Practical Attacks on AES-like Cryptographic Hash Functions

Stefan Kölbl(✉) and Christian Rechberger

Technical University of Denmark, Kongens Lyngby, Denmark
stek@dtu.dk

Abstract. Despite the great interest in rebound attacks on AES-like hash functions since 2009, we report on a rather generic, albeit keyschedule-dependent, algorithmic improvement: A new message modification technique to extend the inbound phase, which even for large internal states makes it possible to drastically reduce the complexity of attacks to very practical values for reduced-round versions. Furthermore, we describe new and practical attacks on Whirlpool and the recently proposed GOST R hash function with one or more of the following properties: more rounds, less time/memory complexity, and more relevant model. To allow for easy verification, we also provide a source-code for them.

Keywords: Hash functions · Cryptanalysis · Collisions · Whirlpool · GOST R · Streebog · Practical attacks

1 Introduction

Cryptographic hash functions are one of the most versatile primitives and have many practical applications like integrity checks, message authentication, digital signature or password protection. Often they are a critical part of more complex systems whose security might fall apart if hash a function does not provide the properties we expect it to have.

Cryptographic hash functions take as input a string of arbitrary finite length and produce a fixed-sized output of n bits called hash. As a consequence, the following main security requirements are defined for cryptographic hash functions:

- **Preimage Resistance:** For a given output y it should be computationally infeasible to find any input x' such that $y = h(x')$.
- **Second Preimage Resistance:** For given $x, y = h(x)$ it should be computationally infeasible to find any $x' \neq x$ such that $y = h(x')$.
- **Collision Resistance:** It should be computationally infeasible to find two distinct inputs x, x' such that $h(x) = h(x')$.

For any ideal hash function with n-bit output size, we can find preimages or second preimages with a complexity of 2^n, and collisions with a complexity of $2^{n/2}$ using generic attacks.

D.F. Aranha and A. Menezes (Eds.): LATINCRYPT 2014, LNCS 8895, pp. 259–273, 2015.
DOI: 10.1007/978-3-319-16295-9_14

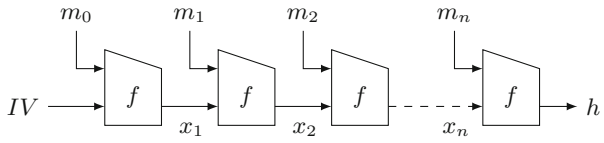

Fig. 1. Iterative construction for a cryptographic hash function.

Most cryptographic hash functions are constructed iteratively by splitting the message into evenly sized blocks m_i and using a compression function f to update the state. We call the intermediate results x_i chaining values and the final output h hash value (Fig. 1).

The security proofs for the hash function rely on the difficulty of finding a collision for this compression function, hence it is also of interest to consider the properties of the compression function and find properties which distinguish it from an ideal function.

- **semi-free start collision:** Find x, m, m' such that $f(x, m) = f(x, m')$.
- **free-start collision:** Find x, x', m, m' such that $f(x, m) = f(x', m')$.
- **near collision:** Find x, m, m' such that $f(x, m) \oplus f(x, m')$ has a low Hamming weight.

To sum up the various types with respect to their relevance: a semi-free-start collision is more interesting than a free-start collision, and a collision is more interesting than a near-collision.

1.1 Motivation

Cryptanalytic attacks are often hard to verify. Cryptanalysts often concentrate on the total running time of the attack, which is boiled down to a single number. While one can argue about the exact transition point between cryptanalytic attacks of practical and theoretical time complexity, it is often placed around an equivalent of 2^{64} calls to the primitive [1]. While this is a reasonable assumption for state-level adversaries, it is out of reach for academic research labs. However, the ability to fully implement and verify attacks is crucial, as this is often the only way to make sure that all details are modelled correctly in the theoretical analysis. In this paper we therefore aim at attacks that can actually be executed (and verified) with limited budget computing resources.

In this paper we show a new practical attack on a class of AES-like hash functions. We show attacks on reduced round versions of the ISO/IEC 10118-3 standard `Whirlpool` [2] and the new Russian federal standard `GOST R 34.11-2012` [3]. The model we consider is semi-free-start attacks on the compression function, which in contrast to the free-start attacks do not allow the attacker to choose different chaining values in a pair of inputs. This reduced degree of freedom makes the task of cryptanalysts harder, but is more relevant as it is closer to the actual use in the hash function.

