Collision Attack on 5 Rounds of Grøstl

Florian Mendel[1](✉), Vincent Rijmen[2], and Martin Schläffer[1]

[1] IAIK, Graz University of Technology, Graz, Austria
florian.mendel@iaik.tugraz.at
[2] Department ESAT/COSIC, KU Leuven and Security Department, iminds, Ghent, Belgium

Abstract. In this article, we describe a novel collision attack for up to 5 rounds of the Grøstl hash function. This significantly improves upon the best previously published results on 3 rounds. By using a new type of differential trail spanning over more than one message block we are able to construct collisions for Grøstl-256 on 4 and 5 rounds with complexity of 2^{67} and 2^{120}, respectively. Both attacks need 2^{64} memory. Due to the generic nature of our attack we can even construct meaningful collisions in the chosen-prefix setting with the same attack complexity.

Keywords: Hash functions · SHA-3 candidate · Grøstl · Collision attack

1 Introduction

In the last few years the cryptanalysis of hash functions has become an important topic within the cryptographic community. Especially the collision attacks on the MD4 family of hash functions (MD5, SHA-1) have weakened the security assumptions of these commonly used hash functions [26–28]. As a consequence NIST has decided to organize a public competition in order to design a new hash function, leading to the selection of Keccak as SHA-3 [19].

During the SHA-3 competition, the three classical security requirements (collision-, preimage- and second-preimage resistance) were not the main target of cryptanalytic attacks. Most results were published on building blocks such as the compression function, block cipher or permutation used in a hash function. Additionally, many distinguishers on these building blocks with minor relevance in practice were considered. Although these results are important from a theoretical point of view, vulnerabilities that can be exploited for the hash function are certainly more important.

In this work, we present new results on the collision resistance of the SHA-3 finalist Grøstl. Grøstl is an iterated hash function based on design principles very different from those used in the MD4 family. The compression function of Grøstl is built from two different permutations that follow the design strategy of the Advanced Encryption Standard (AES) [3,17]. The simple construction of the compression function and the byte-oriented design of Grøstl facilitates the security analysis. In the last years Grøstl has received a large amount of cryptanalysis. However, most of the analysis focus on the building blocks of Grøstl and only a few results have been published for the hash function so far.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 509–521, 2015.
DOI: 10.1007/978-3-662-46706-0_26

Related Work. Grøstl is one of the SHA-3 candidates that has probably received the largest amount of cryptanalysis during the competition. Security analysis of Grøstl was initiated by the design team itself which led to the rebound attack [15]. Since then, several improvements to the rebound attack technique have been made, leading to new results on both the hash function [16] and its underlying components [6,14,22]. For the final round, Grøstl was been tweaked to thwart internal differential attacks [7,21] and to reduce the impact of the rebound attack and its extensions.

The best published attacks on the final version of both Grøstl-256 and Grøstl-512 are collision attacks on 3 rounds of the hash function and on 6 rounds of the compression function [9,23]. Preimage attacks for the compression function of Grøstl-256 and Grøstl-512 have been shown in [29] for 5 and 8 rounds, respectively. Additionally, non-random properties of the Grøstl permutation have been discussed in [1,8]. For a detailed overview of the existing attacks on Grøstl we refer to the ECRYPT II SHA-3 Zoo [4].

Our Contribution. By using a new type of differential trail we are able to show collision attacks on Grøstl for up to 5 rounds. The extension becomes possible by considering differential trails spanning over more than one message block to iteratively cancel differences in the chaining variable. Our new attack combines ideas of the attack on SMASH [13] with the rebound attack [15] on Grøstl. A similar approach has also been used in the attack on Grindahl [20]. The results are collision attacks on the Grøstl-256 hash function reduced to 4 and 5 rounds with a complexity of 2^{67} and 2^{120}, respectively. Both attacks have memory requirements of 2^{64}. Note that the best previously known collision attack on the Grøstl hash function was on 3 rounds with a complexity of 2^{64} [23]. We want to note that the same attack also applies to 5 rounds of Grøstl-512.

Additionally, we show that due to the generic nature of our attack we can construct collisions in the chosen-prefix setting with the same complexity. It has been demonstrated in [24,25] that chosen-prefix collisions can be exploited to construct colliding X.509 certificates and a rogue CA certificate for MD5. Note that in most cases constructing such collisions is more complicated than constructing (random) collisions. Our results and related work for the Grøstl hash function are shown in Table 1.

