Improved Analysis of ECHO-256*

Jérémy Jean[1], María Naya-Plasencia[2], and Martin Schläffer[3]

[1] Ecole Normale Supérieure, France
[2] FHNW, Windisch, Switzerland
[3] IAIK, Graz University of Technology, Austria

Abstract. ECHO-256 is a second-round candidate of the SHA-3 competition. It is an AES-based hash function that has attracted a lot of interest and analysis. Up to now, the best known attacks were a distinguisher on the full internal permutation and a collision on four rounds of its compression function. The latter was the best known analysis on the compression function as well as the one on the largest number of rounds so far. In this paper, we extend the compression function results to get a distinguisher on 7 out of 8 rounds using rebound techniques. We also present the first 5-round collision attack on the ECHO-256 hash function.

Keywords: hash function, cryptanalysis, rebound attack, collision attack, distinguisher

1 Introduction

ECHO-256 [1] is the 256-bit version of one of the second-round candidates of the SHA-3 competition. It is an AES-based hash function that has been the subject of many studies. Currently, the best known analysis of ECHO-256 are a distinguisher on the full 8-round internal permutation proposed in [13] and improved in [10]. Furthermore, a 4-round collision attack of the compression function has been presented in [4]. A previous analysis due to Schläffer in [14] has been shown to be incorrect in [4], but it introduced an alternative description of the ECHO round-function, which has then been reused in several analyses, including this paper. The best results of this paper are a collision attack on the hash function reduced to 5 rounds and a distinguisher of the compression function on 7 rounds. Additionally, we cover two more attacks in the Appendix. The complexities of previous results and our proposed attacks are reported in Table 1.

Apart from the improved attacks on ECHO-256, this paper also covers a number of new techniques. The merging process of multiple inbound phases has been improved to find solutions also for the hash function, where much less freedom is available in the chaining input. For the hash function collision attack on 5 rounds, we use subspace differences which collide with a high probability at the output of the hash function. Additionally, we use multiple phases also in the outbound part to reduce the overall complexity of the attacks. For the 7 round compression function distinguisher, we use the new techniques and algorithms introduced in [10, 11].

Outline. The paper is organized as follows. In Section 2, we describe the 256-bit version of the ECHO hash function and detail an alternative view that has already been used in several analysis [4,14]. In particular, we emphasize the SuperMixColumns and SuperSBox

*This work was supported in part by the French ANR project SAPHIR II, by the French *Délégation Générale pour l'Armement* (DGA), by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322, by the European Commission through the ICT Programme under Contract ICT-2007-216646 ECRYPT II, by the Austrian Science Fund (FWF), project P21936 and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

Table 1: Best known cryptanalysis results on ECHO-256.

Rounds	Time	Memory	Generic	Type	Reference
8	2^{182}	2^{37}	2^{256}	Internal Permutation Distinguisher	[13]
8	2^{151}	2^{67}	2^{256}	Internal Permutation Distinguisher	[10]
4	2^{52}	2^{16}	2^{256}	Compression Function Collision	[4]
4	2^{64}	2^{64}	2^{128}	Hash Function Collision	Section B
5	2^{112}	$2^{85.3}$	2^{128}	Hash Function Collision	Section 3
6*	2^{160}	2^{128}	2^{256}	Compression Function Collision	Section A
6	2^{193}	2^{128}	2^{256}	Compression Function Collision	Section 4
7*	2^{160}	2^{128}	2^{240}	Compression Function Distinguisher	Section A
7	2^{193}	2^{128}	2^{240}	Compression Function Distinguisher	Section 4

* with chosen salt

transformations that ease the analysis. In Section 3, we provide a collision attack on this hash function reduced to 5 rounds and a distinguisher of the 7-round compression function in Section 4.

2 ECHO-256 description

ECHO is an iterated hash function and the compression function of ECHO updates an internal state described by a 16×16 matrix of GF $\left(2^8\right)$ 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 BigSubWords, 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).

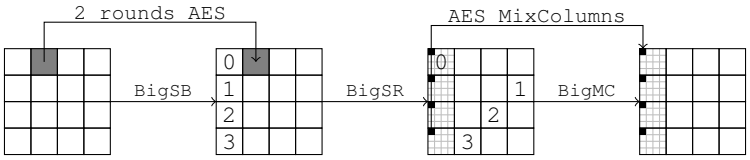

Figure 1: One round of the ECHO permutation. Each of the 16 cells is an AES state.

At the end of the permutation, the BigFinal operation adds the current state to the initial one (feed-forward) and, in the case of ECHO-256, 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 [6]. Since we are not using any property relying on symmetry and adding constants does not change differences, we omit these steps in the following.

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 and round function, but inject messages of size 1536 or 1024 bits respectively in the compression function. Note that the message is padded by adding a single 1 followed by zeros to fill up the last message block. The last 18 bytes of the last message block always contain the 2-byte hash output

size, followed by the 16-byte message length. 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 $S = [C_0C_1C_2C_3]$ 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 that one round, among the ten ones, of the AES-128 permutation is the succession of four transformations: SubBytes (**SB**), ShiftRows (**SR**), MixColumns (**MC**) and AddRoundKey (**AK**). We refer to the original publication [15] for further details.

