Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function

Jérémy Jean[⋆] and Pierre-Alain Fouque

Ecole Normale Supérieure
45 rue d'Ulm – 75230 Paris Cedex 05 – France
{Jeremy.Jean, Pierre-Alain.Fouque}@ens.fr

Abstract. In this paper, we present new results on the second-round SHA-3 candidate ECHO. We describe a method to construct a collision in the compression function of ECHO-256 reduced to four rounds in 2^{52} operations on AES-columns without significant memory requirements. Our attack uses the most recent analyses on ECHO, in particular the **SuperSBox** and **SuperMixColumns** layers to utilize efficiently the available freedom degrees. We also show why some of these results are flawed and we propose a solution to fix them. Our work improves the time and memory complexity of previous known techniques by using available freedom degrees more precisely. Finally, we validate our work by an implementation leading to near-collisions in 2^{36} operations for the 4-round compression function.

Keywords: Cryptanalysis, Hash Functions, SHA-3, ECHO-256, Collision attack

1 Introduction

Recently, the National Institute of Standards and Technology (NIST) initiated an international public competition aiming at selecting a new hash function design [12]. Indeed, the current cryptanalysis of hash functions like SHA-1 and MD5 show serious weaknesses [18,19,20,21]. To study hash functions, one of the most powerful strategy is the differential cryptanalysis, which was introduced in [2] by Biham and Shamir to study the security of block ciphers. It consists in following the evolution of a message pair in the cipher by looking at the differences between the messages while they propagate through the encryption process. This type of analysis is particularly useful for studying hash functions where no secret-key is involved: in this known-key model [5], the attacker can thus follow the message pair at each step of the process. Knudsen generalized the idea in [4] with the concept of *truncated differentials*, aiming at following the *presence* of differences in a word, rather than their actual values. Initiated by the work of Peyrin on Grindhal [13], this kind of analysis leads to many other successful attacks against

[⋆] This author was partially supported by the French ANR project SAPHIR II and the French *Délégation Générale pour l'Armement* (DGA).

block ciphers and hash functions, in particular those based on the AES [6,10] like ECHO. For the AES, since all differences are equivalent, only their presence matters.

Thanks to the SHA-3 contest, new kinds of attacks for AES-based permutations have been suggested in the past few years, in particular the rebound attack [10] and the start-from-the-middle attack [9]. In both cases, the novelty is to start searching for a message pair conforming to a given differential path in the middle of the trail. Doing so, we have the freedom of choosing values and differences where they can reduce greatly the overall cost of the trail.

The rebound technique uses these degrees of freedom to fulfill the most expensive part of the trail at very low average complexity whereas the remaining of the path is verified probabilistically. The number of controlled rounds in that case can not exceed two rounds. The start-from-the-middle technique improves the rebound attack in the sense that it uses the independence in the search process as much as possible. Consequently, it extends the number of controlled rounds to three, without any extra time.

In the present case of ECHO, Schläffer uses in [17] the idea of multiple inbound phases on two different parts of the whole path. Similar techniques have been introduced on Whirlpool [6] and on the SHA-3 proposal LANE [8]. In comparison to the rebound or the start-from-the-middle techniques, we are not limited to a controlled part located in the middle of the path. In the end, the partial message pairs are merged using remaining degrees of freedom. Schläffer's nice attacks permute some linear transformations of the ECHO round function to introduce the **SuperMixColumns** layer, which relies on a large matrix presenting non-optimal diffusion properties. It thus allows to build sparser truncated differential. In this paper, we show that the latest analyses of ECHO made by Schläffer fail with high probability at some point of the merging process: the attacks actually succeed with probability 2^{-128}. Nevertheless, we suggest an attack using degrees of freedom slightly differently to construct collisions and near-collisions in the compression function of ECHO-256 reduced to four rounds.

Our new techniques improve the rebound attack by using freedom degrees more precisely to get and solve systems of linear equations in order to reduce the overall time and memory complexity. We also describe a similar method as the one described by Sasaki et al. in [16] to efficiently find a message pair conforming to a truncated differential through the **SuperSBox** when not all input or output bytes are active. Both new techniques allow to repair some of the Schläffer's results to construct collisions in the compression function of ECHO-256. To check the validity of our work, we implement the attack to get a semi-free-start near-collisions in 2^{36} computations. That is, a chaining value h and a message pair (m, m') colliding on 384 bits out of 512 in the compression function f reduced to four rounds: $f(h, m) =_{384} f(h, m')$.

We summarize our results in Table 1.

The paper is organized as follows. In Section 2, we quickly recall the specifications of the ECHO hash function and the permutation used in the AES. In Section 3, we describe the differential path we use and present an overview of

Table 1. Summary of results detailed in this paper and previous analyses of ECHO-256 compression function. We measure the time complexity of our results in terms of operations on AES-columns. The notation $n/512$ describe the number n of bits on which the message pair collides in the near-collisions. Result from [17] have not been printed since flawed.

Rounds	Time	Memory	Type	Reference
3	2^{64}	2^{32}	free-start collision	[14]
3	2^{96}	2^{32}	semi-free-start collision *	[14]
4.5	2^{96}	2^{32}	distinguisher	[14]
4	2^{36}	2^{16}	semi-free-start near-collision 384/512	**This paper**
4	2^{44}	2^{16}	semi-free-start near-collision 480/512 †	**This paper**
4	2^{52}	2^{16}	semi-free-start collision	**This paper**

* With chosen salt

† This result is an example of other near-collisions that can be derived from the attack of this paper.

the differential attack to find a message pair conforming to this path. Then, in Section 4, we present the collision attack of ECHO-256 compression function reduced to four rounds. Finally, we conclude in Section 5. We validate our results by implementing the near-collision attack.

2 Description of ECHO

The hash function ECHO updates an internal state described by a 16×16 matrix of GF (2^8) elements, which can also be viewed as a 4×4 matrix of 16 AES states. Transformations on this large 2048-bit state are very similar to the one of the AES, the main difference being the equivalent S-Box called **BigSubBytes**, which consists in two AES rounds. The diffusion of the AES states in ECHO is ensured by two *big* transformations: **BigShiftRows** and **BigMixColumns** (Figure 1).

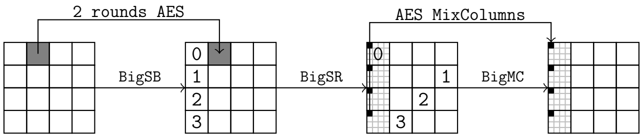


Fig. 1. One round of the ECHO permutation. Each of the 16 cells is an AES state (128 bits).

At the end of the 8 rounds of the permutation in the case of ECHO-256, the **BigFinal** operation adds the current state to the initial one (feed-forward) and adds its four columns together to produce the new chaining value. In this paper, we only focus on ECHO-256 and refer to the original publication [1] for more details on both ECHO-256 and ECHO-512 versions. Note that the keys used in the two AES rounds are an internal counter and the salt, respectively: they are mainly introduced to break the existing symmetries of the AES unkeyed permutation [7]. Since we are not using any property relying on symmetry and that adding constants does not change differences, we omit these steps.

