Rebound Attacks on the Reduced Grøstl Hash Function[⋆]

Florian Mendel[1], Christian Rechberger[2], Martin Schläffer[1], and Søren S. Thomsen[3]

[1] Institute for Applied Information Processing and Communications (IAIK) Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
[2] Dept. of Electrical Engineering ESAT/COSIC, K.U.Leuven, and Interdisciplinary Institute for BroadBand Technology (IBBT), Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium.
[3] Department of Mathematics, Technical University of Denmark Matematiktorvet 303S, DK-2800 Kgs. Lyngby, Denmark
`martin.schlaeffer@iaik.tugraz.at`

Abstract. Grøstl is one of 14 second round candidates of the NIST SHA-3 competition. Cryptanalytic results on the wide-pipe compression function of Grøstl-256 have already been published. However, little is known about the hash function, arguably a much more interesting cryptanalytic setting. Also, Grøstl-512 has not been analyzed yet. In this paper, we show the first cryptanalytic attacks on reduced-round versions of the Grøstl hash functions. These results are obtained by several extensions of the rebound attack. We present a collision attack on 4/10 rounds of the Grøstl-256 hash function and 5/14 rounds of the Grøstl-512 hash functions. Additionally, we give the best collision attack for reduced-round (7/10 and 7/14) versions of the compression function of Grøstl-256 and Grøstl-512.

Keywords: hash function, cryptanalysis, collisions, rebound attack

1 Introduction

In the last few years the cryptanalysis of hash functions has become an important topic within the cryptographic community. The attacks on the MD4 family of hash functions (e.g., MD5 [12, 15], SHA-1 [2, 14]) have especially weakened the confidence in the security of this design strategy. Many new and interesting hash function designs have been proposed as part of the NIST SHA-3 competition [11]. Most submissions are constructed using specific underlying building

⋆ This work was supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II and the fourth author is supported by a grant from the Villum Kann Rasmussen Foundation. Parts of this work were carried out while the third author was visiting Technical University of Denmark, supported by a grant from DCAMM International Graduate Research School, Danish Center for Applied Mathematics and Mechanics.

blocks like permutations, explicit compression functions, or block ciphers. Sometimes, proofs are devised to show that some desirable properties of the hash function (like collision resistance) can be reduced to a property of an underlying building block.

In turn, many cryptanalytic results have been published which consider these building blocks. Often, the resulting attacks are not applicable to the hash function itself. While these results are important to analyze the security of a specific design, it is very difficult to compare the results of different hash function proposals. How can we measure and compare the security margin of different designs? In addition to the (reduced) security parameter that is used for the best attack, a number of other issues heavily influence the answer: Is the design wide-pipe, is it based on the sponge model, or does it use an MD-style iteration? Is the hash function based on an ideal block cipher or a random permutation? All these considerations can be bypassed if we compare cryptanalytic results of the complete hash function instead of different underlying building blocks. Thus, a comparison of different designs is made easier.

In this paper we analyze the *hash* function `Grøstl` [4], which is one of the remaining 2nd-round candidates of the NIST SHA-3 competition. `Grøstl` has very competitive hardware implementation characteristics (see e.g., Tillich et al. [13] for a comparison), is the fastest among the remaining AES-like designs on most platforms, and naturally deserves cryptanalytic attention.

`Grøstl` is based on a wide-pipe compression function that is iterated in an MD-style manner. Since the wide-pipe *compression* function of `Grøstl` is known to be non-random, many distinguishers exist and the hash function has been designed with this fact in mind. With ℓ denoting the output size of the compression function, even collision attacks in $2^{\ell/3}$ time or $2^{\ell/4}$ permutation queries, memoryless preimage attacks in time $2^{\ell/2}$, and very efficient distinguishers (only two calls) are known [4]. Hence a strong output transformation with truncation is an important part of the design.

Shortcut collision attacks on round-reduced versions of the compression function of `Grøstl`-256 have been presented in a series of papers [5,8,9]. As discussed above, additional distinguishers on the compression function are meaningless. However, showing non-random properties of the underlying permutations or the output transformation can have some significance. See e.g., Mendel et al. [8] for results along those lines, where among others, a distinguisher for 7 rounds of the output transformation with complexity 2^{56} is given.

However, little is known about the hash function, which is arguably a more interesting cryptanalytic setting. Only half of the degrees of freedom are available to an attacker for direct manipulation compared to a compression function attack. Also, `Grøstl`-512 has not been considered yet. In this paper, we first improve the rebound attack as originally applied to the `Grøstl`-256 compression function [9]. Using the rebound attack, we give results for the `Grøstl`-512 compression function and present the first analysis of the reduced `Grøstl` hash functions. Our results and the best previously known results are summarized in Table 1.

Table 1: Summary of rebound analysis for the round-reduced `Grøstl` hash and compression functions.