Notations. We consider each state in the ECHO internal permutation, namely after each elementary transformations. We start with S_0, where the IV and the message are combined and end the first round after eight transformations in S_8. To refer to the AES-state at row i and column j of a particular ECHO-state S_n, we use the notation $S_n[i, j]$. Additionally, we introduce *column-slice* to refer to a thin column of size 16×1 of the ECHO state. The process of merging two lists L_1 and L_2 into a new list L is denoted $L = L_1 \bowtie L_2$. In the event that the merging should be done under some relation t, we use the operator $\bowtie_{|t|}$, where $|t|$ represents the size of the constraint to be verified in bits. Finally, in an AES-state, we consider four diagonals (from 0 to 3): diagonal $j \in [0, 3]$ will be the four elements $(i, i + j \pmod 4)$, with $i \in [0, 3]$.

2.1 Alternative description

For an easier description of some of the following attacks, we use an equivalent description of one round of the ECHO permutation. First, we swap the BigShiftRows transformation with the MixColumns transformation of the second AES round. Second, we swap SubBytes with ShiftRows of the first AES round. Swapping these operations does not change the computational result of ECHO and similar alternative descriptions have already been used in the analysis of AES. Hence, one round of ECHO results in the two transformations SuperSBox (SB-MC-SB) and SuperMixColumns (MC-BMC), which are separated just by byte-shuffling operation. The SuperSBox has first been analyzed by Daemen and Rijmen in [2] to study two rounds of AES and has been independently used by Lamberger et al. in [5] and Gilbert and Peyrin in [12] to analyze AES-based hash functions. The SuperMixColumns has been first introduced by Schläffer in [14] and reused in [4]. We refer to those articles for further details as well.

3 Attack on the 5-round ECHO-256 Hash Function

In this section, we use a sparse truncated differential path and the properties of SuperMixColumns to get a collision attack on 5 rounds of the ECHO-256 hash function. The resulting complexity is 2^{112} with memory requirements of $2^{85.3}$. We first describe the truncated differential path (a truncated differential path only considers whether a byte of the state is active or not) and show how to find conforming input pairs. Due to the sparse truncated differential path, we are able to apply a rebound attack with multiple inbound phases to ECHO. Since at most one fourth of each ECHO state is active, we have enough freedom for two inbound phases and are also able to fully control the chaining input of the hash function.

3.1 The Truncated Differential Path

In the attack, we use two message blocks where the first block does not contain differences. For the second message block, we use the truncated differential path given in Figure 2. We use colors (red, yellow, green, blue, cyan) to describe different phases of the attack and to denote their resulting solutions. Active bytes are denoted by black color, and active AES states contain at least one active byte. Hence, the sequence of active AES states for each round of ECHO is as follows:

$$5 \xrightarrow{r_1} 16 \xrightarrow{r_2} 4 \xrightarrow{r_3} 1 \xrightarrow{r_4} 4 \xrightarrow{r_5} 16.$$

Note that in this path, we keep the number of active bytes low, except for the beginning and end. Therefore, we have enough freedom to find many solutions. We do not allow differences in the chaining input (blue) and in the padding (cyan). The last 16 bytes of the padding contain the message length and the two bytes above contain size of the hash function output. Note that the AES states containing the chaining values (blue) and padding (cyan) do not get mixed with other AES states until the first BigMixColumns transformation. Since the lower half of the state (row 2 and 3) is truncated to compute the final hash value, we force all differences to be in the lower half of the message: the feed-forward will then preserve that property.

3.2 Colliding Subspace Differences

In the following, we show that the resulting output differences after 5 rounds lie in a vector space of reduced dimension. This can be used to construct a distinguisher for 5 rounds of the ECHO-256 hash function. However, due to the low dimension of the output vector space, we can even extend this subspace distinguisher to get a collision attack on 5 rounds of the ECHO-256 hash function.

First, we need to determine the dimension of the vector space at the output of the hash function. In general, the dimension of the output vector space is defined by the number of active bytes prior to the linear transformations in the last round (16 active bytes after the last SubBytes), combined with the number of active bytes at the input due to the feed-forward (0 active bytes in our case). This would results in a vector space dimension of $(16+0)\times 8 = 128$. However, a weakness in the combined transformations SuperMixColumns, BigFinal and the output truncation reduces the vector space to a dimension of 64 at the output of the hash function for the truncated differential path in Figure 2.

We can move the BigFinal function prior to SuperMixColumns, since BigFinal is a linear transformation and the same linear transformation $\mathbf{M_{SMC}}$ is applied to all columns in SuperMixColumns. Then, we get 4 active bytes at the same position in each AES state of the 4 resulting column-slices. To each active column-slice C_{16}, we first apply the SuperMixColumns multiplication with $\mathbf{M_{SMC}}$ and then, a matrix multiplication using $\mathbf{M_{trunc}} = [I_8 \mid 0_8]$ which truncates the lower 8 rows. Since only 4 bytes are active in C_{16}, these transformations can be combined into a transformation using a reduced 4×8 matrix $\mathbf{M_{comb}}$ applied to the reduced input C_4, which contains only the 4 active bytes of C_{16}:

$$\mathbf{M_{trunc}} \cdot \mathbf{M_{SMC}} \cdot C_{16} = \mathbf{M_{comb}} \cdot C_4,$$

The multiplication with zero differences of C_{16} removes 12 columns of $\mathbf{M_{SMC}}$ while the truncation removes 8 rows of $\mathbf{M_{SMC}}$. For example, considering the first active column-slice