Two versions of the hash function ECHO have been submitted to the SHA-3 contest: ECHO-256 and ECHO-512, which share the same state size, but inject messages of size 1536 or 1024 bits respectively in the compression function. Focusing on ECHO-256 and denoting f its compression function, H_i the i-th output chaining value, $M_i = M_i^0 \,||\, M_i^1 \,||\, M_i^2$ the i-th message block composed of three chunks of 512 bits each M_i^j and $S = [C_0 C_1 C_2 C_3]$ the four 512-bit ECHO-columns constituting state S, we have ($H_0 = IV$):

$$C_0 \leftarrow H_{i-1} \qquad C_1 \leftarrow M_i^0 \qquad C_2 \leftarrow M_i^1 \qquad C_3 \leftarrow M_i^2$$

AES. We recall briefly one AES round on Figure 2 and refer as well to original publication [11] for further details. The **MixColumns** layer implements a Maximum Distance Separable (MDS) code that ensures a complete diffusion after two rounds. It has good diffusion properties since its branch number, i.e. the sum of input and output active bytes, is always 0 or greater or equal than 5. As for the AES S-Box, it satisfies an interesting differential property: namely, a random differential transition exists with probability approximately 1/2. By enumerating each input/output difference pair, this result can be computed and stored in 2^{16} in the difference distribution table Δ. At the position (δ_i, δ_o), this table contains a boolean value whether the differential transition $\delta_i \rightarrow \delta_o$ exists. That is, if the equality $S(\lambda) + S(\lambda + \delta_i) = \delta_o$ holds for at least one element $\lambda \in \mathrm{GF}\left(2^8\right)$, S being the AES S-Box. We note that this table can be slightly enlarged to 2^{19} to store one solution when possible.

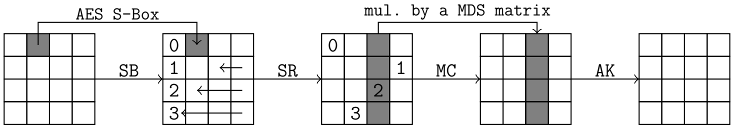


Fig. 2. One round of the AES permutation is the succession of four transformations: SubBytes (**SB**), ShiftRows (**SR**), MixColumns (**MC**) and AddKey (**AK**). Each of the 16 cells is an element of GF $\left(2^8\right)$ (8 bits).

Notations. Throughout this paper, we name each state of the ECHO permutation after each elementary transformation: starting from the first state S0, we end the first round after 8 transformations[1] in S8 and the four rounds in S32. Moreover, for a given ECHO-state Sn, we refer to the AES-state at row i and column j by S$n[i, j]$. Additionally, we introduce *column-slice* or *slice* to refer to a thin column of size 16×1 of the ECHO state. We use *ECHO-column* or simply *column* to designate a column of ECHO, that is a column of four AES states. Similarly, *ECHO-row* or *row* refer to a row of the ECHO state; that is, four AES states.

3 Differential attack for hash functions

To mount a differential attack on a hash function, we proceed in two steps. First, we need to find a good differential path, in the sense that, being probabilistic, it should hold with a probability as high as possible. In the particular case of AES-based hash functions, this generally means a path with a minimum number of active S-Boxes. In comparison with the differential attacks where fixed differences chosen for their high probability go along with the differential path, for this particular design, all differences behave equivalently. Thus, the path is actually a truncated path, precising only whether a difference exists or not.

Second, we have to find a pair of messages following that differential path, which fixes values and differences. In the sequel, we present an equivalent description of the ECHO-permutation and then detail our choice of differential path, using the new round description. The part of the attack that finds a valid message pair for this path using the equivalent description is detailed in Section 3.3.

3.1 Reordering of transformations in the ECHO permutation

SuperSBox. The concept of **SuperSBox** was independently introduced by Lamberger et al. in [6] and by Gilbert and Peyrin in [3] to study two AES rounds. By bringing the two non-linear layers together, this concept is useful to find a message pair conforming to a given differential path and leads to a new kind of cryptanalysis. The design of one AES round describes the sequence **SB-SR-MC** of transformations[2], but we can use the independence of bytes to reorder this sequence. Namely, dealing with the non-linear **BigSubBytes** layer of ECHO, we can permute the first **ShiftRows** with the first **SubBytes** without affecting the final result of the computation. We then glue the two non-linear layers into a unique **SB-MC-SB** non-linear transformation of the permutation. The so-called **SuperSBox** transformation is then viewed as a single non-linear layer operating in parallel on 32-bit AES-columns.

SuperMixColumns. In a similar way, by permuting the **BigShiftRows** transformation with the parallel **MixColumns** transformations of the second AES

[1] Transformations are: **SR** - **SB** - **MC** - **SB** - **SR** - **BSR** - **MC** - **BMC**.

[2] While omitting the key adding.

round, a new *super* linear operation has been introduced by Schläffer in [17], which works on column-slices of size 16 × 1.

This super transformation called **SuperMixColumns** results of 16 parallel applications of **MixColumns** followed by the equivalent in ECHO, that is **BigMixColumns**. This *super* transformation is useful for building particular sparse truncated differential. The matrix of the **SuperMixColumns** transformation is defined as the Kronecker product (or tensor product) of $\mathbf{M}$ with itself, $\mathbf{M}$ being the matrix of the **MixColumns** operation in the AES: $\mathbf{M}_{SMC} = \mathbf{M} \otimes \mathbf{M}$. Schläffer noted in [17] (in Section 3.3) that $\mathbf{M}_{SMC}$ is not a MDS matrix and its branch number is only 8, and not 17.

From this observation, it is possible to build sparse truncated differentials (Figure 3) where there are only 4 active bytes in both input and output slices of the transformation. The path 4 → 16 → 4 holds with probability 2^{-24}, which reduces to 2^{8} the number of valid differentials, among the 2^{32} existing ones. For a given position of output active bytes, valid differentials are actually in a subspace of dimension one. In particular, for slice s, $s \in \{0, 4, 8, 12\}$, to follow the truncated differential 4 → 16 → 4 of Figure 3, we need to pick each slice of differences in the one-dimensional subspace generated by the vector v_s, where:

$$v_0 = [\mathrm{E000\ 9000\ D000\ B000}]^{\mathrm{T}} \qquad v_4 = [\mathrm{B000\ E000\ 9000\ D000}]^{\mathrm{T}}$$
$$v_8 = [\mathrm{D000\ B000\ E000\ 9000}]^{\mathrm{T}} \qquad v_{12} = [\mathrm{9000\ D000\ B000\ E000}]^{\mathrm{T}}$$

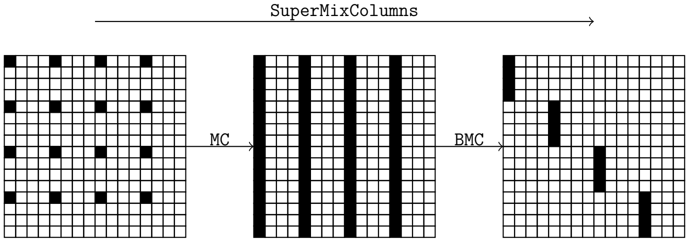


Fig. 3. The **SuperMixColumns** layer in the particular case of the truncated differential 4 → 16 → 4.

This new approach of the combined linear layers allows to build sparser truncated differentials but caused erroneous conclusions when it was used in [17] (in Section 4.1). Namely, at the end of the attack, where two partial solutions need to be merged to get a solution for the whole differential path, everything relies on this critical transformation: we need to solve 16 linear systems. We detail more precisely the problem in Section 3.4, where we study the merge process.