Target	Hash Size	Rounds	Time	Memory	Type	Reference
hash	224,256	4/10	2^{64}	2^{64}	collision	Sect. 5.1
function	384,512	5/14	2^{176}	2^{64}	collision	Sect. 5.2
	256	6/10	2^{120}	2^{64}	semi-free-start collision	[9]
compression	224,256	6/10	2^{64}	2^{64}	semi-free-start collision	[8]
function	256	7/10	2^{120}	2^{64}	semi-free-start collision	Sect. 5.3, [5]
	384,512	7/14	2^{152}	2^{64}	semi-free-start collision	Sect. 5.4

We start the paper by recalling the relevant parts of the `Grøstl` specification in Section 2 and give the basics of the rebound attack in Section 3. The new ideas and improvements which are the basis for our results are presented in Section 4. The results for the hash function and compression function for both `Grøstl`-256 and `Grøstl`-512 are given in Section 5. Finally, we conclude in Section 6.

2 Description of `Grøstl`

The hash function `Grøstl` was designed by Gauravaram et al. as a candidate for the SHA-3 competition [4]. It is an iterated hash function with a compression function built from two distinct permutations P and Q, which are based on the same principles as the AES round transformation [10]. `Grøstl` is a wide pipe design with security proofs for the collision and preimage resistance of the compression function [3]. In the following, we describe the `Grøstl` hash function and the permutations of `Grøstl`-256 and `Grøstl`-512 in more detail.

2.1 The `Grøstl` Hash Function

The input message M is padded and split into blocks $M_1, M_2, \ldots, M_t$ of ℓ bits with $\ell = 512$ for `Grøstl`-256 and $\ell = 1024$ for `Grøstl`-512. The initial value H_0, the intermediate hash values H_i, and the permutations P and Q are of size ℓ as well. The message blocks are processed via the compression function f, which accepts two inputs of size ℓ bits and outputs an ℓ-bit value. The compression function f is defined via the permutations P and Q as follows:

$$f(H, M) = P(H \oplus M) \oplus Q(M) \oplus H.$$

The compression function is iterated in the usual way with $H_0 = IV$ and $H_i \leftarrow f(H_{i-1}, M_i)$ for $1 \leq i \leq t$. The output H_t of the last call of the compression function is processed by an output transformation g defined as $g(x) = \mathrm{trunc}_n(P(x) \oplus x)$, where n is the output size of the hash function and $\mathrm{trunc}_n(x)$ discards all but the least significant n bits of x. Hence, the digest of the message M is defined as $h(M) = g(H_t)$.

2.2 The Grøstl-256 Permutations

As mentioned above, two permutations P and Q are defined for Grøstl-256. The permutations differ only in the used constants. Both permutations operate on a 512-bit state, which can be viewed as an 8×8 matrix of bytes. Each permutation of Grøstl-256 consists of 10 rounds, where the following four AES-like [10] round transformations are applied to the state in the given order:

- AddRoundConstant (AC) XORs a constant to one byte of the state. The constant changes in every round and is different for P and Q.
- SubBytes (SB) applies the AES S-box to each byte of the state.
- ShiftBytes (SH) cyclically rotates the bytes of row i to the left by i positions.
- MixBytes (MB) is a linear diffusion layer, which multiplies each column with a constant 8×8 circulant MDS matrix.

For details on the round transformations we refer to the Grøstl specification [4]. Note that AddRoundConstant is the only transformation that distinguishes P from Q. The properties of the round transformations which are used in the following attacks are similar to those of the AES (see Section 3 for more details).

2.3 The Grøstl-512 Permutations

The permutations used in Grøstl-512 are of size $\ell = 1024$ bits and the state is viewed as an 8×16 matrix of bytes. The permutations use the same round transformations as in Grøstl-256 except for ShiftBytes: Since the permutations are larger, row j is cyclically shifted j positions to the left for $0 \leq j \leq 6$ and row 7 is shifted 11 positions to the left. The number of rounds is increased to 14.

3 The Rebound Attack on Grøstl

The rebound attack was published by Mendel et al. in [9] and is a new tool for the cryptanalysis of hash functions. It can be applied to both block cipher based and permutation based constructions. The idea of the rebound attack is to divide an attack into two phases, an inbound and outbound phase. The inbound phase is an efficient meet-in-the-middle phase, which exploits the available degrees of freedom in the middle of a (truncated) differential path to guarantee that the expensive part of a differential path holds. In the (mainly) probabilistic outbound phase the solutions of the inbound phase are computed backwards and forwards to obtain an attack on the hash or compression function. In the following, we explain the rebound attack using the 6 round semi-free-start collision attack on Grøstl-256. For a more detailed description, we refer to the original paper [9].

3.1 The Truncated Differential Path

The rebound attack on 6 rounds of the Grøstl-256 compression function uses a truncated differential path with a high number of active bytes in the middle and a low number of active bytes at the input and output of each permutation. Due to the wide-trail design strategy, such a path can easily be constructed for `Grøstl`-256. For the attack on 6 rounds, a full active state is placed in the middle of each permutation. The detailed path is given in Fig. 1 and the sequence of active bytes between each round r_i is as follows:

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

By using the same truncated differential path in both permutations P and Q, we can construct a semi-free-start collision for the compression function of `Grøstl`-256 reduced to 6 rounds. In the following, we will show how to find input pairs for P and Q that follow the 6-round differential trail given above by applying a rebound attack.