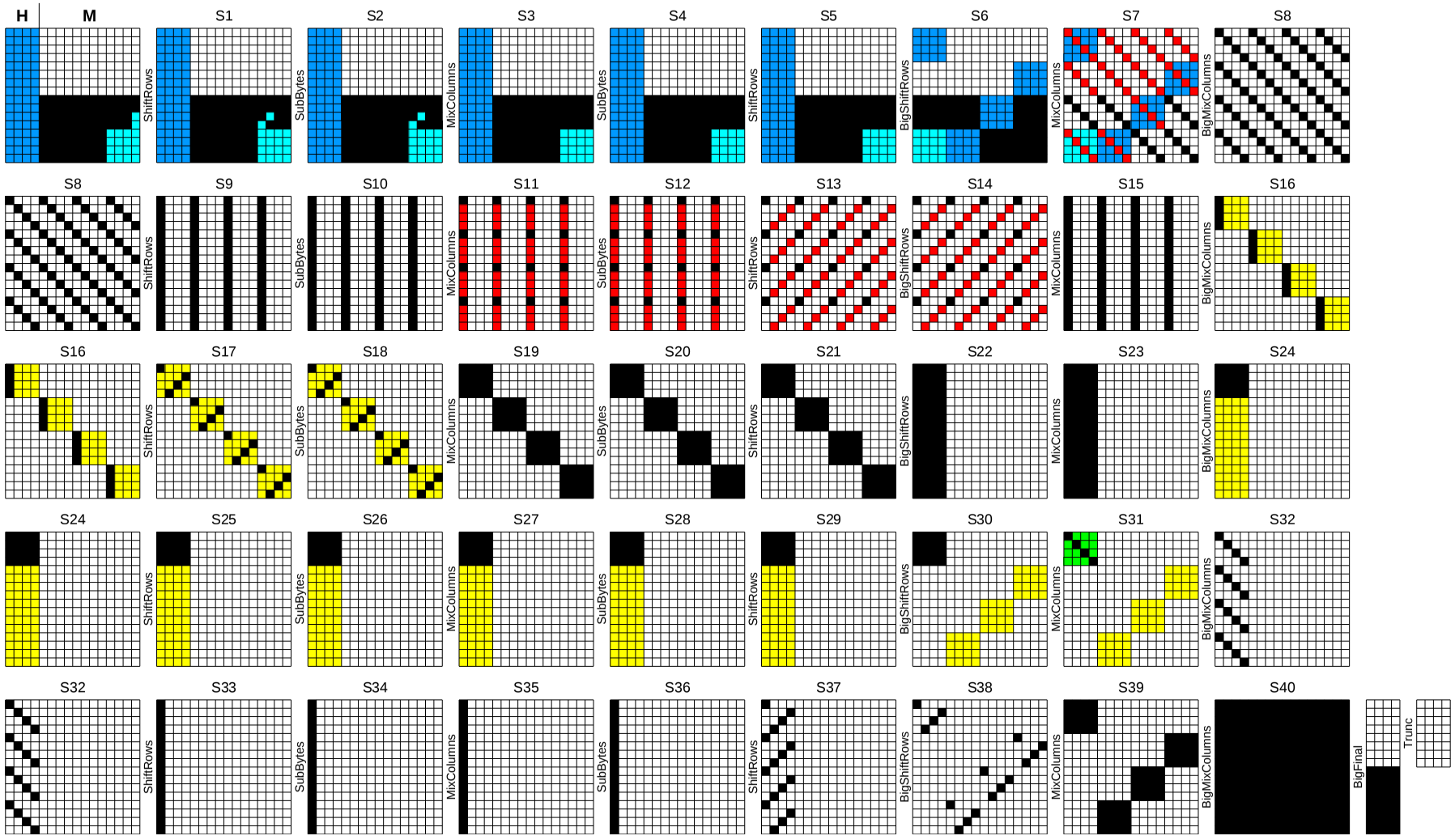

Figure 2: The truncated differential path to get a collision for 5 rounds of ECHO-256. Black bytes are active, blue and cyan bytes are determined by the chaining input and padding, red bytes are values computed in the red inbound phase, yellow bytes in the yellow inbound phase and green bytes in the outbound phase.

leads to:

$$\mathbf{M_{trunc}} \cdot \mathbf{M_{SMC}} \cdot \begin{bmatrix} a\,0\,0\,0\,b\,0\,0\,0\,c\,0\,0\,0\,d\,0\,0\,0 \end{bmatrix}^{\mathrm{T}} = \underbrace{\begin{bmatrix} 4&6&2&2&6&5&3&3 \\ 2&3&1&1&4&6&2&2 \\ 2&3&1&1&2&3&1&1 \\ 6&5&3&3&2&3&1&1 \end{bmatrix}^{\mathrm{T}}}_{\mathbf{M_{comb}}} \cdot \begin{bmatrix} a\,b\,c\,d \end{bmatrix}^{\mathrm{T}}$$

Analyzing the resulting matrix $\mathbf{M_{comb}}$ for all four active column-slices shows that in each case, the rank of $\mathbf{M_{comb}}$ is two, and not four. This reduces the dimension of the vector space in each active column-slice from 32 to 16. Since we have four active columns, the total dimension of the vector space at the output of the hash function is 64. Furthermore, column $i \in \{0, 1, 2, 3\}$ of the output hash value depends only on columns $4i$ of state S_{38}. It follows that the output difference in the first column $i = 0$ of the output hash value depends only on the four active differences in columns 0, 4, 8 and 12 of state S_{38}, which we denote by a, b, c and d. To get a collision in the first column of the hash function output, we get the following linear system of equations:

$$\mathbf{M_{comb}} \cdot \begin{bmatrix} a\,b\,c\,d \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}^{\mathrm{T}}.$$

Since we cannot control the differences a, b, c and d in the following attack, we need to find a solution for this system of equations by brute-force. However, the brute-force complexity is less than expected due to the reduced rank of the given matrix. Since the rank is two, 2^{16} solutions exist and a random difference results in a collision with a probability of 2^{-16} instead of 2^{-32} for the first output column. Since the rank of all four output column matrices is two, we get a collision at the output of the hash function with a probability of $2^{-16\times 4} = 2^{-64}$ for the given truncated differential path.

