \rightarrow 8$ and two $8 \rightarrow 1$ transitions and consider them to be independent, only $2^{1024-448 \cdot 2-56 \cdot 2} = 2^{16}$ valid pairs will remain for the 4-round path in the middle ($1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$) and thus, also for the 7-round path.

Note that due to this lack of freedom a semi-free-start collision using this truncated differential path is not possible. For a collision at the end of 7 rounds, we need about 2^{64} pairs for each, P and Q. Otherwise, a birthday attack on 128 bits (8 active bytes at the input, 8 active bytes prior to MixBytes at the output) is not feasible. By using different positions of active bytes in round 2 and 6, but the same column for P and Q, we can construct about $2^6 \cdot 2^6 \cdot 2^3 \cdot 2^3 = 2^{18}$ different truncated paths. By far not enough for a collision attack. However, one could think of a free-start near-collision attack on 7 rounds of Grøstl-256 but this property gets destroyed by the output transformation.

Therefore, we can only get a distinguisher for the permutation or output transformation of Grøstl-256 reduced to 6.5 rounds (without the final MixBytes transformation). The complexity is 2^{48} instead of 2^{224} for a random 512-bit permutation or 2^{112} for a random 256-bit function. We can get a distinguisher for the full 7 rounds by applying the subspace distinguisher proposed in [19]. Note that the input and the output differences of the 6.5 round attack form a vector space of dimension 64 at the input and output. Since the Mixbytes transformation is linear also the output differences after 7 rounds form a vector space of dimension 64. Hence, we can apply the subspace distinguisher with parameters $N = 512$, $n = 64$, $t = 128$ (generic complexity: $2^{115.4}$) to distinguish 7 rounds of the permutation P and Q. To construct a vector space of size $t = 128$, we need to repeat our attack on the comression function 2^7 times. Hence, the total complexity for the subspace distinguisher of the permutation is about 2^{55} permutation calls with negligible memory.

Similarily, we can use the subspace distinguisher to distinguish the output transformation of Grøstl-256 as well. Note that the 8 active bytes of the input are added to the output by the feed-forward. However, due to the truncation at the end the output differences will still form a vector space of dimension 64. Since Grøstl-256 truncates columns and MixBytes works on columns, we keep only half of the vector space. Hence, we can apply a subspace distinguisher with parameters $N = 256$, $n = 64$, $t = 256$ (generic complexity: $2^{75.9}$) and need to repeat our attack 2^8 times to get a vector space of size $t = 256$. Hence, the total complexity for the subspace distinguisher on 7 rounds of the Grøstl-256 output transformation is about 2^{56} output transformation calls and negligible memory.

6.2 AES Block Cipher

Both proposed techniques apply to the block cipher AES in the known-key distinguisher setting as well. The resulting 7-round differential path for AES is computed by simply extending the 4-round path in both forward and backward direction to give the following sequence of active bytes:

$$4 \rightarrow 1 \rightarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1 \rightarrow 4 \rightarrow 4$$

Note that the last MixColumns operation is omitted in the AES. Since we aim for 4 active bytes in both, plaintext and ciphertext, we would expect to find such a pair with about 2^{48} computations for a random permutation. Note that an equivalent generic attack needs to find a pair with only 4 active bytes at the input and output as well. Hence, the best generic method is to start with 4 active bytes at the input and search for a near-collision on 12 non-active bytes at the output with complexity $2^{(12\cdot 8)/2} = 2^{48}$.

Using the linearized match-in-the-middle attack, we get a known-key distinguisher for 7-rounds of AES with a complexity of about 2^{36} and negligible memory. However, the start-from-the-middle technique allows us to further improve the complexity for the known-key distinguisher to about 2^{24} in time and negligible memory for 7-rounds of AES. Additionally, one may think of other applications of these attack such as near-collisions on a compression function built upon the 7-round reduced AES in Davies-Meyer mode [6,23], or a collision attack on the compression function for 5 rounds.

6.3 Internal Permutation of `ECHO`

It is possible to apply the start-from-the-middle technique to other AES-based hash functions, such as `ECHO` [2] whose internal 2048-bit permutation can been seen as a big AES processing 128-bit words instead of bytes. This directly gives us an improved distinguisher on 7 rounds whose complexity is $2^{3\cdot 128} = 2^{384}$ operations (compared to the previous one with complexity 2^{896}) and memory requirements are 2^{256}. However, we can improve the memory requirements by storing a differential lookup table for the AES super box [9] with size $2^{32} \cdot 2^{32} = 2^{64}$, instead of a differential lookup table for two full AES rounds with size $2^{128} \cdot 2^{128} = 2^{256}$. This is possible due to the fact that one can combine the last MixColumns transformation of the AES with the subsequent BigMixColumns transformation of `ECHO`, since both transformations are linear. Note that this attack only allows to distinguish 7 rounds of the `ECHO` internal permutation from a random 2048-bit permutation, but does not apply to the compression function due to the word compression at its output.

7 Conclusion and Future Work

In this paper, we have proposed two new ways to mount attacks on the SHA-3 candidates `Grøstl` and `ECHO`, and the block cipher AES. Our results improve upon and extend the rebound attack. Both techniques are an extension of the number of rounds in which degrees of freedom can be used to improve from two to four rounds. As a result, we present the best known attacks on constructions where (reduced variants of) *permutations* are used. We improve on the attack on the reduced `Grøstl`-256 compression function (up to 6 out of 10 rounds), and present a distinguisher for 7-rounds of the `Grøstl`-256 permutation and output transformation. Further, we improve upon the distinguisher for 7-rounds of the internal permutation of `ECHO` and the known-key distinguisher for 7-rounds of the block cipher AES. Nevertheless, a comfortable security margin for these SHA-3 candidates remain. Not only because both proposals have a higher number of rounds, but also because in an attack on the hash function much less degrees of freedom are available (compared to an attack on the compression function or permutation).