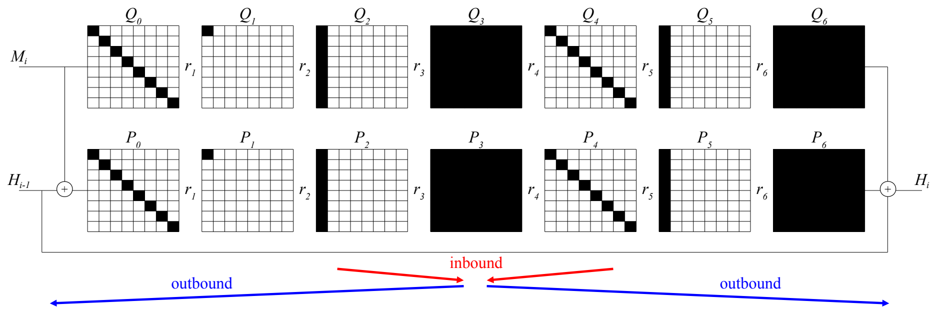

Fig. 1: Overview of the rebound attack on 6 rounds of the `Grøstl`-256 compression function. Black bytes are active.

3.2 The Inbound Phase

We start the rebound attack with the inbound phase in round r_3 and r_4 and deterministically propagate to the full active SubBytes layer in the middle. Hence, we search for differences and values conforming to the truncated differential path shown in Fig. 2. We first choose random differences for the 8 active bytes in P_4. These differences are linearly propagated backward to 64 active bytes at the output of the previous SubBytes layer (P_4^{SB}). Then, we choose random differences for each active byte prior to the MixBytes transformation in P_3^{SH} and linearly propagate forward to the full active input of SubBytes (P_4^{SB}). Note that we can compute each column independently. Next, we need to check whether the input/output differential of all 64 active S-boxes are possible.

For a single S-box, the probability that a random S-box differential exists is about one half, which can be verified by computing the difference distribution table (DDT) of the AES S-box (see [9] for more details). For each valid S-box differential, we get at least two (in some cases 4) possible byte values such that

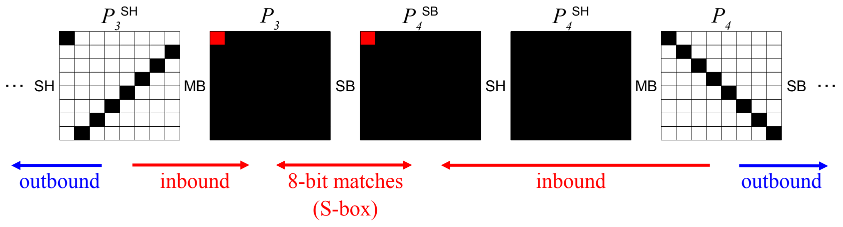

Fig. 2: The inbound phase of the attack on the `Grøstl`-256 compression function using 8-bit S-box matches. The input and output of one S-box is highlighted.

the differential holds. For each column, we try all 2^8 non-zero differences of the according byte in P_3^{SH} and thus, expect one valid differential for all 8 S-boxes of that column. With two independent solutions for each S-box, we get at least 2^8 pairs for one column. Hence, the average complexity to find a valid pair is 1. We repeat this for all 8 active bytes of P_3^{SH} and get about 2^{64} solutions for the inbound phase.

Note that we can choose from about 2^{64} differences for the active bytes in P_4. Hence, we can construct up to 2^{128} pairs that follow the truncated differential path of the inbound phase between state P_3^{SH} and P_4.

3.3 The Outbound Phase

In the outbound phase, we probabilistically propagate the pairs of the inbound phase outwards, to match the differences at the input and output of the permutations. The probability for the propagation from 8 to 1 active byte through the MixBytes transformation in round r_2 is 2^{-56}. Hence, we can construct one pair conforming to the truncated differential path for each of P and Q with a complexity of 2^{56}.

To get a semi-free-start collision, the differences at the input and output of P and Q need to be equal. Note that we can construct pairs for P and Q independently. Hence, we can do a standard birthday attack to match the 8-byte difference at the input, and the 8-byte difference at the output before MixBytes with a complexity of 2^{64}. Since MixBytes is the same linear transformation in both P and Q, the 64 active bytes at the output will match if the differences at the input of MixBytes are equal. Hence, the total complexity for the semi-free-start collision on 6 rounds of the compression function of `Grøstl`-256 is 2^{120} compression function evaluations and 2^{64} memory due to the birthday attack.

4 Extending the Rebound Attack

In this section, we describe three improvements for the rebound attack on `Grøstl`. The first improvements uses 64-bit SuperBoxes [1] instead of 8-bit S-boxes to match the differences in the inbound phase. This idea has already been applied in the improved attack on the Whirlpool hash function in Lamberger et al. [7, Appendix A] and was independently observed in Gilbert and Peyrin [5]. This allows us to extend the inbound phase of the compression function attack on 6 rounds of `Grøstl`-256 by one round. The second idea is to apply the rebound attack to the `Grøstl` hash function by using a common inbound phase at the input of both P and Q. The third contribution addresses `Grøstl`-512. We have constructed new truncated differential paths and apply the rebound attack to the hash and compression function of `Grøstl`-512.