1.2 Contribution

Despite a lot of attention on rebound-attacks of AES and AES-like primitives, we show that more improvements are possible in the inbound phase.

To the best of our knowledge, currently no practical attacks on reduced round `GOST R` have been published. However, there exists a practical 4-round free-start collision attack on the `Whirlpool` compression function [4]. It seems very hard to apply this specific attack directly to `GOST R` due to the extra round in the key schedule, which gives `GOST R` additional security against these free-start attacks.

In this paper we show a new method to carry out a 4-round practical attack on the `Whirlpool` and `GOST R` compression function. Additionally, and in contrast to many other attacks known on `GOST R`, we do not need the freedom to add half a round at the end to turn a near-collision into a collision. As the full hash function also does not end with a half round, we argue that a result on 4 rounds can actually be more informative than a result on 4.5 rounds.

New message modification technique. The attack is based on the rebound attack and start-in-the-middle techniques, and it carefully chooses the key input to significantly reduce the complexity resulting in a very low complexity[1]. We are also able to improve the results on 6.5 rounds by extending this attack. We give an actual example for such a collision, and have the source code of both the attack and the general framework publicly available to facilitate further research on practical attacks[2]. The method is not specific to a particular primitive, but is an algorithmic technique that however depends on two conditions in a primitive to hold (see also Sect. 4).

1.3 Related Work

In Table 1 we summarize the practical results on `Whirlpool` and `GOST R`. As the `GOST R` compression function uses a design similar to the `Whirlpool` hash function [2], many of the previous results on `Whirlpool` can be applied to `GOST R`. We would also like to note on adding half a round at the end for `GOST R`. This does not always make an attack more difficult, and in some cases it makes it easier, as it makes it possible to turn a near-collision into a collision, therefore we distinguish for our attacks if it applies for both cases.

There have also been practical attacks on other AES-based hash functions like `Maelstroem` (6 out of 10 rounds [7]), `Grøstl` (6 out of 10 rounds [8]) and `Whirlwind` (4.5 out of 12 rounds [9]).

1.4 Rebound Attacks

The rebound attack is a powerful tool in the cryptanalysis of hash functions, especially for finding collisions for AES-based designs [10,11]. The cipher is split

[1] Naturally, the improvement is not applicable for constructions or modes that do not allow modification of the key input.

[2] The source-code can be found at https://github.com/kste/aeshash.

Table 1. Summary of attacks with a complexity up to 2^{64} on AES-based hash functions. Time is given in compression function calls and memory in bytes.

Function	Rounds	Time	Memory	Type	Reference
`GOST R`	4.5	2^{64}	2^{16}	semi-free-start collision	[5]
	4.75	practical	2^{8}	semi-free-start near-collision	[6]
	4	$2^{19.8}$	2^{16}	semi-free-start collision	this work
	4.5	$2^{19.8}$	2^{16}	semi-free-start collision	this work
	5.5	2^{64}	2^{64}	semi-free-start collision	[5]
	6.5	2^{64}	2^{16}	semi-free-start collision	this work
`Whirlpool`	4	$2^{25.1}$	2^{16}	semi-free-start collision	this work
	6.5	$2^{25.1}$	2^{16}	semi-free-start near-collision	this work
	4	2^{8}	2^{8}	free-start collision	[5]
	7	2^{64}	2^{8}	free-start collision	[4]

into three sub-ciphers

$$E = E_{fw} \circ E_{in} \circ E_{bw}$$

and the attack proceeds in two steps. First, the inbound phase which is an efficient meet-in-middle in E_{in} using the available degree of freedom. This is followed by a probabilistic part, the outbound phase in E_{fw} and E_{bw} using the solutions from the inbound phase. The basic 4-round rebound attack uses a differential characteristic with $1-8-64-8-1$ active bytes per round and has a complexity of 2^{64}. There are many techniques extending and improving this attack. Some can even improve this very basic and simple setting of a square geometry, like start-from-the-middle [8], super S-Box [12,13] or solving three fully active states in the middle [14,15]. Other generic extensions exploit additional degrees of freedom or non-square geometries to improve results, like and using multiple inbounds [12,16]. In these settings, improved list-matching techniques [17,18] are also a generic improvement.