3.3 High-Level Outline of the Attack

To find input pairs according to the truncated differential path given in Figure 2, we use a rebound attack [8] with multiple inbound phases [5, 7]. The main advantage of multiple inbound phases is that we can first find pairs for each inbound phase independently and then, connect (or merge) the results. Furthermore, we also use multiple outbound phases and separate the merging process into three different parts which can be solved mostly independently:

1. **First Inbound between S_{16} and S_{24}:** find 2^{96} partial pairs (yellow and black bytes) with a complexity of 2^{96} in time and 2^{64} memory.
2. **First Outbound between S_{24} and S_{31}:** filter the previous solutions to get 1 partial pair (green, yellow and black bytes) with a complexity of 2^{96} in time and 2^{64} memory.
3. **Second Inbound between S_7 and S_{14}:** find 2^{32} partial pairs (red and black) for each of the first three BigColumns and 2^{64} partial pairs for the last BigColumn of state S_7 with a total complexity of 2^{64} in time and memory.
4. **First Part in Merging the Inbound Phases:** combine the 2^{160} solutions of the previous phases according to the 128-bit SuperMixColumns condition given in [4]. We get 2^{32} partial pairs (black, red, yellow and green bytes between state S_7 and S_{31}) with complexity 2^{96} in time and 2^{64} memory.
5. **Merge Chaining Input:** repeat from Step 1 for 2^{16} times to get 2^{48} solutions for the previous phases. Compute 2^{112} chaining values (blue) using 2^{112} random first message blocks. Merge these solutions according to the overlapping 20 bytes (red with blue/cyan) in state S_7 to get $2^{48} \times 2^{112} \times 2^{-160} = 1$ partial pair with complexity 2^{112} in time and 2^{48} memory.

6. **Second Part in Merging the Inbound Phases:** find one partial solution for the first two columns of state S_7 according to the 128-bit condition at SuperMixColumns between S_{14} and S_{16} with complexity 2^{64} in time and memory.
7. **Third Part in Merging the Inbound Phases:** find one solution for all remaining bytes (last two columns of state S_7) by fulfilling the resulting 192-bit condition using a generalized birthday attack with 4 lists. The complexity is 2^{64} in time and memory to find one solution, and $2^{85.3}$ in time and memory to find 2^{64} solutions [16].
8. **Second Outbound Phase to get Collisions:** in a final outbound phase, the resulting differences at the output of the hash function collide with a probability of 2^{-64} and we get one collision among the 2^{64} solutions of the previous step.

The total time complexity of the attack is 2^{112} and determined by Step 5; the memory complexity is $2^{85.3}$ and determined by Step 7.

3.4 Details of the Attack

In this section, we describe the each phase of the collision attack on 5 rounds of ECHO-256 in detail. Note that some phases are also reused in the attacks on the compression function of Section 4.

First Inbound between S_{16} and S_{24}. We first search for internal state pairs conforming to the truncated differential path in round 3 (yellow and black bytes). We start the attack by choosing differences for the active bytes in state S_{16} such that the truncated differential path of SuperMixColumns between state S_{14} and S_{16} is fulfilled (Section 2.1). We compute this difference forward to state S_{17} through the linear layers.

We continue with randomly chosen differences of state S_{24} and compute backwards to state S_{20}, the output of the SuperSBoxes. Since we have 64 active S-boxes in this state, the probability of a differential is about $2^{-1\times 64}$. Hence, we need 2^{64} starting differences but get 2^{64} solutions for the inbound phase in round 3 (see [8]). We determine the right pairs for each of the 16 SuperSBox between state S_{17} and S_{20} independently. Using the Differential Distribution Table of the SuperSBoxes, we can find one right pair with average complexity one. In total, we compute 2^{96} solutions for this inbound phase with time complexity 2^{96} and memory complexity of at most 2^{64}. For each of these pairs, differences and values of all yellow and black bytes in round 3 are determined.

Second Outbound between S_{24} and S_{31}. In the outbound phase, we ensure the propagation in round 4 of the truncated differential path by propagating the right pairs of the previous inbound phase forwards to state S_{31}. With a probability of 2^{-96}, we get four active bytes after MixColumns in state S_{31} (green) conforming to the truncated path. Hence, among the 2^{96} right pairs of the inbound phase between S_{16} and S_{24} we expect to find one such right pair.

The total complexity to find this partial pair between S_{16} and S_{31} is then 2^{96}. Note that for this pair, the values and differences of the yellow, green and black bytes between states S_{16} and S_{31} can be determined. Furthermore, note that for any choice of the remaining bytes, the truncated differential path between state S_{31} and state S_{40} is fulfilled.

Second Inbound between S_7 and S_{14}. Here, we search for many pairs of internal states conforming to the truncated differential path between states S_7 and S_{14}. Note that we can independently search for pairs of each BigColumn of state S_7, since the four

BigColumns stay independent until they are mixed by the following BigMixColumns transformation between states S_{15} and S_{16}. For each BigColumn, four SuperSBoxes are active and we need at least 2^{16} starting differentials for each one to find the first right pair.