4.1 Improving the Inbound Phase using SuperBoxes

In the standard inbound phase, the differences are computed inwards through MixBytes to the input and output of the intermediate SubBytes layer. Then, each S-box is checked for a valid differential (see Fig. 2). If we consider SuperBoxes instead of S-boxes we can extend the inbound phase by one full active state and get the following sequence of active bytes (instead of $8 \rightarrow 64 \rightarrow 8$):

$$8 \rightarrow 64 \rightarrow 64 \rightarrow 8$$

A SuperBox of `Grøstl` is defined similar to the SuperBox of the AES [1]. For `Grøstl`, the SuperBox consists of 8 parallel S-boxes, followed by one MixBytes transformation and another 8 parallel S-boxes: SB - MB - SB . Note that the SubBytes and ShiftBytes transformations can be interchanged. Hence, a SuperBox behaves like a non-linear 64-bit S-box. Unfortunately, the differential distribution table (DDT) of the SuperBox has 2^{128} entries which is too much for a collision attack on `Grøstl`-256. However, if the input and output differences of the SuperBox are fixed, we can iterate through all 2^{64} input values to check if a given differential holds.

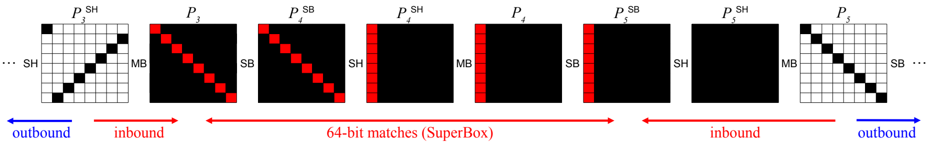

Fig. 3: The inbound phase on the `Grøstl`-256 compression function using 64-bit matches with one SuperBox being highlighted.

In the following, we show how we can still find one solution (pair) for the extended inbound phase with an average complexity of one. We start the inbound phase at state P_3^{SH} and P_5 (see Fig. 3) and proceed as follows:

1. Start with all 2^{64} differences in state P_3^{SH}, compute forwards through MixBytes to state P_3, and store the resulting differences in list L_1.
2. Choose a random difference for state P_5 and compute backward through MixBytes and ShiftBytes to state P_5^{SB}.

3. Connect the output differences of the 8 parallel SuperBoxes (state P_5^{SB}) with the corresponding input differences of the SuperBoxes (state P_3):
   (a) For each SuperBox (column) at state P_5^{SB}, take all 2^{64} possible values and compute both values and differences backward to state P_3.
   (b) We get 2^{64} input differences for each SuperBox in state P_3 and store the resulting differences and values in list L_2.
   (c) To find a solution for the inbound phase, we need to match the 8-byte differences in list L_2 with the corresponding differences of list L_1. Since both lists have 2^{64} entries and we have a condition on 64 bits, we get $2^{64} \times 2^{64} \times 2^{-64} = 2^{64}$ solutions (differences and values) and update L_1 accordingly.
   (d) Repeat this for every SuperBox (column) of state P_5^{SB} and in each case we get 2^{64} solutions again.
4. For the whole inbound phase, we expect 2^{64} solutions with a complexity of 2^{64} in time and memory.

All in all, we can find one solution for the inbound phase with an average complexity of one. Note that we can still choose from 2^{64} differences for state P_5. Hence, we can find up to 2^{128} pairs according to the truncated differential path of the extended inbound phase. In other words, in the inbound phase we can construct up to 2^{128} starting points for the probabilistic outbound phase of the attack.

4.2 Rebound Attack on the Grøstl Hash Function

The main idea of the rebound attack on the Grøstl hash function is to do one half of the inbound phase in each P and Q. We then need to match the differences over the input of the two permutations in the inbound phase (see Fig. 4). The truncated differential path used is similar to the one of the previous section, but "wraps around" the input of P and Q. In this case, the chaining input or IV can be a predefined constant and only the message input (values and differences) is defined by the attack. Note that we use two full active states in each of P and Q since the first ShiftBytes in P and Q cancel out when going around. Hence, the columns of almost two rounds can be solved independently in the inbound phase.

The technique is very similar to the previous section, since we can use independent 64-bit matches again. These two consecutive SuperBoxes (in both P and in Q) are completely independent between state Q_2^{SB} and P_2^{SB}. Again, we can find one solution (pair) for the inbound phase with an average complexity of one. We start the inbound phase with a random difference for state P_2 and compute backward to state P_2^{SB}. Next. we take all 2^{64} nonzero differences in state Q_2, compute backwards to state Q_2^{SB} and store the resulting differences in list L_1. Similar as in Section 4.1, we connect the output difference of the 8 parallel SuperBoxes of P (state P_2^{SB}) with the corresponding output differences of the SuperBoxes of Q (state Q_2^{SB}) by merging lists of size 2^{64}. We get 2^{64} solutions with a complexity of 2^{64} in time and memory. Again, we can repeat the inbound