Outline. The paper is structured as follows. In Sect. 2, we give a short description of the Grøstl hash function. The basic attack strategy and the collision

Table 1. Collision attacks on the Grøstl-256 hash function.

Rounds	Complexity	Memory	Reference
3	2^{64}	-	[23]
4	2^{67}	2^{64}	This work
5	2^{120}	2^{64}	This work

attack for 4 rounds of the hash function is presented in Sect. 3. In Sect. 4, we describe the extension of the attack to 5 rounds, and in Sect. 5 the construction of meaningful collisions is discussed. Finally, we conclude in Sect. 6.

2 Short Description of Grøstl

The hash function Grøstl [5] was one of the 5 finalists in the SHA-3 competition [18]. Grøstl is an iterated hash function with a compression function built from two distinct permutations P and Q, which are based on the AES design principles. In the following, we describe the components of the Grøstl hash function in more detail.

2.1 The Hash Function

The two main variants, Grøstl-256 and Grøstl-512 are used for hash output sizes of $n = 256$ and $n = 512$ bits. The hash function first pads the input message M and splits the message 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 message blocks are processed via the compression function $f(h_{i-1}, m_i)$ and output transformation $\Omega(h_t)$. The size of the chaining value h_i is ℓ bits as well.

$$\begin{aligned} h_0 &= IV \\ h_i &= f(h_{i-1}, m_i) \quad \text{for } 1 \leq i \leq t \\ h &= \Omega(h_t). \end{aligned}$$

The compression function f is based on two ℓ-bit permutations P and Q (sometimes denoted by P_ℓ and Q_ℓ) and is defined as follows:

$$f(h_{i-1}, m_i) = P(h_{i-1} \oplus m_i) \oplus Q(m_i) \oplus h_{i-1}.$$

The output transformation Ω is applied to h_t to give the final hash value h of size n, where $\text{trunc}_n(x)$ discards all but the least significant n bits of x:

$$\Omega(h_t) = \text{trunc}_n(P(h_t) \oplus h_t).$$

2.2 The Permutations P and Q

The two permutations P and Q are designed according to the wide trail strategy [2] and their structure is very similar to the AES. In Grøstl-256 each permutation updates an 8×8 state of 64 bytes in 10 rounds. In one round, the round transformation updates the state by means of the sequence of transformations

$$\mathsf{MB} \circ \mathsf{SH} \circ \mathsf{SB} \circ \mathsf{AC}\,.$$

In the following, we briefly describe the round transformations of P and Q used in the compression function f in more detail.

AddRoundConstant (AC). In this transformation, the state is modified by combining it with a round constant with a bitwise `xor` operation. Different constants are used for the permutations P and Q.

SubBytes (SB). The SubBytes transformation is the same for P and Q and is the only non-linear transformation of the permutations. It is a permutation consisting of an S-box applied to each byte of the state. The 8-bit S-box is the same as in the AES with good cryptographic properties against differential and linear attacks. For a detailed description of the S-box, we refer to [17].

ShiftBytes (SH). The ShiftBytes transformation is a byte transposition that cyclically shifts the rows of the state over different offsets. The ShiftBytes transformation is different for the two permutations P and Q.

MixBytes (MB). The MixBytes transformation is a permutation operating on the state column by column. To be more precise, it is a left-multiplication by an 8×8 MDS matrix over $\mathbb{F}_{2^8}$. The coefficients of the matrix are determined in such a way that the *branch number* of MixBytes (the smallest nonzero sum of active input and output bytes of each column) is 9, which is the maximum possible for a transformation with these dimensions. This transformation is the same for both permutations P and Q.

2.3 Alternative Description of `Grøstl`

To simplify the description of attack in the following sections, we use an equivalent alternative description of `Grøstl`. Let P' and Q' denote the permutation P and Q without the last application of MixBytes. Then, by setting

$$\begin{aligned} h'_0 &= \mathsf{MB}^{-1}(\mathrm{IV}) \\ h'_i &= P'(\mathsf{MB}(h'_{i-1}) \oplus m_i) \oplus Q'(m_i) \oplus h'_{i-1} \quad \text{for } 1 \le i \le t \\ h &= \Omega(\mathsf{MB}(h'_t)) \end{aligned}$$

with $h_i = \mathsf{MB}(h'_i)$, we get an equivalent description of `Grøstl`, where the last MixBytes transformation of the permutations has been swapped with the XOR operation of the feed-forward.