2 Description of `GOST R`

This section gives a short description of the `GOST R` compression function as we will use it for describing our attack in detail. As we are only looking at the compression function, we leave out some details not relevant for the upcoming attack in order to simplify the description. For a more detailed description of `GOST R` we refer to [3].

The compression function g uses two 512-bit inputs (the message block m and the chaining value h) to update the state in the following way (see Fig. 2)

$$g_N(h, m) = E(L \circ P \circ S(h), m) \oplus h \oplus m \tag{1}$$

where E is an AES-based block cipher using a state of 8×8 bytes and S, P, L are the same functions as used in this block cipher (see below).

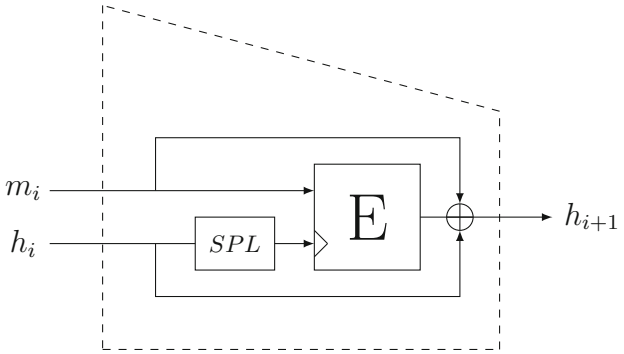

Fig. 2. An outline of the `GOST R` compression function. The chaining input is processed through an additional round before entering E

If we want to find a collision for the compression function, the following equation must hold

$$\Delta m_i \oplus \Delta h_i \oplus \Delta E(h_i, m_i) = 0 \tag{2}$$

2.1 Block Cipher E

The block cipher E takes two 512-bit inputs M and K^0 and produces a 512-bit output C. The state update consists of 12 rounds r and a final key addition.

$$\begin{aligned} L_1 &= L \circ P \circ S \circ AK(M, K^0) \\ L_{i+1} &= L \circ P \circ S \circ AK(L^i, K^i) \quad i = 1 \ldots 11 \\ C &= AK(L^{12}, K^{12}) \end{aligned}$$

The following four operations are used in one round (see Fig. 3):

- **AK** Adds the key byte-wise by XORing it to the state.
- **S** Substitutes each byte of the state independently using an 8-bit S-Box.
- **P** Transposes the state.
- **L** Multiplies each row by an 8×8 MDS matrix.

The 512-bit key input is expanded to 13 subkeys $K_0, \ldots, K_{12}$. This is done similar to the state update but AK is replaced with the addition of a round-dependent constant RC^r.

$$\begin{aligned} L_{i+1} &= L \circ P \circ S \circ AK(K^0, RC^0) \quad i = 0 \ldots 11 \\ K^{12} &= AK(L^{12}, K^{12}) \end{aligned}$$

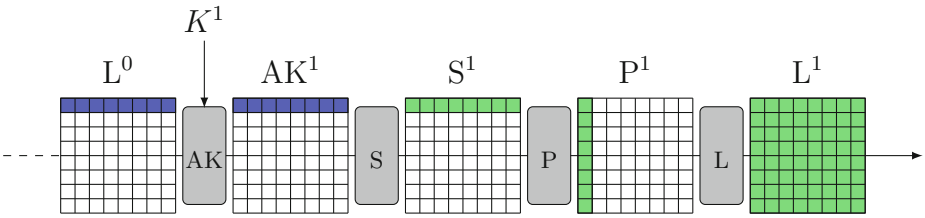

Fig. 3. The four operations used in one round of `GOST R`.

2.2 Notation

The notation we use for naming the states is:

- The state after applying the round function $\{AK, S, P, L\}$ in round r is named $\{AK^r, S^r, P^r, L^r\}$
- The byte at position x, y in state X^r is named $X^r_{x,y}$
- A row is denoted by $X^r_{*,y}$ and a column by $X^r_{x,*}$
- ■ and ■ denote that there is a difference in a byte.
- ■ and ■ are used for highlighting values of a byte.