3.2 Truncated differential path

As in the more recent analyses of ECHO [15,17], we consider the path at the byte-level: this allows to build paths sparser than the ones we could obtain by considering only the AES-state level [1,3,9]. Our path is mostly borrowed from [17] and counts 418 active S-Boxes for the ECHO-permutation reduced to four rounds. In comparison to the path from [17], we increase significantly the number of active S-Boxes in the first round to decrease the time complexity of the attack. We note that the number of active S-Boxes is not directly correlated with the complexity of the attack. Moreover, in that case of an AES-based permutation, we can consider a truncated differential path because the actual differences are not really important since they are all equivalent: only their *presence* matters.

Figure 4 presents the truncated differential path used in this attack on the compression function reduced to four rounds. The attack being quite technical, colors have been used in order to improve the reader's understanding of the attack.

3.3 Finding a message pair conforming to the differential path

Strategy. To find a message pair that follows the differential path of Figure 4, our attack splits the whole path into two distinct parts and merges them at the end. In the sequel, we refer to these two parts as *first subpart* and *second subpart*. The attack of Schläffer in [17] proceeds similarly but uses the rebound attack technique in the two subparts. We reuse this idea of finding message pairs conforming to partial truncated parts but most of our new techniques avoid the rebound attack on the **SuperSBox**. Both subparts are represented in the Figure 4: the first one starts in S7 and ends in S14 and fixes the red bytes of the two messages, whereas the second one starts at S16 until the end of the four rounds in S31 and fixes the yellow bytes. Additionally, the chaining value in the first round of the path are the blue bytes.

SuperSBox. In the differential path described on Figure 4, there are many differential transitions through the **SuperSBox** of the third round where input differences are reduced to *one* active byte. We are then interested in differential transitions such as the one described in Figure 5. For this kind of transition, the distribution difference table of the **SuperSBox** would work but requires 2^{64} to be computed and stored[3]. We show that we can compute a pair of columns satisfying this path in 2^{11} operations on one AES-column.

Let us consider the input difference to be $\Delta_i = [\delta_i,\, 0,\, 0,\, 0]^{\mathrm{T}}$ reduced to one active byte δ_i and the output difference $\Delta_o = \left[\delta_o^1,\, \delta_o^2,\, \delta_o^3,\, \delta_o^4\right]^{\mathrm{T}}$: we aim at finding a pair of AES-columns (c_1, c_2) conforming to those differences; that is: $c_1 + c_2 = \Delta_i$ and $\mathbf{SuperSBox}(c_1) + \mathbf{SuperSBox}(c_2) = \Delta_o$. In a precomputation phase of 2^{16}, we compute and store the differential distribution table of the AES S-Box.

[3] In that case, we could compute and store smaller tables in 2^{40} for the four possible positions of active bytes.

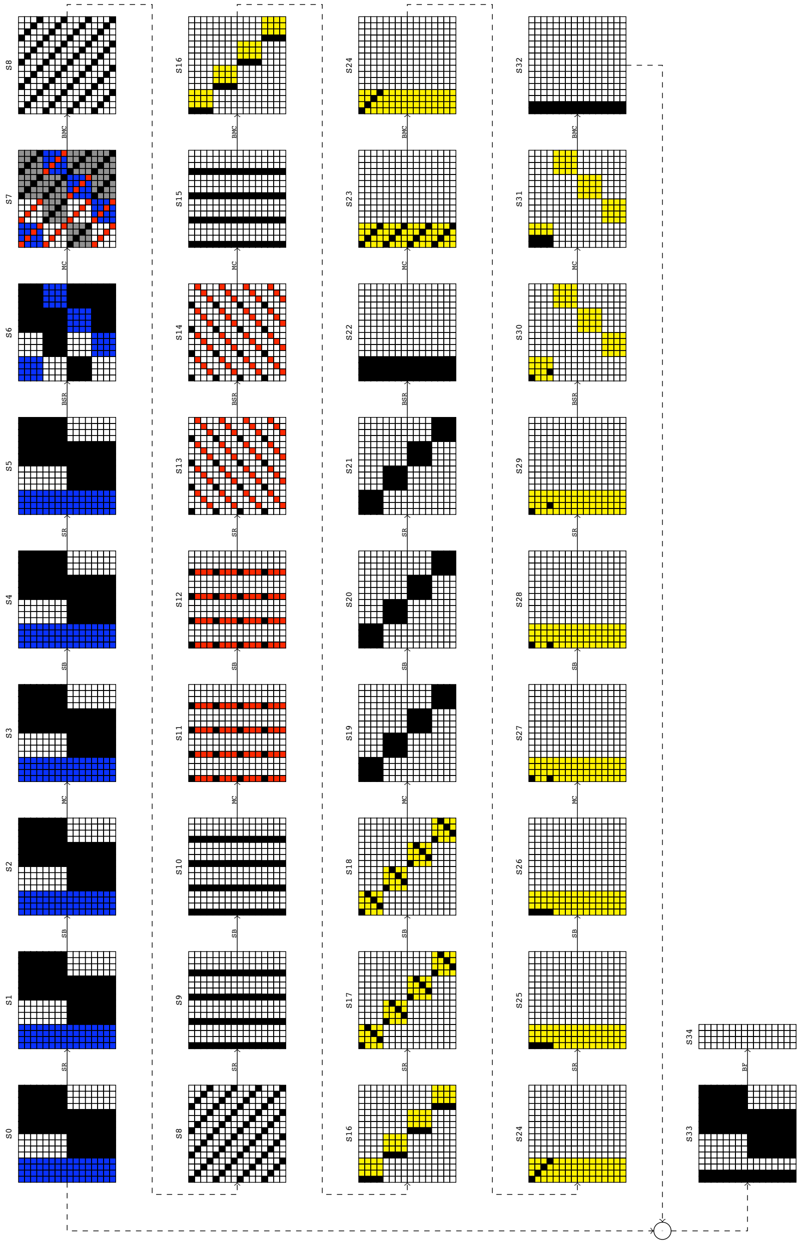


Fig. 4. The differential path used in this attack on the ECHO-256 compression function reduced to four rounds. To find a valid pair of messages, we split the path into two parts: the first subpart between S7 and S14 (red bytes) and the second subpart between S16 and S31 (yellow bytes). Black bytes are the only active bytes, blue bytes come from the chaining value and gray bytes in the first round are set to get a collision (or a near-collision) in the compression step.

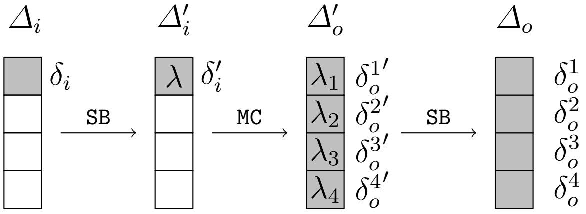


Fig. 5. A **SuperSBox** differential transition with only one active input byte.

The differential properties of the AES S-Box restrict the number of output differences of the first **SubBytes** layer to $2^7 - 1$ and for each one, the underlying values are set. Denoting δ'_i one of the output differences of this layer and λ the associated value such that $S^{-1}(\lambda) + S^{-1}(\lambda + \delta'_i) = \delta_i$, we can propagate this difference $\Delta'_i = [\delta'_i,\, 0,\, 0,\, 0]^{\mathrm{T}}$ linearly to learn the four differences at the input of the second **SubBytes** layer. We note $\Delta'_o = \mathbf{MC}(\Delta'_i) = [\delta_o^{1'},\, \delta_o^{2'},\, \delta_o^{3'},\, \delta_o^{4'}]^{\mathrm{T}}$ those differences. Here, both the input and the output differences are known and the four differential transitions $\delta_o^{i'} \rightarrow \delta_o^i$ exist with probability approximately 2^{-4}. Since we can restart with $2^7 - 1$ different δ'_i, we get approximately 2^3 valid differential transitions. Each of these transitions fixes the underlying values, noted $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. At this point, all intermediate differences conform to the path, but in terms of values, we need to ensure that λ is consistent with $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. To check this, we exhaust the 2^4 valid vectors of values we can build by interchanging λ_i and $\lambda_i + \delta_o^{i'}$.