3 Collision Attack for 4 Rounds of `Grøstl`

To get improved attacks on the `Grøstl` hash function, we view `Grøstl` as a strengthened variant of SMASH [10]. The essential difference between the two designs is that `Grøstl` employs a second nonlinear permutation Q, where SMASH employs a scaling by the constant θ, i.e. a linear map (see Fig. 1). The hash function SMASH has been broken by subsequently controlling the output difference of the compression function using the linearity of θ. After the application of 257 respectively 513 message blocks, a colliding output difference can be constructed [13]. In this section, we show how to achieve the same for 4 rounds of `Grøstl` by having differences in only one permutation.

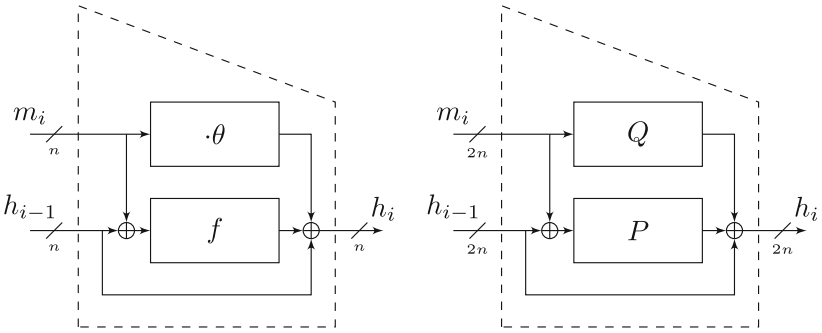

Fig. 1. The compression function of (a) SMASH and (b) Grøstl.

3.1 The Second-Preimage Attack on SMASH

The second-preimage attack on SMASH presented in [13] is based on the technique of *controllable output differences* for the compression function. By carefully selecting consecutive message blocks, an attacker can step-by-step convert an arbitrary starting difference in the chaining variable into an arbitrary output difference. The attack is deterministic and the number of consecutive controllable message blocks is equal to the length of the chaining variable. The nonlinearity of f is made ineffective by strictly controlling its input differences and values. Controlling the input values of f implies that the input values of the linear map are determined. Fortunately, for a linear map it suffices to know the input *difference* to compute the output difference. The output difference of the linear map is controlled by the number of message blocks.

3.2 Application to Reduced Grøstl

At the first sight, the attack on SMASH does not apply to Grøstl, because the strong nonlinearity of P and Q makes it difficult to control the output differences of both permutations. However, by having no differences in Q, we can use the whole freedom of the message block to control the differential propagation in P. Since we cannot control the differences completely, we need to apply a variation of the technique on SMASH, to get a zero output difference at the compression function.

Our attack will start from an arbitrary difference in the chaining variable and convert it into an output difference equal to zero after 9 steps. The first message block can be selected arbitrarily. The only requirement is a difference in the message. The next 8 message blocks are fully controlled by the attacker and must not contain any differences. Then, each of the 8 message blocks is used to cancel one eighth of the differences at the output of the compression function to result in a collision at the end (see Fig. 2).

3.3 Details of the Attack

To simplify the description of the attack we use the alternative description of Grøstl given in Sect. 2.3. Since the last MixBytes transformation is moved out of the compression function, the limited set of differences at the output are more clearly visible.

The core of our collision attack on the reduced hash function are truncated differential trails with only 8 active bytes at the output of P'. Two full active states are placed at the beginning and the number of active bytes for the 4-round trail are given as follows:

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

For such a truncated trail, we can construct a pair following the trail with an amortized complexity of 1 (even for a given input differences). We postpone the detailed explanation how to do so until Sect. 3.4.