The difference in S_{14} is already fixed due to the yellow inbound phase but we can still choose at least 2^{32} differences for each active AES state in S_7. Using the rebound technique, we can find one pair on average for each starting difference in the inbound phase. Then, we independently iterate through all 2^{32} starting differences for the first, second and third column and through all 2^{64} starting differences for the fourth column of state S_7. We get 2^{32} right pairs for each of the first three columns and 2^{64} pairs for the fourth column. The complexity to find all these pairs is 2^{64} in time and memory.

For each resulting right pair, the values and differences of the red and black bytes between states S_7 and S_{14} can be computed. Furthermore, the truncated differential path in backward direction, except for two cyan bytes in the first states, is fulfilled. In the next phase, we partially merge the right pairs of the yellow and red inbound phase. But first, we recall the conditions for this merge.

First Part in Merging the Inbound Phases. For each pair of the previous two phases, the values of the red, yellow and black bytes of state S_{14} and S_{16} are fixed. These two states are separated by the linear SuperMixColumns transformation: taking the first column-slice as an example, we get

$$\begin{aligned}\mathbf{M_{SMC}} \cdot [A_0\ x_0\ x_1\ x_2\ A_1\ x_3\ x_4\ x_5\ A_2\ x_6\ x_7\ x_8\ A_3\ x_9\ x_{10}\ x_{11}]^{\mathrm{T}} \\ = [B_0\ B_1\ B_2\ B_3\ y_0\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7\ y_8\ y_9\ y_{10}\ y_{11}]^{\mathrm{T}},\end{aligned}$$

where $\mathbf{M_{SMC}}$ is the SuperMixColumns transformation matrix, A_i the input bytes determined by the red inbound phase and B_i the output bytes determined by the yellow inbound phase. All bytes x_i and y_i are free to choose. As shown by Jean and Fouque [4], we only get a solution with probability 2^{-8} for each column-slice due to the low rank of the $\mathbf{M_{SMC}}$ matrix. In [4] (Appendix A), the 8-bit condition for that particular column-slice that ensures the system to have solutions has been derived and is given as follows:

$$2 \cdot A_0 + 3 \cdot A_1 + A_2 + A_3 = 14 \cdot B_0 + 11 \cdot B_1 + 13 \cdot B_2 + 9 \cdot B_3. \quad (1)$$

Similar 8-bit conditions exist for all 16 column-slices. In total, each right pair of the two (independent) inbound phases results in a 128-bit condition on the whole SuperMixColumns transformation between states S_{14} and S_{16}.

Remember that we have constructed one pair for the yellow inbound phase and in total, $2^{32} \times 2^{32} \times 2^{32} \times 2^{64} = 2^{160}$ pairs for the red inbound phase. Among these 2^{160} pairs, we expect to find 2^{32} right pairs which also satisfy the 128-bit condition of the SuperMixColumns between states S_{14} and S_{16}. In the following, we show how to find all these 2^{32} pairs with a complexity of 2^{96}.

First, we combine the $2^{32} \times 2^{32} = 2^{64}$ pairs determined by the two first BigColumns of state S_7 in a list L_1 and the $2^{32} \times 2^{64} = 2^{96}$ pairs determined by the last two BigColumns of state S_7 in a list L_2. Note that the pairs in these two lists are independent. Then, we separate Equation (1) into terms determined by L_1 and terms determined by L_2:

$$2 \cdot A_0 + 3 \cdot A_1 = A_2 + A_3 + 14 \cdot B_0 + 11 \cdot B_1 + 13 \cdot B_2 + 9 \cdot B_3. \quad (2)$$

We apply the left-hand side to the elements of L_1 and the right-hand side to elements of L_2 and sort L_1 according to the bytes to be matched.

Then, we can simply merge (join) these lists to find those pairs which satisfy the 128-bit condition imposed by the SuperMixColumns and store these results in list $L_{12} = L_1 \bowtie_{128} L_2$. This way, we get $2^{64} \times 2^{96} \times 2^{-128} = 2^{32}$ right pairs with a total complexity of 2^{96}. We note that the memory requirements can be reduced to 2^{64} if we do not store the elements of L_2 but compute them online. The resulting 2^{32} solutions are partial right pairs for the black, red, yellow and green bytes between state S_7 and S_{31}.

Merge Chaining Input. Next, we need to merge the 2^{32} results of the previous phases with the chaining input (blue) and the bytes fixed by the padding (cyan). The chaining input and padding overlap with the red inbound phase in state S_7 on $5 \times 4 = 20$ bytes. This results in a 160-bit condition on the overlapping blue/cyan/red bytes. To find a pair verifying this condition, we first generate 2^{112} random first message blocks, compute the blue bytes of state S_7 and store the results in a list L_3.

Additionally, we repeat 2^{16} times from the yellow inbound phase but with other starting points[4] in state S_{24}. This way, we get $2^{16} \times 2^{32} = 2^{48}$ right pairs for the combined yellow and red inbound phases, which also satisfy the 128-bit condition of SuperMixColumns between states S_{14} and S_{16}. The complexity is $2^{16} \times 2^{96} = 2^{112}$. We store the resulting 2^{48} pairs in list L_{12}.

Next, we merge the lists according to the overlapping 160-bits ($L_{12} \bowtie_{160} L_3$) and get $2^{48} \times 2^{112} \times 2^{-160} = 1$ right pair. If we compute the 2^{112} message blocks of list L_3 online, the time complexity of this merging step is 2^{112} with memory requirements of 2^{48}. For the resulting pair, all differences between states S_4 and S_{33} and all colored byte values (blue, cyan, red, yellow, green and black) between states S_0 and S_{31} can be determined.