2.3 Differential Properties

The attacks in this paper are based on differential cryptanalysis, and the resulting complexity correlates with the differential properties of the round functions. Therefore, to ease understanding, we give a short overview of the properties that are relevant for our attack.

The linear layer L has a strong influence on the number of active S-Boxes. There is no proof given that the linear layer L is MDS or has a branch number of 9 in the `GOST R` reference [3], but it was shown that this is the case in [19]. Hence, if we have one active byte at the input we will get 8 active bytes at the output with probability one. If we have a active bytes at the input the probability that there will be b active bytes at the output under the condition $a \neq 0, b \neq 0$ and $a + b \geq 9$ is $2^{(b-8)8}$.

The properties of the S-Box have a strong influence on the complexity of our attack, as will be seen later. Given a S-Box $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$

$$\{x \mid S(x) \oplus S(x \oplus a) = b\} \tag{3}$$

is the number of solutions for an input a and output difference b. Table 2 gives the number of solutions for some S-Box designs used in practice.

To get a bound on the probability of the differential characteristics we are interested in the maximum value of Eq. 3 which we will refer to as the *maximum differential probability* (mdp) of an S-Box. A 4-round differential characteristic has at least 81 active bytes due to the properties of the linear layer, therefore any 4-round characteristic has a probability of $\leq \text{mdp}^{81}$.

Table 2. Comparison of different 8-bit S-Box designs used in AES-based hash functions.

Solutions	AES	Whirlpool	GOST R
0	33150	39655	38235
2	32130	20018	22454
4	255	5043	4377
6	-	740	444
8	-	79	25
256	1	1	1

For the rebound attack it is also important to know the average number of possible output differences, given a non-zero input difference. We will refer to this as the average number of solutions (ANS) for an S-Box which can be computed by constructing the *differential distribution table* (DDT). The ANS corresponds to the average number of non-zero entries in each row of the DDT.

This property influences the complexity for the matching step in the inbound phase and increases the costs of finding one solution. For the `GOST R` S-Box we get on average 107.05 solutions.

3 Attack on GOST R

In this section we describe our 4-round practical attack in detail and also show how it can be applied to more rounds. The description of the attack is split into two parts. First, we find a differential characteristic leading to a collision. Then we show how to construct a message pair following this characteristic in a very efficient way.

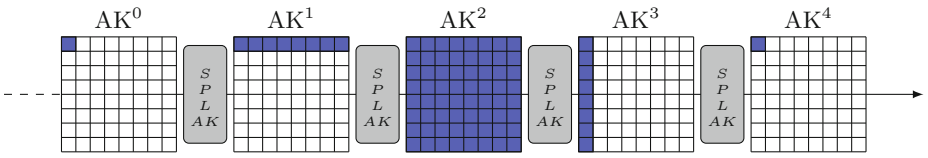

Fig. 4. The 4-round differential characteristic used in our attack.

3.1 Constructing the Differential Characteristic

For our 4-round attack we use a characteristic of the form $1 - 8 - 64 - 8 - 1$ (see Fig. 4). This truncated differential path has the minimal number of possible active S-Boxes for 4 rounds and is the starting point for many attacks. Next, we will determine the values of the differences before continuing with the construction of the message pair.

The approach we use for this is based on techniques from the rebound attack, like the start-in-the-middle technique used in [8]. This approach would also give us an efficient way to find both the characteristic and message pair for a characteristic of the form $1 - 8 - 64 - 8$. However this would still lead to a higher attack complexity if extended to 4 rounds. Hence, we only use ideas from this approach to determine the differential characteristic and do not assume the key input as constant.

Precomputation. First we pre-compute the differential distribution table (DDT) of the S-Box and we also construct a list M_{lin}. This list contains all possible 255 non-zero values of $P_{0,0}$ and the result after applying L (see Fig. 5).

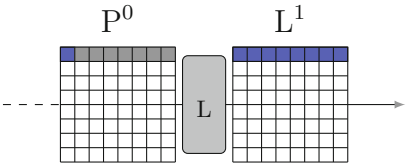

Fig. 5. Computing list M_{lin} for all 255 values of $P^0_{0,0}$ (blue) to find all possible transitions from 1 to 8 bytes. Gray bytes are set to zero (Color figure online).