The high-level overview of the 4-round attack is shown in Fig. 2. In each iteration, the differences in 8 bytes are canceled. Since this has a probability 2^{-64}, we need to compute 2^{64} pairs for P' (for the given input differences) to find a *right* pair that result in the desired output difference. The attack can then be summarized as follows:

1. Choose arbitrary message blocks m_1, m_1^* and compute h'_1. Repeat this until one gets a full active state in h'_1. Note that randomly selected m_1, m_1^* produce a full active state in h'_1 with probability at least 3/4.
2. Use a *right* pair for P' following the trail of (1) to cancel 8 bytes of the difference in the state h'_2, cf. Fig. 2.
3. Use a *right* pair for P' for a rotated variant of the trail of (1) to cancel another 8 bytes of the difference in the state h'_3.
4. Repeat steps 3–4 in total 8 times until a collision has been found in h'_9.

The complexity of the attack is 8 times finding a *right* pair for P' to iteratively cancel the difference in the state $h'_2, \ldots, h'_9$. We will show in the following section that such a *right* pair can be constructed with complexity of 2^{64}, resulting in a total attack complexity of $8 \cdot 2^{64} = 2^{67}$.

3.4 Finding a Right Pair for P'

In this section, we show how to find a *right* pair for P' reduced to 4 rounds following the truncated differential trail in (1) using the rebound attack [15]. Note that the input difference is fixed by $\mathsf{MB}(h'_{i-1})$ and we target an output difference such that 8 bytes of the difference in h'_i can be canceled. Unlike the classical rebound attack, the inbound phase is placed at the beginning and covers the first two rounds, while the outbound phase covers the last two rounds.

Using super-box matches [6,11,12], we can find 2^{64} pairs (solutions) for the inbound phase with a complexity of 2^{64} in time and memory. In the outbound phase, all these pairs will follow the 4 round truncated differential trail with a