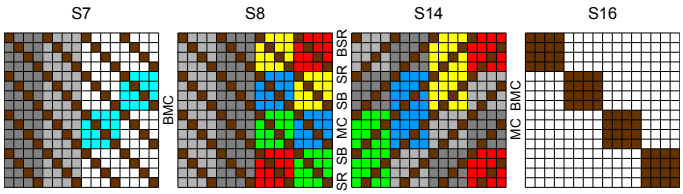

Figure 3: States used to merge the two inbound phases with the chaining values. The merge inbound phase consists of three parts. Brown bytes show values already determined (first part) and gray values are chosen at random (second part). Green, blue, yellow and red bytes show independent values used in the generalized birthday attack (third part) and cyan bytes represent values with the target conditions.

Second Part in Merging Inbound Phases. To completely merge the two inbound phases, we need to find according values for the white bytes. We use Figure 3 to illustrate the second and third part of the merge inbound phase. In this figure, we only consider values and therefore, do not show active bytes (black). Furthermore, all brown and cyan bytes have already been chosen in one of the previous steps. In the second part of the merge inbound phase, we only choose values for the gray and light-gray bytes. All other colored bytes show steps of the following merging phase.

We first choose random values for all remaining bytes of the two first columns in state S_7 (gray and light-gray) and independently compute the columns forward to state S_{14}. Note that we need to try $2^{2\times8+1}$ values for AES state $S_7[2,1]$ to also match the 2-byte (cyan) and 1-bit padding at the input in AES state $S_0[2,3]$. Then, all gray, light-gray,

[4] Until now, we have chosen only 2^{96} out of 2^{128} differences for this state.

cyan and brown bytes have already been determined either by an inbound phase, chaining value, padding or just by choosing random values for the remaining free bytes of the two first columns of S_7. However, all white, red, green, yellow and blue bytes are still free to choose.

By considering the linear SuperMixColumns transformation, we observe that in each column-slice, 14 out of 32 input/output bytes are already fixed and 2 bytes are still free to choose. Hence, we expect to get 2^{16} solutions for this linear system of equations. Unfortunately, also for the given position of already determined 14 bytes, the linear system of equations does not have a full rank. Again, we can determine the resulting system using the matrix $\mathbf{M_{SMC}}$ of SuperMixColumns. As an example, for the first column-slice, the system is given as follows:

$$\begin{aligned}\mathbf{M_{SMC}} \cdot [&A_0\ L_0\ L_1\ L_2\ A_1\ L'_0\ L'_1\ L'_2\ A_2\ x_6\ x_7\ x_8\ A_3\ x_9\ x_{10}\ x_{11}]^{\mathrm{T}} \\ &= [B_0\ B_1\ B_2\ B_3\ y_0\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7\ y_8\ y_9\ y_{10}\ y_{11}]^{\mathrm{T}}.\end{aligned}$$

The free variables in this system are $x_6, \ldots, x_{11}$ (green). The values A_0, A_1, A_2, A_3, B_0, B_1, B_2, B_3 (brown) have been determined by the first or second inbound phase and the values L_0, L_1, L_2 (light-gray) and L'_0, L'_1, L'_2 (gray) are determined by the choice of arbitrary values in state S_7. We proceed as before and determine the linear system of equations which needs to have a solution:

$$\begin{bmatrix} 3\,1\,1\,3\,1\,1 \\ 2\,3\,1\,2\,3\,1 \\ 1\,2\,3\,1\,2\,3 \\ 1\,1\,2\,1\,1\,2 \end{bmatrix} \cdot \begin{bmatrix} x_6\ x_7\ x_8\ x_9\ x_{10}\ x_{11} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} c_0\ c_1\ c_2\ c_3 \end{bmatrix}^{\mathrm{T}}. \tag{3}$$

The resulting linear 8-bit equation to get a solution for this system can be separated into terms depending on values of L_i and on L'_i, and we get $f_1(L_i) + f_2(L'_i) + f_3(a_i, b_i) = 0$, where f_1, f_2 and f_3 are linear functions. For all other 16 column-slices and fixed positions of gray bytes, we get matrices of rank three as well. In total, we get 16 8-bit conditions and the probability to find a solution for a given choice of gray and light-gray values in states S_{14} and S_{16} is 2^{-128}. However, we can find a solution to these linear equations using the birthday effect and a meet-in-the-middle attack with a complexity of 2^{64} in time and memory.

We start by choosing 2^{64} values for each of the first (gray) and second (light-gray) BigColumns in state S_7. We compute these values independently forward to state S_{14} and store them in two lists L and L'. We also separate all equations of the 128-bit condition into parts depending only on values of L and L'. We apply the resulting functions f_1, f_2, f_3 to the elements of lists L_i and L'_i, and merge two lists $L \bowtie_{128} L'$ using the birthday effect.

Third part in Merging Inbound Phases. We continue with a generalized birthday match to find values for all remaining bytes of the state (blue, red, green, yellow, cyan and white of Figure 3). For each column in state S_{14}, we independently choose 2^{64} values for the green, blue, yellow and red columns, and compute them independently backward to S_8. We need to match the values of the cyan bytes of state S_7, which results in a condition on 24 bytes or 192 bits. Since we have four independent lists with 2^{64} values in state S_8, we can use the generalized birthday attack [16] to find one solution with a complexity of $2^{192/3} = 2^{64}$ in time and memory.