Construction

1. Start with a random difference in $AK^4_{0,0}$ and propagate it back to S^2 through the inverse round functions. For the linear steps this is deterministic, and for propagating through the S-Box we choose a random possible input difference to the given output difference. After this step we will have a full active state in S^2.
2. For each difference in S^2 we look up the set of all possible input differences from the DDT for each byte of the state.
3. Check for each row of AK^2 whether there is a possible match with the rows stored in M_{lin} (see Fig. 6).
   - The probability that a single byte matches is $107.05/255 \approx 2^{-1.252}$ therefore a row matches with a probability of $2^{-10.018}$.
   - If we take into account that M_{lin} has 255 entries we expect to find a match with a probability of $1 - (1 - 2^{-10.018})^{255} \approx 2^{-2.2}$.
   - Therefore the probability for a match of all 8 rows is given by

$$(2^{-2.2})^8 = 2^{-17.6} \tag{4}$$

After this step we have found a characteristic spanning from S^1 to AK^4. Now we have to repeat the previous process for a single row to find the right differences in AK^1. This has a probability of $2^{-2.2}$ of succeeding. Hence we need to repeat the whole process $2^{19.8}$ times to obtain one solution.

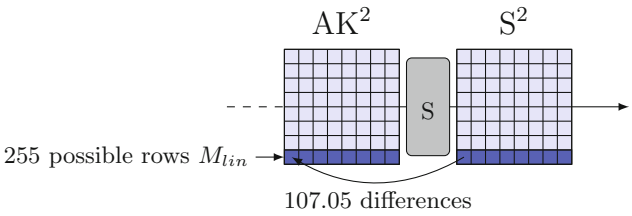

Fig. 6. The matching step in the middle is done on each row individually. There are 2^8 possible values for each row $AK^2_{*,j}$ for $j = 0, 1, \ldots, 7$.

Note that we can only choose 255 differences for $AK^4_{0,0}$, but we can also freely choose from the set of possible differences when propagating from S^3 to AK^3. This gives us an additional 107.05 choices for each row in S^2 leading to $\approx 2^{54}$ possible values for the state S^2. Hence, we have enough starting points for finding our differential characteristic.

3.2 Finding the Message Pair

Now we want to find a message pair which follows the previously constructed characteristic. At this point only the differences, but not the values of the state, are fixed. We start by fixing the values of AK^2 such that the 64 differential transitions $S^2 = S(AK^2)$ are fulfilled.

Next we use the key input to solve any further conditions on the active S-Boxes in order to lower the complexity of our attack. This step is split into solving the conditions on $S^1_{*,0} = S(AK^1_{*,0})$ and $S^3_{0,*} = S(AK^3_{0,*})$.

Solving Conditions at the Start. We have 8 conditions on $S^1_{*,0}$ which we need to solve. These conditions can be solved row-wise by choosing the corresponding values in K^2 such that $P^{-1}(L^{-1}(AK^2 \oplus K^2)) = S^1$. We can do this step row-wise by solving a linear equation. As there is only a single byte condition for

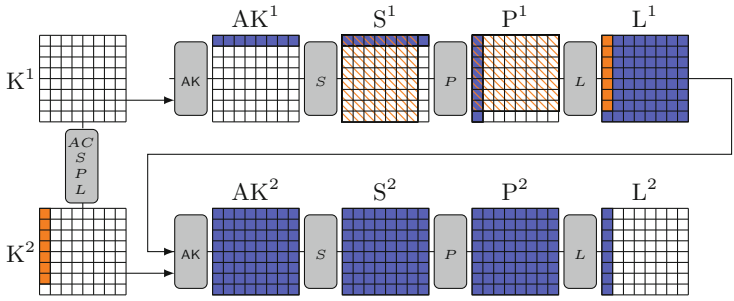

Fig. 7. The values of AK^2 are fixed. We solve 7 of the conditions on S^1 by using the freedom in K^2 (bytes marked orange), which allows us to influence the values on the bytes in S^1 (orange slash pattern).

each row, we only need one byte in the corresponding row of K^2 to solve the equation (see Fig. 7). The remaining bytes are fixed to arbitrary values as we have no further conditions to fulfill at this step. These bytes could be used to solve more conditions for other differential characteristics or to construct additional solutions, as we will do for extending the attack on more rounds.