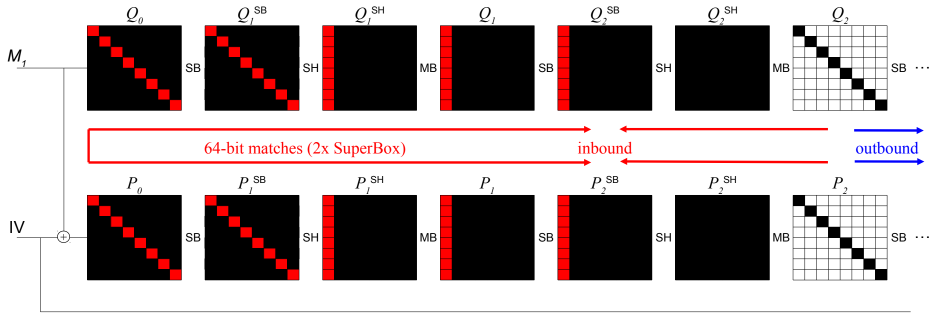

Fig. 4: The inbound phase of the attack on the hash function `Grøstl`-256 with one 64-bit match (two SuperBoxes) being highlighted.

phase about 2^{64} times with other starting differences in P_2. Hence, we can construct up to 2^{128} starting points for the subsequent probabilistic outbound phase of the attack.

4.3 Constructing Truncated Differential Paths for `Grøstl-512`

The difficult part of the rebound attack on `Grøstl`-512 is to find a "good" truncated differential path. However, using a match-in-the-middle on the SuperBox, we can construct a path with similar properties as for `Grøstl`-256. The complexity of the rebound attack is determined by the outbound phase. Hence, we need a truncated differential path with as few active bytes in the outbound phase as possible. Similar to `Grøstl`-256, a straightforward truncated differential path starts with (a minimum of) 8 active bytes at both ends of the inbound phase. In the following, we show how the inbound phase of such a path works for the hash function, and how to get a valid truncated differential path for the inbound phase of the compression function as well.

The Hash Function. For the rebound attack on the `Grøstl`-512 hash function, the truncated differential path of the inbound phase is given in Fig. 5. Due to the symmetry of the ShiftBytes transformations in P and Q, we can again do the 64-bit matches over each two SuperBoxes independently (see Section 4.2). Contrary to the `Grøstl`-256 case, some output differences of the SuperBoxes in state P_2^{SB} and Q_2^{SB} are zero. However, the list L_1 still contains 2^{64} entries and we also generate 2^{64} differences for the list L_2 by iterating through all values of each SuperBox. Again, we have a condition on 64 bits (including zero differences) and thus, still expect 2^{64} solutions with a complexity of 2^{64}. Since we can choose from 2^{64} differences for both P_2^{SB} and Q_2^{SB}, we again expect to find 2^{128} solutions for the inbound phase.

The Compression Function. For the `Grøstl`-512 compression function, a differential path with 8 active bytes at each end of the inbound phase does not work (see Fig. 6). Although we use a SuperBox in the inbound phase this results in an impossible truncated differential path. For most columns of the MixBytes transition in the middle, the sum of active bytes at input and output is below 9, which is not possible according to the MDS property of MixBytes. With only 8 active bytes in state P_3^{SH} and P_5, we do not get enough active bytes for a valid MixBytes transformation in round r_4. Also rotating the position of active bytes in state P_3^{SH} or P_5 does not give a valid truncated differential path. However, we can add a second active column at the output of the inbound phase (see Fig. 7). This results in an almost full active state in round r_4 and the truncated differential path is valid. Again, we can apply the same technique as in the previous section and expect 2^{64} solutions of the inbound phase with a complexity of 2^{64} by merging lists of size 2^{64}. Note that with 24 active bytes in P_3^{SH} and P_5, we can get up to 2^{192} solutions (starting points in the outbound phase) in the inbound phase.

5 Results of Rebound Attacks on Reduced `Grøstl`

In this section, we apply the improved inbound techniques of the previous section to the round-reduced `Grøstl` hash functions and compression functions.

5.1 Collisions for 4 Rounds of `Grøstl`-256

The complete truncated differential path for the collision attack on 4 rounds of the `Grøstl`-256 hash function is given in Fig. 8. The sequence of active bytes in each round for both, P and Q are given as follows:

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

The details for the inbound phase of the attack are given in Section 4.2. Remember that we get 2^{64} pairs with a complexity of 2^{64} conforming to the truncated differential path up to round r_2. In the outbound phase, each of these pairs propagate to the output of the permutations according to the truncated differential path given in Fig. 8 with a probability of one. To get a zero output difference of the hash function, the 8-byte differences prior to the last MixBytes need to be the same (see Section 3.3). Since we have 2^{64} solutions for the inbound phase, and we have a 64-bit condition in the outbound phase, we expect to get one pair which results in a collision. The complexity of this collision attack on the `Grøstl`-256 hash function is thus, 2^{64} in both time and memory.

Note that using the previous techniques a collision attack on 5 rounds according to the following truncated differential path for both, P and Q is not possible:

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

Each of the two $8 \rightarrow 1$ transitions of MixBytes in round r_3 have a probability of 2^{-56}. Together with the probabilistic match on 64 bits at the end of the path, the total complexity is $2^{56+56+64} = 2^{176}$ which exceeds the generic complexity for a collision attack on `Grøstl`-256.