In more detail, we need to match values after the BigMixColumns transformation in the backward direction. Hence, we first multiply each byte of the four independent lists by the four multipliers of the InvMixColumns transformation. Then, we get 24 equations

containing only XOR conditions on bytes between the target value and elements of the four independent lists, which can be solved using a generalized birthday attack.

To improve the average complexity of this generalized birthday attack, we can start with larger lists for the green, blue, yellow and red columns in state S_{14}. Since we need to match a 192-bit condition, we can get $2^{3 \cdot x} \times 2^{-192} = 2^x$ solutions with a time and memory complexity of $\max\{2^{64}, 2^x\}$ (see [16] for more details). Note that we can even find solutions with an average complexity of 1 using lists of size 2^{96}. Each solutions of the generalized birthday match results in a valid pair conforming to the whole 5-round truncated differential path.

Second Outbound Phase to get Collisions. For the collision attack on 5 rounds, we start the generalized birthday attack of the previous phase with lists of size $2^{85.3}$. This results in $2^{3 \cdot 85.3} \times 2^{-192} = 2^{64}$ solutions with a time and memory complexity of $2^{85.3}$, or with an average complexity of $2^{21.3}$ per solution. These solutions are propagated outwards in a second, independent outbound phase. Since the differences at the output collide with a probability of 2^{-64}, we expect to find one pair which collides at the output of the hash function. The time complexity is determined by merging the chaining input and the memory requirements by the generalized birthday attack. To summarize, the complexity to find a collision for 5 rounds of the ECHO-256 hash function is given by about 2^{112} compression function evaluations with memory requirements of $2^{85.3}$.

4 Distinguisher on the 7-round ECHO-256 Compression Function

In this section, we detail our distinguisher on 7 rounds in the known-salt model. First, we show how to obtain partial solutions that verify the path from the state S_6 to S_{23} with an average complexity of 2^{64} in time, as we obtain 2^{64} solutions with a cost of 2^{128}. These partial solutions determine also the values of the blue bytes (in Figure 4). Next, we show how to do the same for the yellow part of the path from S_{30} to S_{47}. Finally, we explain how to merge these partial solutions for finding one that verifies the whole path.

4.1 Finding pairs between S_6 and S_{23}

We explain here how to find 2^{64} solutions for the blue part with a cost of 2^{128} in time and 2^{64} in memory. This is done with a stop-in-the-middle algorithm similar to the one presented in [11] for improving the time complexity of the ECHO-256 distinguisher. This algorithm has to be adapted to this particular situation, where all the active states belong to the same BigColumn.

We start by fixing the difference in S_8 to a chosen value, so that the transition between S_6 and S_8 is verified. We fix the difference in the active diagonals of the two AES-states $S_{23}[0,0]$ and $S_{23}[3,1]$ to a chosen value.

From state S_8 to S_{13}, we have four different SuperSBox groups involved in the active part. From states S_{16} to S_{22}, we have 4×4 SuperSBox groups involved (4 per active AES state). Those 16 groups, as well as the 4 previous ones, are completely independent from S_{16} to S_{22} (respectively from S_8 to S_{13}). From the known difference in S_8, we build four lists of values and differences in S_{13}: each list corresponds to one of the four SuperSBox groups. Each list is of size 2^{32} because once we know the input difference, we try all the possible 2^{32} possible values and then we can compute the values and differences in S_{13} (as