In this step we can generate up to 2^{56} solutions per row. Note that we only do this step for 7 rows, as we need the last row in the next step.

Solving Conditions at the End. For solving the conditions $S^3 = S(AK^3)$, we can use the bytes in $K^2_{*,7}$. These bytes form a column in $\mathsf{KP}^3_{7,*}$ (see Fig. 8), which allows us to solve a single byte condition per row for AK^3.

1. Assume that $K^2_{*,0-6}$ are fixed and propagate them forward to KP^3.
2. We can now solve the conditions for each row individually. In each row there are 7 bytes fixed in KP^3 and a single byte in K^3 (from AK^3). This gives us a linear equation with one solution per row and allows us to solve all conditions on AK^3.

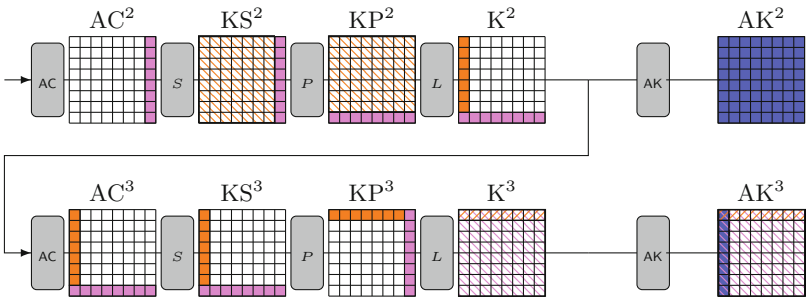

Fig. 8. Solving all the conditions on AK^3. The orange values are fixed from the previous step and the purple values are used to fulfill the conditions on AK^3.

Remaining Conditions. We still need to solve one byte condition on $S^1_{0,7}$, which can be done by repeating the previous procedure 2^8 times. The bytes which are used to solve the conditions on AK^3 form a row in K^2 and influence the values of L^1 resp. P^1 and S^1 (see Fig. 11 in Appendix A). This implies that we can change the value of $S^1_{0,7}$ by constructing different solutions for $K^2_{*,7}$.

The only remaining condition is $\Delta AK^0_{0,0} = \Delta AK^4_{0,0}$, which can again be solved by repeating the previous steps 2^8 times. It follows that we need to repeat the algorithm shown in Sect. 3.2 about 2^{16} times.

Complexity. We can construct the differential characteristic with a complexity of $2^{19.8}$. Finding a message pair following this characteristic requires 2^{16} steps using our message modification technique. Hence, the total complexity of the attack is $\approx 2^{19.9}$. We have implemented this attack and verified our results. The un-optimized proof-of-concept implementation in Python is publily available [20]. An example for a 4-round collision can be found in Appendix B.

3.3 Extending the Attack

As we only need to control 15 bytes of the key, we can extend the attack on 6.5 rounds by using a characteristic of the form $8-1-8-64-8-1-8$. In this case we would use the same approach to find the differential characteristic for 4 rounds and in the message modification part we would construct more solutions by using the additional freedom in the key. This will influence the differences at the input/output of the 6.5 rounds. The complexity of this attack is $\approx 2^{64}$, as the 8-byte difference at the input/output needs to be equal (Fig. 9).

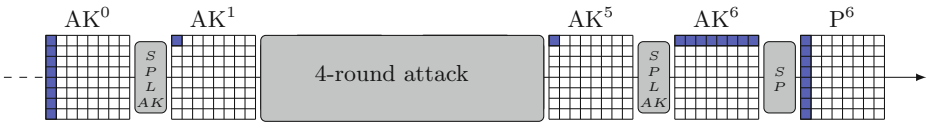

Fig. 9. The 4-round attack is extended by one round in the beginning and one round in the end to mount an attack on 6.5 rounds.