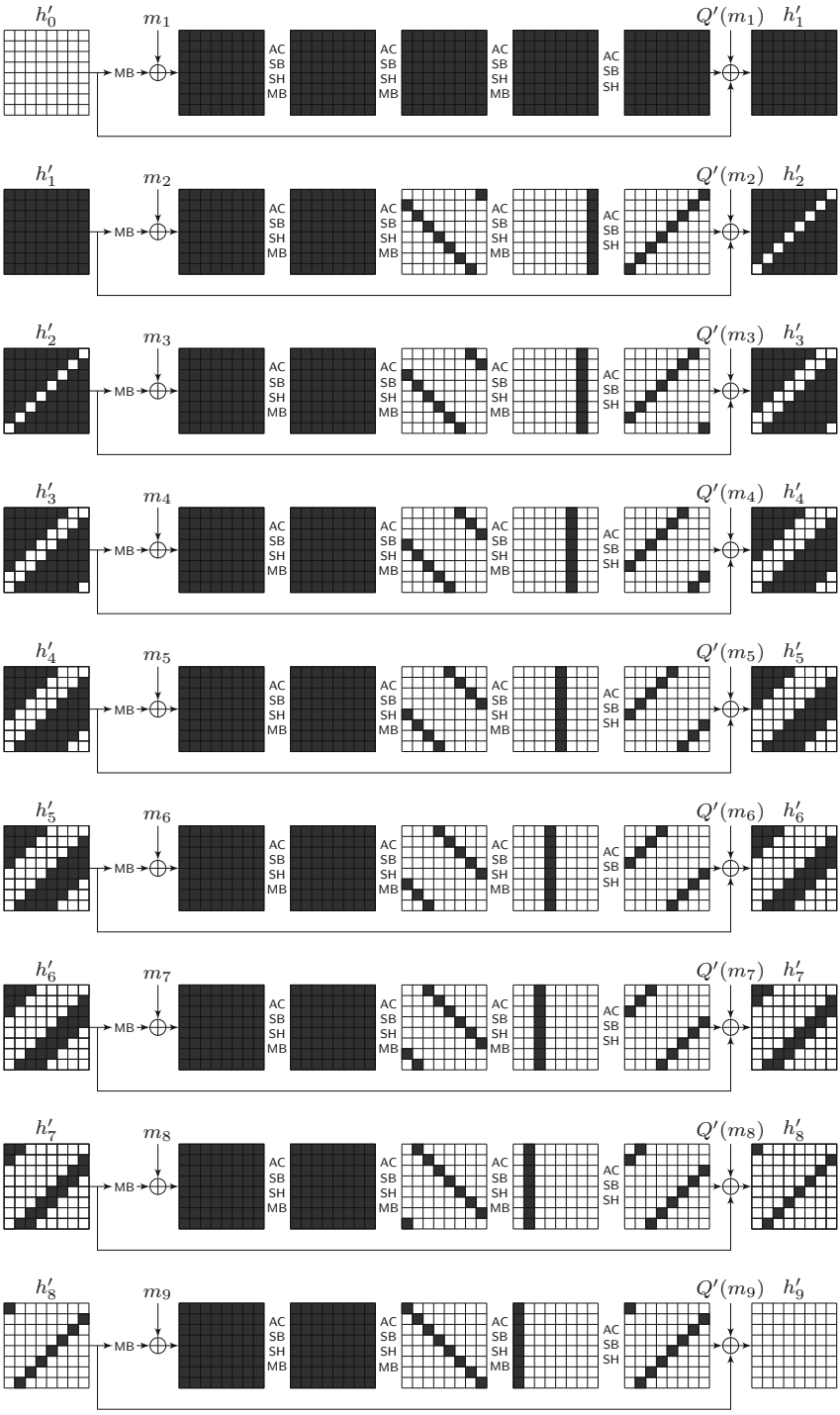

Fig. 2. Overview of the attack on 4 rounds.

probability of 1 and one of these 2^{64} pairs will match the desired output difference (condition on 64 bits). In other words, using the rebound attack we can find a *right* pair for P' with a complexity of 2^{64} in time and memory.

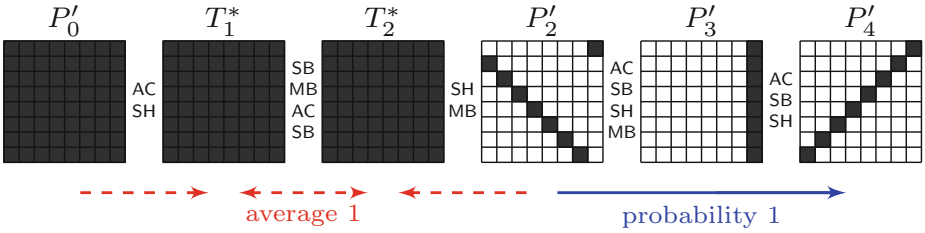

Fig. 3. Truncated differential trail for P' used in the attack on 4 rounds.

In order to make the subsequent description of the rebound attack easier, we swap the SubBytes and ShiftBytes transformation in the first round of permutation P' (see Fig. 3). Note that this can always be done without affecting the output of the round. Then, the attack can be summarized as follows:

1. Compute the input difference of the permutation (P'_0) forward to state T^*_1.
2. Compute all 2^{64} differences of state P'_2 backward to state T^*_2 and store them in a list L.
3. Connect the single difference of state T^*_1 with the 2^{64} differences of state T^*_2 using independent super-box matches. For each column $c = \{0, 1, \ldots, 7\}$ we proceed as follows:
   (a) Take all 2^{64} values for column c of state T^*_1 and compute both values and differences forward to column c of state T^*_2.
   (b) Check for matching 8-byte column differences in list L. Since we compute 2^{64} differences forward and have 2^{64} entries in L, we get 2^{64} solutions (differences and values) for the match. We update L to contain these 2^{64} solutions.
4. For each column and thus, for the whole inbound phase the number of resulting solutions is 2^{64}. The total complexity is 2^{64} in time and memory.

Since the truncated differential trail in the outbound part (the last 2 rounds) has probability 1, we get in total 2^{64} pairs following the truncated differential trail and one of these pairs is expected to be a right pair, i.e. result in the desired output difference (condition on 64 bits).

4 Extending the Attack to 5 Rounds

In this section, we present a collision attack for the `Grøstl`-256 hash function reduced to 5 rounds with a complexity of about 2^{120} and memory requirements of 2^{64}. The attack is an extension of the attack on 4 rounds. However, since

the freedom in finding right pairs for the 5-round trail is limited, we need more message blocks for the attack to succeed. In the attack, we use the following sequence of active bytes in P' (cf. Fig. 4):

$$64 \xrightarrow{r_1} 64 \xrightarrow{r_2} 8 \xrightarrow{r_3} 1 \xrightarrow{r_4} 8 \xrightarrow{r_5} 8. \tag{2}$$

However, it is important to note that for this truncated differential trail (with a fixed input difference) only 2^8 pairs exist, in contrast to 2^{64} for the 4 round trail. This complicates the application of the attack. The complexity of finding these 2^8 pairs is 2^{64} using the rebound attack, as described in Sect. 3.4.