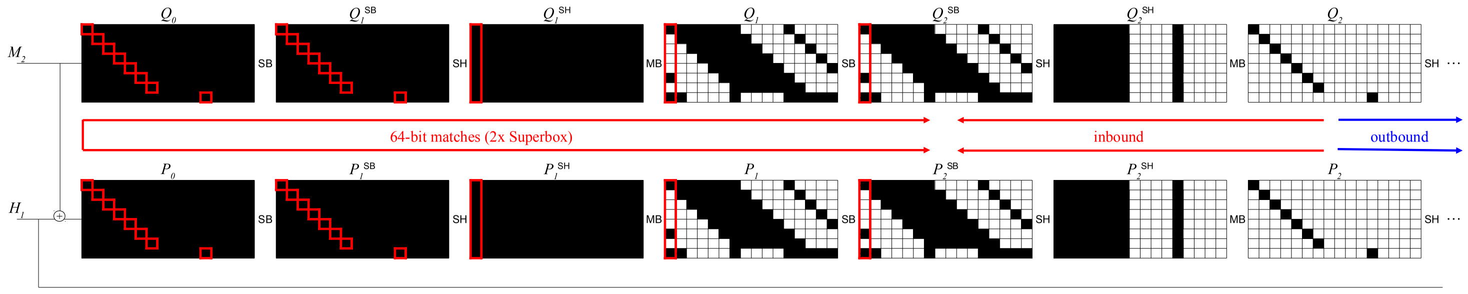

Fig. 5: Inbound phase of the attack on the `Grøstl`-512 hash function with one 64-bit match (two SuperBoxes) being highlighted.

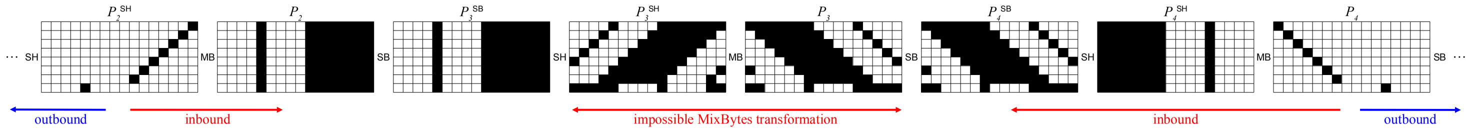

Fig. 6: Impossible inbound phase of the attack on the `Grøstl`-512 compression function.

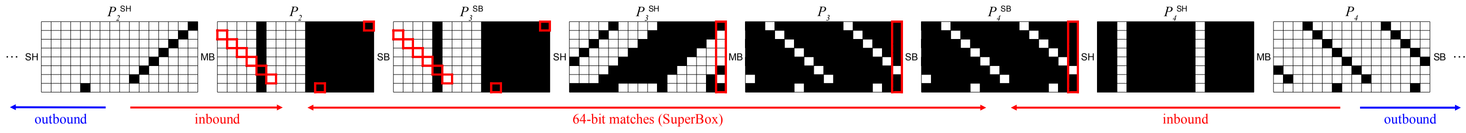

Fig. 7: Inbound phase of the attack on the `Grøstl`-512 compression function with one 64-bit match (SuperBox) being highlighted.

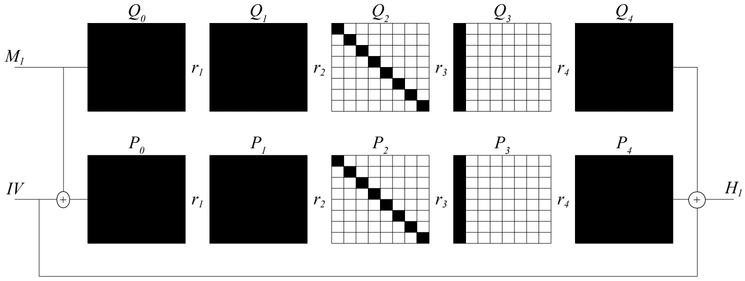

Fig. 8: Truncated differential path for the collision attack on 4 rounds of the `Grøstl`-256 hash function.

5.2 Collisions for 5 Rounds of `Grøstl`-512

Contrary to the collision attack on `Grøstl`-256 we can extend the truncated differential path for `Grøstl`-512 to 5 rounds, with the following number of active bytes in each, P and Q:

$$128 \xrightarrow{r_1} 64 \xrightarrow{r_2} 8 \xrightarrow{r_3} 1 \xrightarrow{r_4} 8 \xrightarrow{r_5} 64$$

The complexity of the outbound phase is given by the two probabilistic $8 \rightarrow 1$ transitions of MixBytes in round r_3 of P and Q, and the match of the 64-bit differences prior to the last MixBytes transformation in round r_5. Hence, the total complexity of the attack is $2^{56+56+64} = 2^{176}$ compression function evaluations. Note that we need to construct 2^{176} solutions in the inbound phase for the attack to succeed. However, as shown in Section 4.3, we can only find up to 2^{128} pairs for the inbound phase.