4 Application to Other AES-based Hash Functions

The message modification technique presented is not specific to `GOST R`, but requires a few criteria to be met. First the transposition layer has to have the property that every byte of a single row/column is moved to a different row/column (see Fig. 10). This is true for all AES-based hash functions we consider in this paper, as it is a desired property against other attacks.

The second criteria is that there is a key addition in every round, hence our attack is applicable to both `Whirlpool` and `GOST R`. Permutation-based designs like `Grøstl` do not have this property. The attacker has less control of the input for each round, which makes the hash function more resistant against these types of attacks.

The complexity of the attack depends on the choice of the S-Box, as this directly influences the costs of constructing the differential characteristic. Given the average number of solutions $\overline{s}$ for $\Delta out = S(\Delta in)$ with a fixed value Δin, this directly gives the complexity for the matching step of the attack

$$\left(1-\left(1-\frac{\overline{s}}{255}\right)^{255}\right)^{8} \tag{5}$$

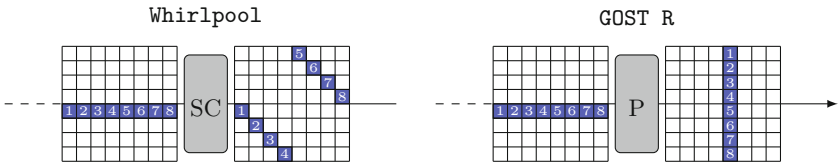

Fig. 10. The transposition layer used in `Whirlpool` and `GOST R`.

Table 3. Comparing the maximum differential probability (MDP) and average number of solutions (ANS) for different 8-bit S-Boxes in AES-based designs.

S-Box	MDP	ANS	Matching Costs	$\#S^2$
AES	2^{-6}	127	$2^{6.42}$	$2^{55.91}$
Whirlpool	2^{-5}	101.49	$2^{25.10}$	$2^{53.32}$
GOST-R	2^{-5}	107.05	$2^{19.77}$	$2^{53.94}$

and the number of possible states for S^2 is $\approx \overline{s}^{\,8}$. A comparison of the different S-Boxes used in AES-based hash functions is given in Table 3.

5 Conclusion

In this paper, we have shown new practical attacks for both the `Whirlpool` and `GOST R` compression function. We presented a 4-round attack with very low complexity of $2^{25.10}$ resp. $2^{19.8}$. Importantly, the attack is fully verified and source-code for it is available. In the case of `GOST R` the attack can be extended to find collisions for 6.5 rounds with a complexity of 2^{64} and for `Whirlpool` we can extend it to construct a near-collision in 50 bytes with a complexity of $2^{25.10}$ for 6.5 rounds of the compression function. The difference in the results for `GOST R` and Whirlpool is due to the ShiftColumns operation which does not align the bytes to lead to a collision for the differential characteristic we use.

Our attack is applicable to all AES-based primitives where it is possible for the attacker to control the key input for a few rounds. This significantly reduces the complexity of previous attacks and might be useful to speed up other attacks on AES-based hash-function designs.

References

1. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 299–319. Springer, Heidelberg (2010)
2. Barreto, P., Rijmen, V.: The Whirlpool hashing function. In: First open NESSIE Workshop, Leuven, Belgium, vol. 13, p. 14 (2000)
3. Dolmatov, V., Degtyarev, A.: GOST R 34.11-2012: Hash Function (2013). http://tools.ietf.org/html/rfc6986
4. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating fundamental security requirements on whirlpool: improved preimage and collision attacks. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 562–579. Springer, Heidelberg (2012)
5. Wang, Z., Yu, H., Wang, X.: Cryptanalysis of GOST R Hash Function. Cryptology ePrint Archive, Report 2013/584 (2013). http://eprint.iacr.org/