All in all, among the 2^{3+4} vectors of values we can build, only a fraction 2^{-8} will satisfy the 8-bit condition on λ. This means that the considered differential transition $\Delta_i \rightarrow \Delta_o$ through the **SuperSBox** occurs with probability 2^{-1} and if the transition exists, we can recover an actual AES-column pair in $2^7\, 2^4 = 2^{11}$ operations.

3.4 Overview of the attack

In this subsection, we describe the main steps used to find a message pair conforming to the differential path. We begin by the sensitive part of the attack, which caused erroneous statements in [17]: the merging phase of the two partial solutions.

Merging step. Assume for a moment that we solved both subparts of the path, i.e. the red bytes between S7 and S14 are fixed as well as the yellow ones between S16 and S31: we have two partial solutions for the complete differential path. The truncated differential of Figure 4 is then partially verified but to merge the two parts, we need to set the white bytes so that the **SuperMixColumns** transition from S14 to S16 is possible.

Due to the particular construction of $\mathbf{M}_{SMC}$, some algebra considerations show that for the already-known values in S14 (red) and S16 (yellow), the **SuperMixColumns** transition will not be possible unless a 128-bit constraint is satisfied: the remaining degrees of freedom can not be used to satisfy this relation. Since all of the 16 column-slices of the considered matrices are independent, this leads to 16 constraints on 8 bits.

The flaw in [17] is to assume these relations are true, which holds only with probability 2^{-128}, whatever the value of unset bytes are. These equalities need to be true so that the 16 linear systems have solutions. The first system associated to the first slice is given by:

$$\begin{aligned}\mathbf{M_{SMC}}\left[a_0\ x_0\ x_1\ x_2\ a_1\ x_3\ x_4\ x_5\ a_2\ x_6\ x_7\ x_8\ a_3\ x_9\ x_{10}\ x_{11}\right]^{\mathrm{T}} &= \\ \left[b_0\ b_1\ b_2\ b_3\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\right]^{\mathrm{T}}&\end{aligned} \tag{1}$$

where a_i and b_i are known values, x_i are unknowns and $*$ is any value in GF $\left(2^8\right)$. This system has solutions if a particular linear combination of $[a_0,\ a_1,\ a_2,\ a_3]^{\mathrm{T}}$ and $[b_0,\ b_1,\ b_2,\ b_3]^{\mathrm{T}}$ lies in the image of some matrices: this constraints the already-known values to verify an 8-bit relation. The constraint comes from the fact that $\mathbf{M}_{SMC}$ is the Kronecker product $\mathbf{M} \otimes \mathbf{M}$. For example, in the following, we denote by a_i, $0 \leq i \leq 3$, the four known values of slice 0 of S14 coming from the first subpart (red) and b_i the known values for the same slice in S16, from the second subpart (yellow). With this notation, the first system will have solutions if and only if the following condition is satisfied:

$$2a_0 + 3a_1 + a_2 + a_3 = 14b_0 + 11b_1 + 13b_2 + 9b_3. \tag{2}$$

See Appendix A for the detailed proof. These constraints for each slice of the **SuperMixColumns** transition can also be viewed in a different way: consider all the b_i known for all slices, thus we can only pick 3 out of 4 a_i per slice in S14 and determine the last one deterministically. Alternatively, due to the **ShiftRows** and **BigShiftRows** transitions, we can independently determine slices 0, 4 and 8 in S12 so that slice 12 of S12 would be totally determined. This transfers the problem into the first subpart of the path.

Step 1. We begin by finding a pair of `ECHO`-columns satisfying the truncated path reduced to the first `ECHO`-column between S7 and S12. This is done with a randomized AES-state of the column, used to get and solve linear equations giving all differences between S7 and S9. Indeed, differences between S7 and S9 for the first column only depend on the four differences in S7[2,0][4]. Then, we search for valid differential transitions through the AES S-Box between S9 and S10 to finally deduce a good pair of `ECHO`-columns. This step can be done in 2^{12} operations on AES-columns (Section 4.1).

[4] Linear relations can be deduced by linearly propagating differences in S7[2,0] forwards until S9.

Step 2. Once we solved the first ECHO-column, we can deduce all differences between S12 and S16: indeed, the wanted **SuperMixColumns** transition imposes them as discussed in Section 3.1. This step is done in constant time (Section 4.2).

Step 3. Now that we have the differences in S16, we have a starting point to find a message pair for the second subpart of the whole truncated path: namely, states between S16 and S31 (yellow bytes). To do so, the idea is similar as in Step 1: since all differences between S20 and S24 only depend on the four differences of S24[5], we can use a randomized AES-column c in S18 to get four independent linear equations in S20 and thus deduce all differences between S20 and S24. Then, we search for input values for the 15 remaining **SuperSBoxes**, which have only one active byte at their input (Section 3.3). This succeeds with probability 2^{-15} so that we need to retry approximately 2^{15} new random c. The whole step can be done in 2^{26} operations on AES-columns (Section 4.2).

Being done, the truncated path is followed until the end of the four rounds in S32. Note that we can filter the **MixColumns** transition between S26 and S27 in a probabilistic way so that less slices would be active in S32.

Step 4. Getting back to the first subpart of the truncated path, we now find a valid pair of ECHO-columns satisfying the truncated path between S7 and S12 reduced to the second ECHO-column. This is basically the same idea as in Step 1. This can be done in 2^{12} operations on AES-columns as well (Section 4.3). Note that this step could be switched with Step 3.

Step 5. To construct a valid pair of ECHO-columns satisfying the truncated path between S7 and S12 reduced to the third ECHO-column, we proceed as before (steps 1 and 4), but we start by randomizing three AES states instead of one: indeed, differences between S7 and S9 at the input of the non-linear layer now depend on 12 differences, the ones in S7[0,2], S7[1,2] and S7[3,2]. Getting 12 linear systems then allow to learn those differences and we can finally search for four valid differential transitions through the AES S-Box in 2^4 operations on AES-columns (Section 4.3).

Step 6. The merging step in [17] fails with high probability, but we know how to get into the valid cases: since the three first ECHO-columns of the first subpart are now known, we can deduce the whole last ECHO-column allowing the 16 needed equations mentioned before. There is no freedom for that column, so we are left with a probabilistic behavior to check if it follows the column-reduced truncated differential. We then propagate the pair of deduced values backwards until S8 and check if the **invBigMixColumns** transition behave as wanted: namely, four simultaneous 4 → 3 active bytes, which occurs with probability $(2^{-8})^4$. Hence,

[5] Linear relations can be deduced by linearly propagating the four differences of S24[0,0] backwards until S20.

we need to restart approximately 2^{32} times the previous Step 5 to find a valid pair of ECHO-columns satisfying both the path between S7 and 12 and the 128-bit condition imposed by the merging step. This step can be performed in 2^{36} operations on AES-columns (Section 4.3).