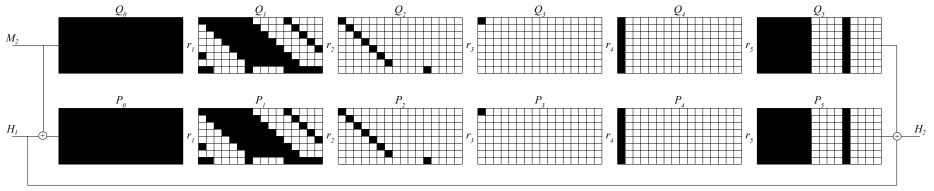

Fig. 9: Truncated differential path for the collision attack on 5 rounds of the `Grøstl`-512 hash function. An additional first block is used to generate enough freedom for the attack to succeed.

We can get the needed additional freedom for a 5 round collision attack by prepending a first message block. The collision attack works as follows. First we choose an arbitrary first message block. Then, we repeat the inbound phase for all 2^{128} possible starting points to get 2^{128} solutions. Since the probability of the outbound phase is 2^{-176} we need to repeat the inbound phase with 2^{48} different first message blocks to find a collision for 5 rounds. The total complexity of the attack is about $2^{64+56+56} = 2^{176}$ compression function evaluations and 2^{64} memory.

5.3 Semi-Free-Start Collision for 7 Rounds of Grøstl-256

The improved inbound phase using the SuperBox allows to extend the 6-round semi-free-start collision attack on Grøstl-256 by one round. The truncated differential path is given in Fig. 10. The sequence of active bytes in each round for both, P and Q are given as follows:

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 64 \xrightarrow{r_5} 8 \xrightarrow{r_6} 8 \xrightarrow{r_7} 64$$

The details of the inbound phase of the attack are given in Section 4.1 and we can get one pair with an average complexity of one. The solutions of the inbound phase are propagated outwards as in the attack on 6 rounds (see Section 3.3). We have one $8 \rightarrow 1$ MixBytes transition in round r_2 with probability 2^{-56}, and a birthday match on $2 \cdot 64$ bits at the input and output with complexity 2^{64}. Hence, the total complexity of the attack is 2^{120} compression function evaluations and 2^{64} memory.

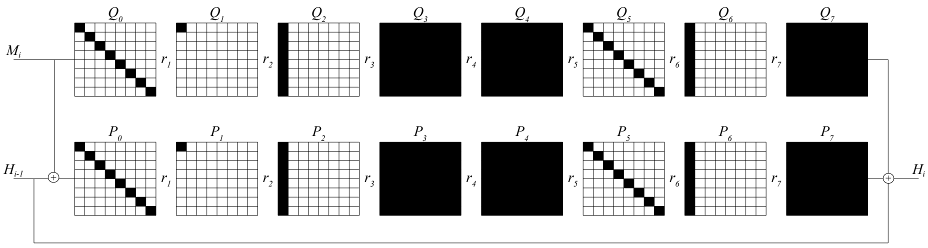

Fig. 10: The truncated differential path for the semi-free-start collision on 7 rounds of the compression function of Grøstl-256.

Note that it seems to be difficult to extend this attack to 8 rounds. Adding one more $8 \rightarrow 1$ transition in the outbound phase, increases the complexity of the attack to be above 2^{128}. If we extend the truncated differential path at the beginning or end of the permutation, we need to match a full active state which has a birthday complexity of at least 2^{256}. By adding a third full active state in the middle, the columns in the match-in-the-middle phase are not independent anymore and we would need to match the differences of a full active state.

5.4 Semi-Free-Start Collision for 7 Rounds of Grøstl-512

The truncated differential path for the inbound phase of the rebound attack on the Grøstl-512 compression function has 8 active bytes in round r_3 and 16 active bytes in round r_5. The resulting 7-round truncated differential path is similar to the Grøstl-256 case (see Fig. 11) and the sequence of active bytes is given as follows:

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 110 \xrightarrow{r_5} 16 \xrightarrow{r_6} 16 \xrightarrow{r_7} 110$$

In the inbound phase, we connect the differences between the input of SubBytes of round r_4 and the output of SubBytes of round r_5 by using the SuperBox again. We get one solution with an average complexity of one.

The complexity of the attack is determined by the outbound phase. We have one probabilistic $8 \to 1$ MixBytes transition in round r_2, and do a birthday match in 8 active bytes at the beginning and 16 active bytes at the end of the path. Hence, the total complexity for the collision attack on 7 rounds is $2^{56+32+64} = 2^{152}$ with memory requirements of 2^{64} due to the inbound phase and birthday match.

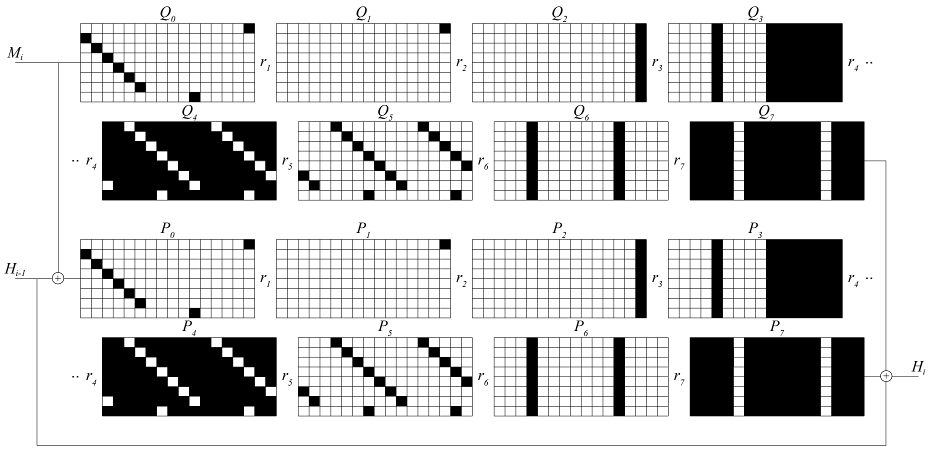

Fig. 11: Truncated differential path for the semi-free-start collision on 7 rounds of Grøstl-512.