6. AlTawy, R., Kircanski, A., Youssef, A.M.: Rebound Attacks on Stribog. Cryptology ePrint Archive, Report 2013/539 (2013). http://eprint.iacr.org/
7. Kölbl, S., Mendel, F.: Practical attacks on the maelstrom-0 compression function. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 449–461. Springer, Heidelberg (2011)
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.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
9. Barreto, P.S.L.M., Nikov, V., Nikova, S., Rijmen, V., Tischhauser, E.: Whirlwind: a new cryptoaphic hash function. Des. Codes Crypt. **56**(2–3), 141–162 (2010)
10. 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)
11. Lamberger, M., Mendel, F., Schläffer, M., Rechberger, C., Rijmen, V.: The rebound attack and subspace distinguishers: application to whirlpool. J. Cryptol., 1–40 (2013)
12. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. [21] 126–143
13. 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)
14. 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)
15. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved cryptanalysis of AES-like permutations. J. Cryptology **27**(4), 772–798 (2014)
16. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. [21] 106–125
17. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 719–740. Springer, Heidelberg (2012)
18. Naya-Plasencia, M.: How to improve rebound attacks. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer, Heidelberg (2011)
19. Kazymyrov, O., Kazymyrova, V.: Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012. Cryptology ePrint Archive, Report 2013/556 (2013). http://eprint.iacr.org/
20. https://github.com/kste/aeshash
21. Matsui, M. (ed.): ASIACRYPT 2009. LNCS, vol. 5912. Springer, Heidelberg (2009)

A Solving Conditions

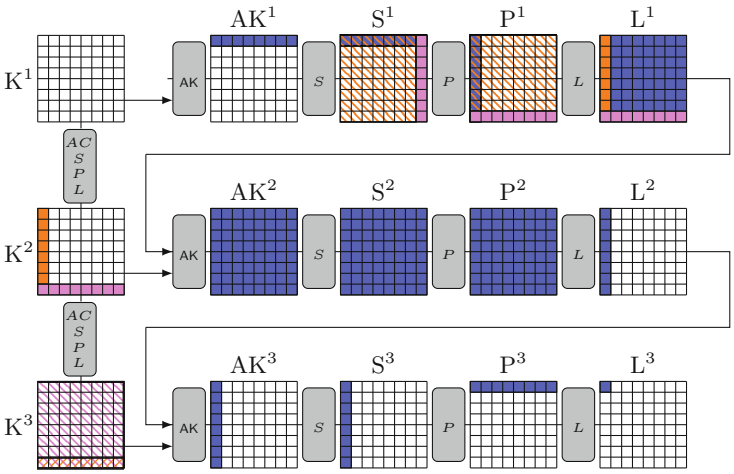

Fig. 11. Solving both conditions on S^1 and AK^3. The bytes marked purple solve the conditions on AK^3 and a single condition on S^1, whereas the orange bytes solve 7 conditions on S^1.

B Colliding Message Pair

Here a colliding message pair (M, M') and the chaining value are given. The message pair has been found by using the 4-round characteristic and the difference in the messages is $\Delta AK^0_{0,0} = \Delta AK^4_{0,0} = fc$. All values are given in hexadecimal notation.

K^0

```
e80c313d6875c049
865604acdb024055
5cfd18c5d1a5a3f3
793359475ae90836
528e80e4aedfd674
20c777c0306ff6da
da4f9acc1e4fc9c4
04f486b91fa6bdeb
```

AK^0	AK'^0	ΔAK^0
81c2cf897a89e94f	7dc2cf897a89e94f	fc00000000000000
11e6c3ac6020a3c9	11e6c3ac6020a3c9	0000000000000000
ef44c4305cfade20	ef44c4305cfade20	0000000000000000
c04dc08f87db9f8f	c04dc08f87db9f8f	0000000000000000
b56983ec66229993	b56983ec66229993	0000000000000000
1bf0262cb3b92956	1bf0262cb3b92956	0000000000000000
dcb1e1511414ac83	dcb1e1511414ac83	0000000000000000
8efcd0ef76fa4671	8efcd0ef76fa4671	0000000000000000

AK^4	AK'^4	ΔAK^4
1444aab6eb146b46	e844aab6eb146b46	fc00000000000000
0692f61cf6b43022	0692f61cf6b43022	0000000000000000
935d2a8720ae7f0f	935d2a8720ae7f0f	0000000000000000
329542c23a812da1	329542c23a812da1	0000000000000000
eca3ba2984ab0219	eca3ba2984ab0219	0000000000000000
636b834737911049	636b834737911049	0000000000000000
0ce0d7edefe72c55	0ce0d7edefe72c55	0000000000000000
6dc8a467a81f8587	6dc8a467a81f8587	0000000000000000