Step 7. To get a collision in the compression function, we then need to take care of the compression phase in the **BigFinal** operation: the feed-forward and the xor of the four ECHO-columns. The collision is reached when the sum of the two active AES-states per row in S0 equals the active one in S32. We have enough degrees of freedom to determine values in associated states of S7 (gray) to make this happens. Together with the probabilistic filter of Step 3, this step may impose the global time complexity of the attack; so, weakening the final objective (to get a near-collision, for instance) can make the whole attack practical (Section 4.4).

Step 8. The last step consists in filling all the remaining bytes by solving the 16 linear systems mentioned in Step 6, while taking care at the same time that the **invBigMixColumns** between S8 and S7 reaches the values determined by Step 7. Due to the particular structure of the solution sets, the systems can be solved in parallel in 2^{32} operations on AES-columns (Section 4.5).

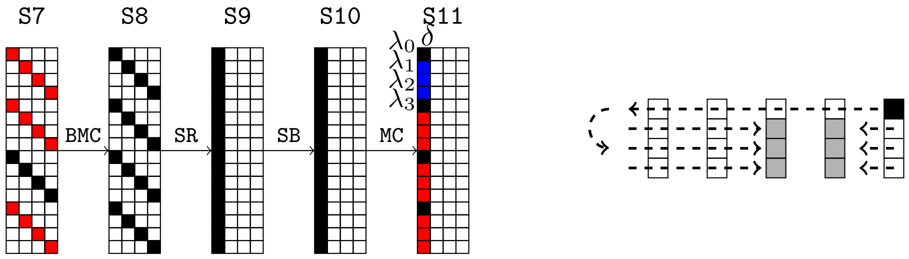
4 Collision on the 4-round compression function

4.1 Partial message pair for the first subpart

This step aims at finding a pair of ECHO-columns satisfying the truncated differential of Figure 6. We consider the first column separately from the others in order to reach a situation where the merging process will be possible. Indeed, once we fix a slice, we can determine the differences at the beginning of the second subpart in S16.

The previous method suggested in [17] (in Section 4.1) to find paired values following this truncated differential is a rebound attack working in time 2^{32} and using the differential distribution table of the **SuperSBox** of size 2^{64}. We show how we can find an ECHO-column pair conforming to this reduced path in 2^{12} operations on AES-columns without significant memory usage.

Rather than considering the whole column at once, we start by working on the top AES state in S11, that is S11[0,0]. We begin by choosing random values $(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$ for the first AES-column of S11[0,0] (blue bytes), such that the active byte is set to difference δ, also chosen at random in GF $\left(2^8\right) \setminus \{0\}$. Starting from S11[0,0] and going backwards, those values and differences are propagated deterministically until S8[0,0]. Since there is only one active byte per slice in the considered ECHO-column of S7, each of the associated four slices of S8 lies in a subspace of dimension one. Therefore, solving four simple linear systems leads to the determination of the 12 other differences of S8.



(a) Truncated differential.

(b) Flow of AES states.

Fig. 6. Truncated differential path (a) used for the first subpart of the attack for one `ECHO`-column. We represent on (b) the order in which AES states are randomized (black) or deduced by a small rebound attack (gray).

Therefore, in the active slice of S9 of Figure 6 at the input of the **SubBytes** layer, the four first paired bytes have values and differences known, whereas in the 12 other positions, only differences are set. Our goal now is to find good values for these byte pairs, which can be achieved by a small rebound attack on the AES S-Box where the output differences are propagated from S11 by choosing random differences. Thus, we iterate on the $(2^8)^3$ possible unset differences of S11 and propagate them linearly to S10. When both input and output differences of the 12 AES S-Boxes are known, we just need to ensure that these 12 differential transitions are possible. This is verified by the precomputed table[6] Δ. It ensures that the 12 transitions will occur simultaneously with probability 2^{-12}. Since we can try approximately $(2^8)^3$ output differences, we will have about $(2^8)^3\ 2^{-12}\ (2^4)^3 \approx 2^{24}$ different vectors of values by trying them all. The factor $(2^4)^3$ comes from the possibility of interchanging the two solutions of the equations 12 times to get more vectors of values[7].

All in all, for any $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \delta$ picked at random in GF $\left(2^8\right)$ (with non-null difference), we can find 2^{24} `ECHO`-column pairs in S12 in 2^{12} such that the associated truncated differential from S12 to S7 is verified. We could thus build approximately $2^{24+8\times 5} = 2^{64}$ pairs of `ECHO`-columns following the column-reduced truncated differential.

4.2 Finding a message pair for the second subpart

We now get a partial message pair conforming to the first subpart of the truncated path reduced to a single `ECHO`-column. Rather than completing this partial message pair for the three other active slices in S12, we now find a message pair conforming to the second subpart of the truncated path, located in the third round from S16 to S24 (yellow bytes).

[6] This is the differential distribution table of the AES S-Box, which required 2^{16} bits to be stored.

[7] There are cases where four solutions exist, but for simplicity, we consider only the more common two-solution case.

Indeed, the mere knowledge of a single active slice pair of S12 in the first subpart is sufficient to get a starting point to find a message pair for the second subpart, i.e. yellow bytes. This is due to the desired transition through the **SuperMixColumns** transition: as explained in Section 3.1, differences in S14 lie in one-dimensional subspaces. Once a slice pair for the first slice of S12 is known and computed forwards to S14 (black and red bytes on Figure 7), there is no more choice for the other differences in S14. Finally, all differences between S12 and S17 have been determined by linearity of involved transformations.

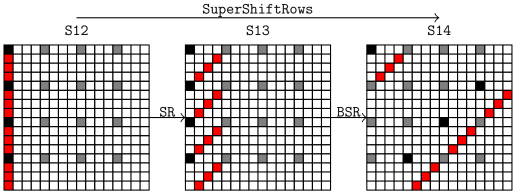


Fig. 7. The **SuperShiftRows** layer where only the values and differences of the first slice of S12 are known (black and red bytes).

At this point, only input differences of **SuperSBoxes** of the third round are known. We note that all operations between S20 and S24 are linear, so that all differences in those states only depend on the four differences of S24. We denote by k_i the non-null difference of column slice $i \in \{0, 1, 2, 3\}$ in state S24. By linearly propagating differences in S24 backwards to S20, we obtain constraints on the 64 output differences of the **SuperSBox** in S20. To find the actual differences, we need to find the four k_i and thus determine four independent linear equations. Considering arbitrarily the first AES-column of S20[0,0] (Figure 8), differences are: $[84k_0, 70k_3, 84k_2, 70k_1]^{\mathrm{T}}$ (black bytes).

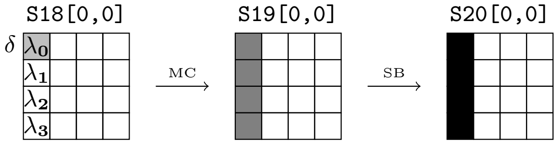


Fig. 8. The **MixColumns** and **SubBytes** transitions on the first AES-column between S18[0,0] and S20[0,0].

Starting from S18, let δ a random difference among the $2^7 - 1$ possible ones imposed by S17 for the considered columns (Figure 8). Any choice imposes the value associated to the differential transition as well: we denote it λ_0. At this step, we introduce more freedom by picking random values for the three remaining bytes of the column: $(\lambda_1, \lambda_2, \lambda_3)$. Note that we can choose $(2^8)^3 = 2^{24}$ of them and thus 2^{31} starting points in total. After this randomization the AES-column in S18, the same AES-columns in S19 and S20 are fully determined. We then need to link the four bytes with the differences provided by the right part of the path from S24 to S20: this is done by simple algebra by solving four linear equations in four variables, which are k_i, $0 \leq i \leq 3$.