On the other hand, the new techniques of this paper have been optimized for this setting and do not directly apply to other settings where more degrees of freedom are available. Sources for such degrees of freedom are salt, counter, or key inputs. While the analysis typically gets more complicated if more freedom is available, much better attacks can be expected. As an example we refer to a recent extension of the rebound attack on the full 10-round Whirlpool compression function [19]. Note that Whirlpool is a block cipher based construction which offers additional degrees of freedom through its conservative key schedule. Some SHA-3 candidates use block-cipher based compression functions with key-schedules less conservative than Whirlpool. Hence, more degrees of freedom are available to an attacker and better results may be expected along those lines.

In general, the rebound attack and its extensions as described in this paper, work with any differential or truncated differential. However, the diffusion properties of AES based hash functions allow a very simple construction of good truncated differential paths, which facilitates the analysis. Nevertheless, future work will include the application of the rebound idea on other hash constructions, even though this may require sophisticated tools to obtain good results, as was the case for *e. g.* SHA-1 [11].

Acknowledgments

We would like to thank Joan Daemen for the idea on the linearized match-in-the-middle attack, and Henri Gilbert, Mario Lamberger, Vincent Rijmen and the anonymous referees for useful comments and discussions. The work in this paper has been supported in part by the European Commission under contract ICT-2007-216646 (ECRYPT II) and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. Paulo S. L. M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function. Submitted to NESSIE, September 2000. Revised May 2003. Available online at `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html` (2008/12/11).
2. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST, 2008. Available online at `http://crypto.rd.francetelecom.com/echo/`.
3. Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org`.
4. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
5. Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *CRYPTO*, LNCS. Springer, 2009. To appear.
6. Bram Cohen and Ben Laurie. AES-hash, 2001. Available online at `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/aes-hash/aeshash.pdf`.

7. Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001.
8. Joan Daemen and Vincent Rijmen. *The Design of Rijndael.* Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.
9. Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *LNCS*, pages 78–94. Springer, 2006.
10. Joan Daemen and Vincent Rijmen. Plateau characteristics. *IET Information Security*, 1(1):11–17, March 2007.
11. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
12. Ewan Fleischmann, Christian Forler, and Michael Gorski. The Twister Hash Function Family. Submission to NIST, 2008.
13. Pierre-Alain Fouque, Jacques Stern, and Sébastien Zimmer. Cryptanalysis of Tweaked Versions of SMASH and Reparation. In *SAC*, LNCS. Springer, 2008. To appear.
14. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST, 2008. Available online at `http://www.groestl.info`.
15. Dmitry Khovratovich. Cryptanalysis of hash functions with structures. In Michael J. Jacobson, Vincent Rijmen, and Rei Safavi-Naini, editors, *SAC*, LNCS. Springer, 2009. To appear.
16. Dmitry Khovratovich, Alex Biryukov, and Ivica Nikolic. The Hash Function Cheetah: Specification and Supporting Documentation. Submission to NIST, 2008.
17. Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Alex Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 39–57. Springer, 2007.
18. Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.
19. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, LNCS. Springer, 2009. To appear.
20. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolić, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full LANE Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, LNCS. Springer, 2009. To appear.
21. Florian Mendel, Christian Rechberger, and Martin Schläffer. Cryptanalysis of Twister. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *LNCS*, pages 342–353, 2009.
22. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren Steffen Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.
23. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
24. Marine Minier, Raphael C.-W. Phan, and Benjamin Pousse. Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks. In Bart

Preneel, editor, *AFRICACRYPT*, volume 5580 of *LNCS*, pages 60–76. Springer, 2009.

25. 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.
26. 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. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17).
27. Thomas Peyrin. Cryptanalysis of Grindahl. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 551–567. Springer, 2007.
28. Shuang Wu, Dengguo Feng, and Wenling Wu. Cryptanalysis of the LANE Hash Function. In Michael J. Jacobson, Vincent Rijmen, and Rei Safavi-Naini, editors, *SAC*, LNCS. Springer, 2009. To appear.

A Message and Chaining Variable Example for the 6-Round Differential Path of Grøstl-256

We give here in hexadecimal display a chaining variable and message pair example ($[H_1, M_1]$,$[H_2, M_2]$) that verifies the 6-round differential path for Grøstl-256.

```
H1 = fdab6faf65da3531e5a7f611baba937d
     b18648152738a5fe4bd38ca5a8b050e7
     3d734623aed6f7a35e3fb3d72eba5e60
     1712a3d23d76fe79ccbba10461dddee0

M1 = 66b16a712984a23ca99283090e5818c7
     c7f46fcd74c54b7a9950a4bfcb2861b1
     1f90846a04c92172af57a58ad9b747a3
     a26dca926c18f410ad0f40f52800d27b


H2 = 21ab6faf65da3531e51bf611baba937d
     b186c5152738a5fe4bd38c88a8b050e7
     3d734623ecd6f7a35e3fb3d72e6c5e60
     1712a3d23d767779ccbba10461ddde66

M2 = f8b16a712984a23ca9ef83090e5818c7
     c7f434cd74c54b7a9950a40fcb2861b1
     1f90846a29c92172af57a58ad95547a3
     a26dca926c18d710ad0f40f52800d27f
```