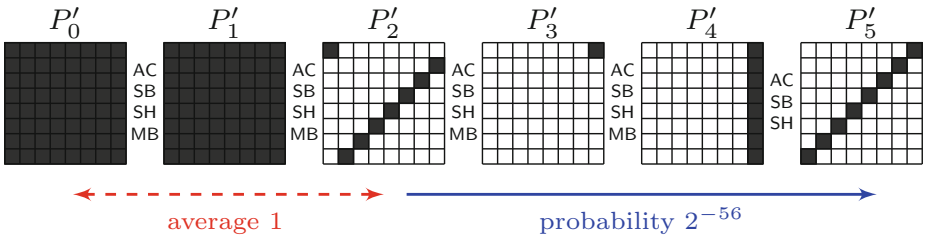

Fig. 4. Truncated differential trail for P' used in the attack on 5 rounds.

4.1 Details of the Attack

Since we can construct only 2^8 pairs following the truncated differential trail, but need to cancel a 64-bit difference at the output of the compression function, each step of the attack succeeds only with a probability of 2^{-56}. However, this can be compensated for by using more message blocks in each step of the attack. Then, the attack can be summarized as follows:

1. Use the rebound attack (cf. Sect. 3.4) to find 2^8 pairs following the truncated differential trail. This has a complexity of 2^{64} in time and memory.
2. For each of these 2^8 pairs check if it can be used to cancel the corresponding 8 bytes of differences in state h'_i. This has a probability of 2^{-56}.
3. If no such *right* pair exists, then choose arbitrarily one of the 2^8 pairs and compute the state h'_i. This generates a new starting point with new differences for the next iteration, while keeping the same bytes inactive.
4. After 2^{56} new starting points, we expect to find a *right* pair with the desired output difference.

Since we need 2^{56} new starting points to cancel the differences in 8 bytes of the state, the complexity of the attack is equivalent to $8 \cdot 2^{64+56} = 2^{123}$ compression function evaluations. Note that the length of such a colliding message is about $8 \cdot 2^{56} = 2^{59}$ blocks.

4.2 Reducing the Length of the Colliding Message Pair

Beside the large time and memory complexity of the attack, one might also see the length of the colliding message pair as a limiting factor of the attack. However, the length of the colliding message pair can be significantly reduced by using a tree-based approach. Instead of choosing only one of the 2^8 pairs to generate a new starting point, we can continue with all pairs in parallel. By using a huge tree with 8 levels and 2^8 branches at each level, we get $(2^8)^8 = 2^{64}$ nodes at level 8. One of these 2^{64} nodes will have the desired output difference. This way, the length of the colliding message pair can be reduced to $8 \cdot 8 + 1 = 65$ message blocks.

4.3 Improving the Complexity of the Attack

The complexity of the attack can be slightly improved by using denser characteristics except when canceling the last 8 bytes. Instead of using a truncated differential trail with a $8 \rightarrow 1$ transition in round 3 of the trail, we can use truncated differential trails with $8 \rightarrow 8, 8 \rightarrow 7, \ldots, 8 \rightarrow 2$ which have a probability greater than 2^{-48}. The complexity of the attack is then dominated by the last iteration where we still need an $8 \rightarrow 1$ transition. This will improve the attack complexity by a factor of 8 resulting in a total complexity of 2^{120} compression function evaluations and 2^{64} memory.

5 Collisions in the Chosen-Prefix Setting

In a collision attack on a hash function an attacker has to find two arbitrary messages M and M^* such that $H(M) = H(M^*)$. However, in practice it might be required that the two messages contain some meaningful information, such that it can be used to practically compromise a cryptographic system. Such an example are, for instance, collisions in the chosen-prefix setting, where an attacker searches for a pair (M, M^*) such that

$$H(M_{pre} \| M) = H(M^*_{pre} \| M^*) \tag{3}$$

for a chosen-prefix (M_{pre}, M^*_{pre}). In [24,25], it was shown that such a more powerful attack exists for MD5. Moreover, the application of the attack to construct colliding X.509 certificates and the creation of a rogue certification authority certificate has been shown.