Although we could construct an 8-round truncated differential path with the following number of active bytes, we cannot find enough pairs for a collision attack on the compression function:

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 110 \xrightarrow{r_5} 16 \xrightarrow{r_6} 2 \xrightarrow{r_7} 8 \xrightarrow{r_8} 64$$

The path is constructed by carefully placing the positions of active bytes in round r_6 such that the two active bytes are shifted into the same column in round r_7. With three $8 \to 1$ MixBytes transitions and a birthday match on $2 \cdot 64$ bits at the input and output, we would get a total complexity of $2^{3 \cdot 56 + 2 \cdot 32} = 2^{232}$. Note that we get only $2^{3 \cdot 64} = 2^{192}$ solutions for the inbound phase (see Section 4.3). After the three probabilistic MixBytes transitions, we get only $2^{192 - 3 \cdot 56} = 2^{24}$ valid pairs for each permutation. Contrary to the Grøstl-512 hash function, we cannot use the freedom of a previous message block in the middle of the compression function. Hence, this attack on 8-rounds of Grøstl-512 compression function does not work.

6 Conclusion

In this work, we have presented a variety of new results on the SHA-3 candidate Grøstl. We improve the rebound attack on the compression function of Grøstl-256 by one round and provide the first results for Grøstl-512. Most importantly, we give the first cryptanalytic results for the Grøstl hash function and achieve 4 out of 10 rounds for Grøstl-256, and 5 out of 14 rounds for Grøstl-512. This allows to reason about the security margin of Grøstl and compare it with other hash functions based on different building blocks. However, for many candidates, only results on their underlying compression function, permutation or block cipher are known at this point.

The given results allow for the first time a high-level comparison between permutation based and block-cipher based hashing from a cryptanalytic perspective. The block-cipher based Whirlpool hash function and the permutation based Grøstl hash function share a number of similarities: 8-bit S-boxes arranged in an 8x8 geometry and AES-like round transformations. The S-boxes are different, but their exact specification does not make a difference with respect to the attacks we consider here. Whereas the rebound attack can break up to 8 rounds of the Whirlpool hash function [6,7] with complexity below 2^{128}, it can only break 4 rounds of the Grøstl hash function with complexity below 2^{128}. The main reason is the fact that in most block-cipher designs round keys are added at several places during the computation, also in the block cipher at the core of Whirlpool. Used in an unkeyed setting, this mixing of inputs during the computation gives an attacker easier access for manipulating internal state variables, and in turn allows more efficient attacks.

The ideas presented in this paper are also applicable to other AES-based hash functions like ECHO, SHAvite-3, LANE, and Cheetah. Additionally, future work will include the application of the rebound idea to other hash function constructions. This may require more sophisticated tools to obtain appropriate (truncated) differential paths first, whereas for the so far considered AES-based constructions, good differentials are easily obtainable “by hand”.

References

1. Daemen, J., Rijmen, V.: Understanding Two-Round Differentials in AES. In: Prisco, R.D., Yung, M. (eds.) Security and Cryptography for Networks 2006, Proceedings. LNCS, vol. 4116, pp. 78–94. Springer (2006)
2. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer (2006)

3. Fouque, P.A., Stern, J., Zimmer, S.: Cryptanalysis of Tweaked Versions of SMASH and Reparation. In: Avanzi, R., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography 2008, Proceedings. LNCS, vol. 5381, pp. 136–150. Springer (2009)
4. 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), available online at `http://www.groestl.info`.
5. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations. Cryptology ePrint Archive, Report 2009/531 (2009), `http://eprint.iacr.org/`
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Cryptanalysis of the Whirlpool Hash Function, manuscript.
7. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009, Proceedings. LNCS, vol. 5912, pp. 126–143. Springer (2009)
8. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography 2009, Proceedings. LNCS, vol. 5867, pp. 16–35. Springer (2009)
9. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) Fast Software Encryption 2009, Proceedings. LNCS, vol. 5665, pp. 260–276. Springer (2009)
10. 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)
11. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register Notice (November 2007), `http://csrc.nist.gov`
12. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In: EUROCRYPT 2007. pp. 1–22. Springer (2007)
13. Tillich, S., Feldhofer, M., Kirschbaum, M., Plos, T., Schmidt, J.M., Szekely, A.: High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein. Cryptology ePrint Archive, Report 2009/510 (2009), `http://eprint.iacr.org/`
14. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005, Proceedings. LNCS, vol. 3621, pp. 17–36. Springer (2005)
15. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005, Proceedings. LNCS, vol. 3494, pp. 19–35. Springer (2005)