After solving, we have the four differences k_i of state S24: we propagate them backwards from S24 to S20 and learn all the differences between S20 and S24. Only one pair of AES-columns out of the 16 was used in S18 to deduce differences k_i in S24, so we now try to find values for the 15 left (Figure 9).

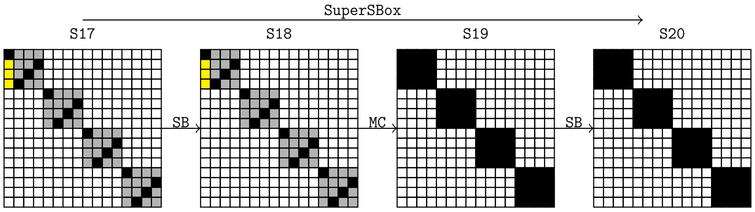


Fig. 9. Last step to get a message pair conforming to the second subpart of the path: finding the 15 remaining AES-columns using the **SuperSBox** properties. Black bytes are active and yellow bytes have already been defined in the previous step, as well as differences of the first AES-column of the first AES-state. Gray bytes are inactive and the target of this step.

Each of the remaining AES-columns, can be viewed as a differential transition through a **SuperSBox** between S17 and S20 where all differences have been previously set. As described in 3.3, we have 15 differential transitions through the **SuperSBox** with only one input active byte in each. The 15 transitions occur simultaneously with probability 2^{-15} and if so, we can recover the 15 AES-column pairs in parallel in 2^{11} using the technique previously described. Since there are 15 AES-columns to find in S17, we need to generate approximately 2^{15} new $(\delta, \lambda_0), \lambda_1, \lambda_2, \lambda_3$ and restart the randomization in S18[0,0].

Considering *one* message pair conforming to a single ECHO-column of the first subpart of the truncated path as starting point, the number of pairs we can build which follow the truncated path for this second subpart is: $2^7\ 2^{8\times3}\ 2^{-15} \approx 2^{16}$. We note that we get one in 2^{26} operations in parallel.

In the collision attack on the compression function, we further extend this step by probabilistically filtering the active bytes in the **MixColumns** transition between S26 and S27. Among the 2^{16} message pairs we can build that follow the truncated path between S16 and S26, only one in average will verify the $4 \to 2$ transition through **MixColumns**. If such a pair is found then the pair conforms the truncated path until the end of the four rounds; otherwise, we need to find a new starting point, i.e. a new slice pair for slice 0 in S12. We reduce to two active bytes and not one or three because this is a best compromise we can make to lower the overall time complexity of the collision attack.

4.3 Completing the partial message pair of the first subpart

As discussed in Section 3.4, to solve the merging step, slice 12 of S12 is constrained by slices 0, 4 and 8 of S12. All values of slice pair 0 have been determined (Section 4.1) and used to fix yellow bytes and thus get a message pair conforming to the second subpart of the truncated path (Section 4.2).

Consequently, we only have freedom on the slice pairs 4 and 8 in S12. We determine values of slice pair 4 in the same way as slice 0 by considering the first subpart of the truncated path from S7 to S14 reduced to the second ECHO-column. There is a single active byte per slice in this ECHO-column of S7, so that we can build approximately 2^{60} valid columns[8] in that position in 2^{12} operations on AES-columns for a single one.

As soon as we have one, we use the remaining freedom of slice 8 to generate simultaneously slice pairs 8 and 12 of S12. We note that in the two last ECHO-columns of S7, there are three active bytes per slice (Figure 4). The method we suggest starts by finding a slice pair for slice 8 conforming to the truncated differential reduced to the third ECHO-column between S7 and S12. We proceed in the same way as we did for slices 0 and 4 and then, we deduce deterministically the slice pair 12 from the constraints imposed by the merge. Finally, we check whether that slice pair conforms the truncated differential reduced to the last ECHO-column until S7, namely the four simultaneous transitions $4 \rightarrow 3$ through **invMixColumns** between S8 and S7.

The cost of 2^4 to construct a slice pair for slice 8 allows to repeat it 2^{32} times to pass the probability $\left(2^{-8}\right)^4$ of finding a valid slice pair for slice 12 conforming to both the linear constraints of the merge and the truncated differential through **invBigMixColumns**. Note that we have enough degrees of freedom to do so since we can find approximately $(2^7)^4\,(2^8)^{3\times 3} = 2^{100}$ valid slice pairs for slice 8. However, only 2^{32} are needed, which completes this step in 2^{36} operations on AES-columns and fixes all the red bytes between S7 and S14.

4.4 Compression phase in the feed forward

After four rounds, the round-reduced compression function applies the feed forward (S33 $\leftarrow$ S0+S32) and XORs the four columns together (**BigFinal**). This operation allows to build the differential path such that differences would cancel out each other. As shown in the global path (Figure 4), states S0 and S32 XORed together lead to state S33 where there are three active AES-states in each row. In terms of differences, if each row sums up to zero, then we get a collision for the compression function in S34 after the **BigFinal**.

As we constructed the path until now, in both S0 and S32, we still have freedom on the values: only differences in S32 located in the two first slices are known from the message pair conforming to the second subpart of the truncated

[8] Note that in Section 4.1, we could build 2^{64} of them because differences were chosen freely, whereas in the present case, differences are constrained by the AES S-Box differential properties to sets of size $2^7 - 1$. We thus loose 2^4 degrees of freedom.

path. These differences thus impose constraints on the two other active pair states per row in S0. Namely, for each row r of S0 where active AES states are located in columns c_r and c'_r, we have $\mathrm{S0}[r, c_r]+\mathrm{S0}[r, c'_r] = \mathrm{S32}[r, 0]$. Additionally, differences in S4 are known by linearly propagating the known differences from S7.

After the feed-forward, we cancel differences of each row independently: we describe the reasoning for an arbitrary row. We want to find paired values in the two active states of the considered row of S0, say (A, A') and (B, B'), such that they propagate with correct differences in S4, which are known, and with correct diagonal values (red bytes) in S7 after the **MixColumns**. In the sequel (Figure 10), we subscript the AES-state A by j to indicate that A_j is the AES-state A propagated until `ECHO`-state Sj with relevant transformations according to Figure 4.

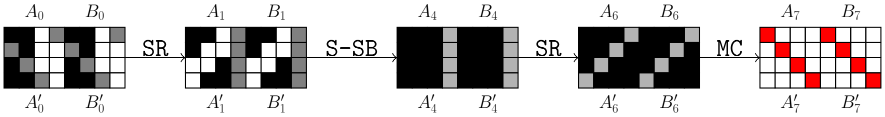


Fig. 10. Propagation of the pairs of AES-states (A_i, A'_i) and (B_i, B'_i) in a single `ECHO`-row in the first round. Non-white bytes represent active bytes; those in S7 (in red) are the known values and differences from the message pair conforming to the first subpart of the truncated path.