However, in most cases constructing such collisions is more complicated than constructing (random) collisions. In the case of MD5 the collision attack in the chosen-prefix setting has a complexity of 2^{49}, while the currently best collision attack on MD5 has a complexity of 2^{16}. However, in `Grøstl` the collision attack in the chosen-prefix setting has the same complexity as the collision attack. Due to the generic nature of the collision attack, differences in the chaining variables can be canceled efficiently (cf. Sect. 3).

6 Conclusion

In this work, we have provided new and improved cryptanalysis results for the `Grøstl` hash function, which significantly improving on previously known results. To be more precise, by using a new type of differential trail we were able to show collision attacks on `Grøstl` for up to 5 rounds. The extension becomes possible by considering differential trails spanning over more than one message block.

Moreover, due to the generic nature of our attack we can also construct meaningful collisions, i.e. collisions in the chosen-prefix setting with the same complexity. It has been shown in the past that such collisions might be exploited for instance to construct colliding X.509 certificates.

Although our results do not threaten the security of `Grøstl`, we believe that they will lead to a better understanding of the security margin of the hash function.

Acknowledgments. The work has been supported in part by the Austrian Government through the research program COMET (Project SeCoS, Project Number 836628) and through the research program FIT-IT Trust in IT Systems (Project SePAG, Project Number 835919), by the Secure Information Technology Center-Austria (A-SIT), and by the Research Fund KU Leuven, OT/13/071.

References

1. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of Keccak and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
2. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
3. Daemen, J., Rijmen, V.: The Design of Rijndael: AES–The Advanced Encryption Standard. Springer, New York (2002)
4. European Network of Excellence in Cryptology: ECRYPT II SHA-3 Zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
5. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl - a SHA-3 candidate. Submission to NIST (Round 3). January 2011. http://www.groestl.info
6. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
7. Ideguchi, K., Tischhauser, E., Preneel, B.: Improved collision attacks on the reduced-round Grøstl hash function. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 1–16. Springer, Heidelberg (2011)
8. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved rebound attack on the finalist Grøstl. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 110–126. Springer, Heidelberg (2012)
9. Jean, J., Naya-Plasencia, M., Peyrin, T.: Multiple limited-birthday distinguishers and applications. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 533–550. Springer, Heidelberg (2014)

10. Knudsen, L.R.: SMASH—a cryptographic hash function. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 228–242. Springer, Heidelberg (2005)
11. 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. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
12. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The rebound attack and subspace distinguishers: application to whirlpool. Cryptology ePrint Archive, Report 2010/198 (2010). http://eprint.iacr.org/
13. Lamberger, M., Pramstaller, N., Rechberger, C., Rijmen, V.: Second preimages for SMASH. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 101–111. Springer, Heidelberg (2006)
14. 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.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
15. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: cryptanalysis of reduced whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
16. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attacks on the reduced `Grøstl` hash function. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 350–365. Springer, Heidelberg (2010)
17. National Institute of Standards and Technology: FIPS PUB 197: advanced encryption standard. Federal Information Processing Standards Publication 197, U.S. Department of Commerce. November 2001. http://www.itl.nist.gov/fipspubs
18. National Institute of Standards and Technology: Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Fed. Reg. 27(212):62212–62220 (2007). http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
19. National Institute of Standards and Technology: SHA-3 selection announcement, October 2012. http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf
20. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
21. Peyrin, T.: Improved differential attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer, Heidelberg (2010)
22. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-Sbox analysis: applications to ECHO and Grøstl. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 38–55. Springer, Heidelberg (2010)
23. Schläffer, M.: Updated differential analysis of Grøstl (2011). http://www.groestl.info/
24. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
25. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)
26. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)

27. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
28. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
29. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage attack on round-reduced Grøstl hash function and others. In: Canteaut, A. (ed.) Fast Software Encryption. LNCS, vol. 7549, pp. 127–145. Springer, Heidelberg (2012)