The known differences of S4 actually sets the output differences of the **SuperSBox** layer: namely, $A_4 + A'_4 = \Delta_4$ and $B_4 + B'_4 = \Delta'_4$, where Δ_4 and Δ'_4 are the known differences in the considered row of S4. The constraint on the known diagonal values in A_7 and B_7 restricts the available freedom in the choice of the AES-columns of A_6 and B_6 (and linearly, to their equivalent A'_6 and B'_6 with diagonal values in A'_7 and B'_7) to reach the already-known diagonal values in S7 (red bytes). An alternative way of stating this is: we can construct freely the three first columns of (A_4, A'_4) and (B_4, B'_4) and deduce deterministically the fourth ones with the next **MixColumns** transition, since 4 out of 8 input or output bytes of **MixColumns** fix the 4 others. Furthermore, this means that if the three first columns of A_1, A'_1, B_1 and B'_1 are known, then we can learn the values of the remaining columns of S1 (bytes in gray).

We thus search valid input values for the three first **SuperSBoxes** of S1: to do so, we randomize the two differences per AES-column in this state and get valid paired values with probability 2^{-1} in 2^{18} computations with respect to output differences Δ_4 (Section 3.3). Consequently, we can deduce the differences of the same AES-columns in $B_1 + B'_1$ to get a zero sum with S32 after the **BigFinal**. This holds with the same 2^{-1} probability, with respect to Δ'_4. Once we have the three differential transitions for the three first AES-columns of both AES-states, all the corresponding values are then known and we propagate them in A_6, A'_6, B_6 and B'_6 (black bytes). Since in S7, diagonal values are known, we deduce the remaining byte of each column in A_6, A'_6, B_6 and B'_6 (gray) and propagate them backwards until S1.

The final step defines the nature of the attack: to get a collision, we check if those constrained values cancel out in the feed-forward, which holds with probability 2^{-32}. Restarting with new random values in S1 and in parallel on the four rows, we find a collision in $2^{18}\,2^{2}\,2^{32} = 2^{52}$ operations on AES-columns. Indeed, we need to repeat 2^{32} times the search of valid paired input values for the **SuperSBox**, which is done in time 2^{18} and succeeds with probability 2^{-2}.

4.5 Final merging phase

After we have found message pairs following both subparts of the truncated path so that the merge is possible, we need to finalize the attack by merging the two partial solutions.

In practice, this means finding values for each white bytes in the truncated path and in particular, at the second **SuperMixColumns** transition between S14 and S16. For each of the 16 slices, we get a system of linear equations like (1). In each solution set, each variable only depends on 3 others, and not on *all* the 11 others. This stems from the structured matrix $\mathbf{M}_{SMC}$. For example, in the first slice, we have:

$$L_0(x_0, x_3, x_6, x_9) = c_0 \tag{3}$$

$$L_1(x_1, x_4, x_7, x_{10}) = c_1 \tag{4}$$

$$L_2(x_2, x_5, x_8, x_{11}) = c_2 \tag{5}$$

where L_0, L_1, L_2 are linear functions and c_0, c_1, c_2 constants linearly deduced from the 8 known-values a_i and b_i, $0 \le i \le 3$, of the considered system.

In this phase of the merging process, we also need to set white bytes accordingly to the known values in S7 stemming from the feed-forward. We pick random values for unset bytes in S7[1,3] and S7[2,2] (Figure 11), such that all values in the two last `ECHO`-columns of S7 are set. Consequently, by indirectly choosing values for gray bytes in S14, we set the values of half of the unknowns

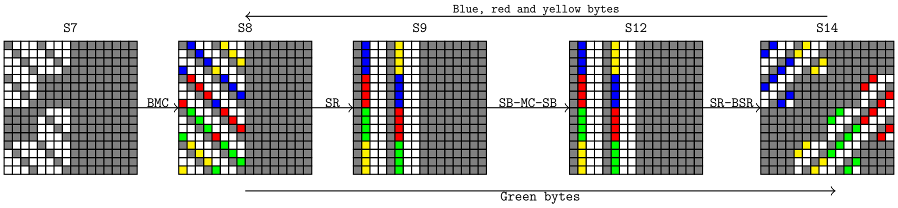


Fig. 11. After randomization of states S7[1,3] and S7[2,2], all values of gray bytes are known. Colors show the flow of values in one step of the merging process.

per slice. For example, the system for the first slice becomes:

$$L'_0(x_0, x_3) = c'_0 \tag{6}$$
$$L'_1(x_1, x_4) = c'_1 \tag{7}$$
$$L'_2(x_2, x_5) = c'_2 \tag{8}$$

where L'_0, L'_1, L'_2 are linear functions and c'_0, c'_1, c'_2 some constants.

The three equations (6), (7), (8) are independent, which allows to do the merge in three steps: one on each pair of slices $(1, 5)$, $(2, 6)$ and $(3, 7)$ of S12. Figure 11 represents in color only the first step, on the slice pair $(1, 5)$ of S12. We show that each of the three steps can be done in 2^{32} computations and detail only the first step.

Because of the dependencies between bytes within a slice in S14, any choice of blue bytes in S12[0,0] determines blue bytes on S12[1,1] (and the same for yellow and red bytes, Figure 11). In total, we can choose $\left(2^{8\times4}\right)^3 = 2^{96}$ different values for the blue, yellow and red AES-columns of state S12. Since we are dealing with values, we propagate them backwards until S8. The **BigMixColumns** transition from S7 to S8 for these two slices imposes the 8 green values in S8[2,0] and S8[3,1]. Going forwards through the **SuperSBox**, we deduce green values in S14 and check whether the four pairs of green bytes satisfy the linear constraints in S14, which occur with probability $\left(2^{-8}\right)^4 = 2^{-32}$. We then have to restart with approximately 2^{32} new blue bytes and random yellow and red ones before satisfying the four constraints simultaneously.

After repeating this step for slices $(2, 6)$ and $(3, 7)$, we get a valid message pair that follows all the truncated path of Figure 4.

5 Conclusion

In this article, we introduce new results on ECHO-256 compression function reduced to four rounds by describing a collision attack. Our result is the first one which does not need to store the large difference distribution table of the **SuperSBox**, which contributes in making the attack practical. We also prove that the latest results by Schläffer on ECHO are flawed and we suggest a way to correct it in some ways. We also improve the time and space complexity of the attack by taking into account more precisely the available degrees of freedom. We describe as well an efficient way to find paired input values conforming to particular truncated differentials through the **SuperSBox** where not all input bytes are active. Finally, we validate our claims by implementing a practical variant of the described attack. We believe this work can lead to new attacks: in particular, the collision attack by Schläffer on ECHO-256 might be corrected using our new techniques.

References

1. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (updated) (2009)

2. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: CRYPTO'91. (1991)
3. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong, S., Iwata, T., eds.: FSE. Volume 6147 of Lecture Notes in Computer Science., Springer (2010) 365–383
4. Knudsen, L.R.: Truncated and higher order differentials. In: Fast Software Encryption - Second International Workshop, Leuven, Belgium, LNCS 1008, Springer-Verlag (1995) 196–211
5. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 315–324
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui, M., ed.: Advances in Cryptology – ASIACRYPT 2009, Proceedings. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 126–143
7. Le, T.V., Sparr, R., Wernsdorf, R., Desmedt, Y.: Complementation-Like and Cyclic Properties of AES Round Functions. In Dobbertin, H., Rijmen, V., Sowa, A., eds.: AES Conference. Volume 3373 of Lecture Notes in Computer Science., Springer (2004) 128–141
8. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 106–125
9. 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, Proceedings. Volume 5867 of Lecture Notes in Computer Science., Springer (2009) 16–35
10. 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. Volume 5665 of Lecture Notes in Computer Science., Springer (2009) 260–276
11. National Institute for Science, Technology (NIST): Advanced E.D.F.-G.D.F.ncryption Standard (FIPS PUB 197) (November 2001) http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
12. National Institute of Standards, Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Technical report, DEPARTMENT OF COMMERCE (November 2007)
13. Peyrin, T.: Cryptanalysis of Grindahl. In Kurosawa, K., ed.: Advances in Cryptology – ASIACRYPT 2007, Proceedings. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 551–567
14. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO. Volume 6223 of Lecture Notes in Computer Science., Springer (2010) 370–392
15. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: Advances in Cryptology – CRYPTO 2010. Volume 6223 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2010) 370–392
16. 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. Volume 6477 of Lecture Notes in Computer Science., Springer (2010) 38–55
17. Schläffer, M.: Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. (2010) To appear in Selected Areas in Cryptography 2010, Proceedings.

18. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer, R., ed.: Advances in Cryptology – EUROCRYPT 2005, Proceedings. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 1–18
19. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: Advances in Cryptology – CRYPTO 2005, Proceedings. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
20. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: Advances in Cryptology – EUROCRYPT 2005, Proceedings. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 19–35
21. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In Shoup, V., ed.: Advances in Cryptology – CRYPTO 2005, Proceedings. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 1–16

A Merging process in detail

An instance of the problem to solve is the following: given a_0, a_1, a_2, a_3, b_0, b_1, b_2, $b_3 \in \mathrm{GF}\left(2^8\right)$, find x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, $x_{11} \in \mathrm{GF}\left(2^8\right)$ such that:

$$\begin{aligned}\mathbf{M_{SMC}} &\begin{bmatrix}a_0 & x_0 & x_1 & x_2 & a_1 & x_3 & x_4 & x_5 & a_2 & x_6 & x_7 & x_8 & a_3 & x_9 & x_{10} & x_{11}\end{bmatrix}^{\mathrm{T}} = \\ &\begin{bmatrix}b_0 & b_1 & b_2 & b_3 & * & * & * & * & * & * & * & * & * & * & * & *\end{bmatrix}^{\mathrm{T}}\end{aligned} \tag{9}$$

where $*$ is any value in $\mathrm{GF}\left(2^8\right)$. Since we are only interested in the four first output values (the problem is similar for others slices), we do not take into consideration the lines other than the four first ones. Let $\mathbf{M_{SMC}}|_{0,1,2,3}$ be that matrix. The system to be solved can be rewritten as ($\mathbf{M_{SMC}}|_{0,1,2,3}^{j}$ is the matrix composed of rows $0, 1, 2, 3$ and column j from $\mathbf{M_{SMC}}$):

$$\begin{aligned}\mathbf{M_{SMC}}|_{0,1,2,3}^{1,2,3,5,6,7,9,10,11,13,14,15} &\begin{bmatrix}x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11}\end{bmatrix}^{\mathrm{T}} = \\ &\mathbf{M_{SMC}}|_{0,1,2,3}^{0,4,8,12}\begin{bmatrix}a_0\\a_1\\a_2\\a_3\end{bmatrix} + \begin{bmatrix}b_0\\b_1\\b_2\\b_3\end{bmatrix}\end{aligned} \tag{10}$$

Now, we make the assumption that at least one solution to the problem exists. This means that the right-hand side of (10) lies in the image of the matrix $\mathbf{M_{SMC}}|_{0,1,2,3}^{1,2,3,5,6,7,9,10,11,13,14,15}$ from the left-hand side. Because the matrix $\mathbf{M_{SMC}}$ is a Kronecker product of $\mathbf{M}$ with itself, $\mathbf{M_{SMC}}|_{0,1,2,3}^{1,2,3,5,6,7,9,10,11,13,14,15}$, $\mathbf{M_{SMC}}|_{0,1,2,3}^{9,10,11,13,14,15}$ and $\mathbf{M_{SMC}}|_{0,1,2,3}^{1,2,3,5,6,7}$ share the same image, described by:

$$S_0 = \left\{ \quad [t_0, t_1, t_2, L(t_0, t_1, t_2)]\,, t_0, t_1, t_2 \in \mathrm{GF}\left(2^8\right) \quad \right\} \tag{11}$$

where $L(t_0, t_1, t_2) = 247t_0 + 159t_1 + 38t_2$. Finally, if a solution exists, this means that:

$$\underbrace{\begin{bmatrix} 4 & 6 & 2 & 2 \\ 2 & 3 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 6 & 5 & 3 & 3 \end{bmatrix}}_{\mathbf{M_{SMC}}|^{0,4,8,12}_{0,1,2,3}} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \in S_0 \tag{12}$$

In other words, this means that the following equality is true:

$$14b_0 + 11b_1 + 13b_2 + 9b_3 = 2a_0 + 3a_1 + a_2 + a_3. \tag{13}$$

The given parameters a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3 are then constrained on an 8-bit condition. The converse is then: if this relation is not satisfied, then the problem has no solution.

We took the example of the very first slice, but the problem is similar for the 16 different slices in S14/S16. Namely, per slice, parameters need to satisfy the following equalities:

Slice	**Condition**
0	$14b_0 + 11b_1 + 13b_2 + 9b_3 = 2a_0 + 3a_1 + a_2 + a_3$
1	$11b_0 + 13b_1 + 9b_2 + 14b_3 = 2a_0 + 3a_1 + a_2 + a_3$
2	$13b_0 + 9b_1 + 14b_2 + 11b_3 = 2a_0 + 3a_1 + a_2 + a_3$
3	$9b_0 + 14b_1 + 11b_2 + 13b_3 = 2a_0 + 3a_1 + a_2 + a_3$
4	$14b_0 + 11b_1 + 13b_2 + 9b_3 = a_0 + 2a_1 + 3a_2 + a_3$
5	$11b_0 + 13b_1 + 9b_2 + 14b_3 = a_0 + 2a_1 + 3a_2 + a_3$
6	$13b_0 + 9b_1 + 14b_2 + 11b_3 = a_0 + 2a_1 + 3a_2 + a_3$
7	$9b_0 + 14b_1 + 11b_2 + 13b_3 = a_0 + 2a_1 + 3a_2 + a_3$
8	$14b_0 + 11b_1 + 13b_2 + 9b_3 = a_0 + a_1 + 2a_2 + 3a_3$
9	$11b_0 + 13b_1 + 9b_2 + 14b_3 = a_0 + a_1 + 2a_2 + 3a_3$
10	$13b_0 + 9b_1 + 14b_2 + 11b_3 = a_0 + a_1 + 2a_2 + 3a_3$
11	$9b_0 + 14b_1 + 11b_2 + 13b_3 = a_0 + a_1 + 2a_2 + 3a_3$
12	$14b_0 + 11b_1 + 13b_2 + 9b_3 = 3a_0 + a_1 + a_2 + 2a_3$
13	$11b_0 + 13b_1 + 9b_2 + 14b_3 = 3a_0 + a_1 + a_2 + 2a_3$
14	$13b_0 + 9b_1 + 14b_2 + 11b_3 = 3a_0 + a_1 + a_2 + 2a_3$
15	$9b_0 + 14b_1 + 11b_2 + 13b_3 = 3a_0 + a_1 + a_2 + 2a_3$

The main problem in the reasoning of [17] is to assume that a solution exists, while for some parameters, there is no solution.

In the end, if the condition is verified we can choose x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 freely and determine x_9, x_{10}, x_{11} afterwards. If a solution exists, there are $\left(2^8\right)^9 = 2^{72}$ solutions to the problem. Taking any other slice leads to a very similar description of the set of solutions, with the same kind of dependencies between the variables.