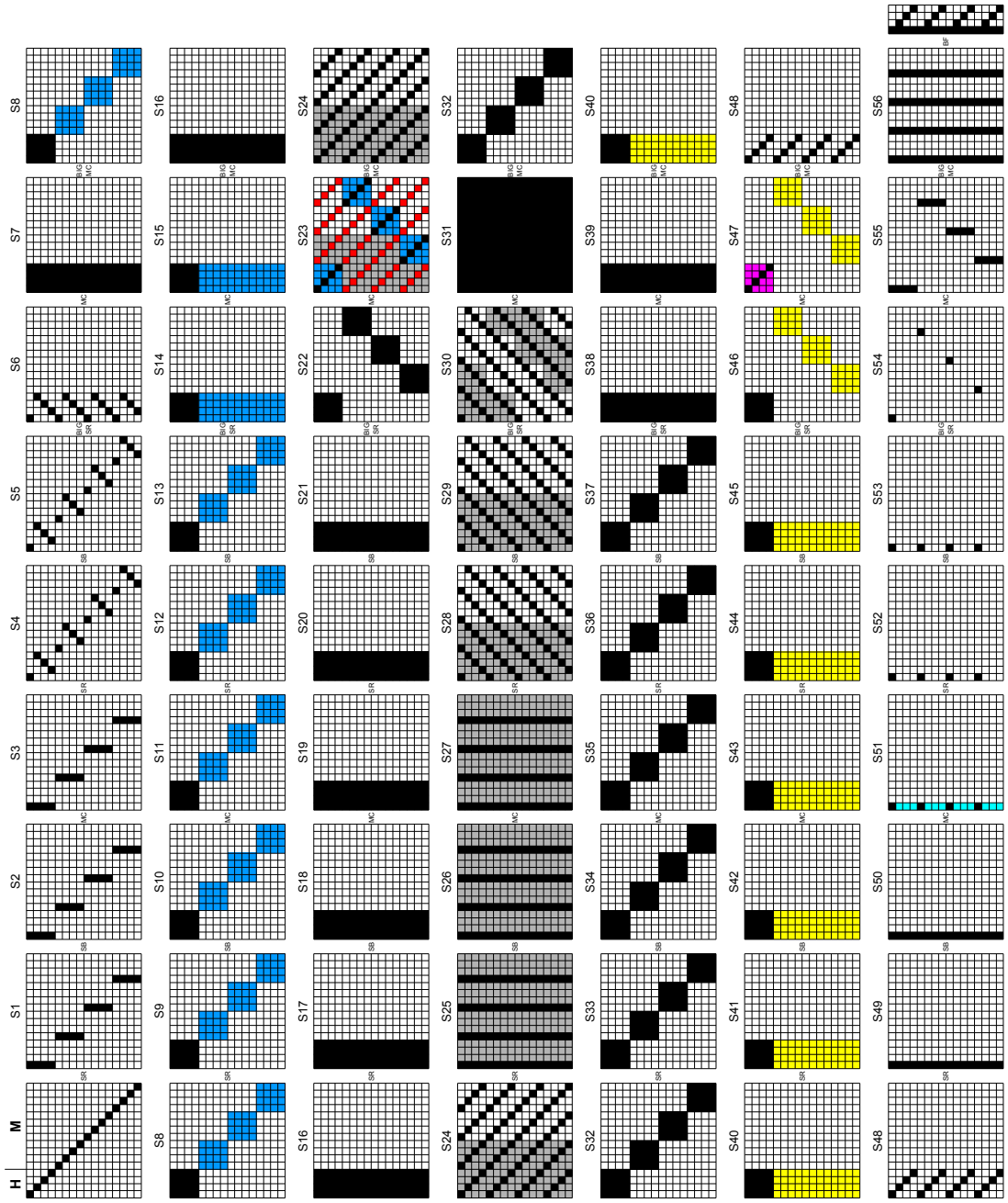

Figure 4: Differential path for the seven-round distinguisher.

we said, the four groups are independent in this part of the path). In the sequel, those lists are denoted L_A^0, L_A^1, L_A^2 and L_A^3.

There are 64 bits of differences not yet fixed in S_{23}. Each active diagonal only affects the AES state where it is in, so we can independently consider 2^{32} possible differences for one diagonal and 2^{32} differences for the other. We can now build the 16 lists corresponding to the 16 SuperSBox groups as we did before, but considering that: the 8 lists corresponding to 8 groups of the two AES states $S_{16}[0,0]$ and $S_{16}[3,0]$, as they have their differences in S_{22} already fixed, have a size of 2^{32} (corresponding to the possible values for each group). These are the lists $L_{0,0}^i$ and $L_{3,0}^i$, with $i \in [0,3]$ that represents the ith diagonal of the

state. But the lists $L^i_{1,0}, L^i_{2,0}$, with $i \in [0,3]$, as they do not have yet the difference fixed, have a size of 2^{32+32} each, as we can consider the 2^{32} possible differences for each not fixed diagonal independently. Next, we go through the 2^{64} possible differences of the two first diagonals (diagonals 0 and 1) of the active AES state in S_{15}. For each one of these 2^{64} possible differences:

- The associated differences in the two same diagonals in the four active AES states of S_{16} can be computed. Consequently, we can check in the previously computed ordered lists $L^i_{j,0}$ with $j \in [0,3]$ and $i \in [0,1]$ where we find this difference[5]. For $j \in \{0,3\}$, on average, we obtain one match on each one of the lists $L^0_{0,0}$, $L^1_{0,0}$, $L^0_{3,0}$ and $L^1_{3,0}$. For $j \in \{1,2\}$, we obtain 2^{32} matches, one for each of the 2^{32} possible differences in the associated diagonals in S_{23}. That is 2^{32} matches for $L^0_{1,0}$ and $L^1_{1,0}$, where a pair of values formed by one element of each list is only valid if they were generated from the same difference in S_{23}. Consequently, we can construct the list $L^{0,1}_{1,0}$ of size 2^{32} where we store the values and differences of those two diagonals in the AES state $S_{16}[1,0]$ as well as the difference in S_{23} from which they were generated. Repeating the process for $L^0_{2,0}$ and $L^1_{2,0}$, we construct the list $L^{0,1}_{2,0}$ of size 2^{32}. We can merge the lists $L^{0,1}_{1,0}$, $L^{0,1}_{2,0}$ and the four fixed values for differences and values obtained from the matches in the lists $L^0_{0,0}$, $L^1_{0,0}$, $L^0_{3,0}$ and $L^1_{3,0}$, corresponding to the AES states $S_{16}[0,0]$ and $S_{16}[3,0]$. This generates the list $L^{0,1}$ of size 2^{64}. Each element of this list contains the values and differences of the two diagonals 0 and 1 of the four active AES states in S_{16}. As we have all the values for the two first diagonals in the four AES states, for each one of these elements, we compute the values in the two first diagonals of the active state in S_{15} by applying the inverse of BigMixColumns. We order them according to these values.
- Next, we go through the 2^{64} possible differences of the two next diagonals (diagonals 2 and 3) of the active AES state in S_{15}. For each one of these 2^{64} possible differences:
  - All the differences in the AES state $S_{13}[0,0]$ are determined. We check in the lists L^0_A, L^1_A, L^2_A and L^3_A if we find a match for the differences. We expect to find one in each list and this determines the values for the whole state $S_{15}[0,0]$ (as the elements in these lists are formed by differences and values). This means that the value of the active AES state in S_{15} is also completely determined. This way, we can check in the previously generated list $L^{0,1}$ if the correct value for the two diagonals 0 and 1 appears. We expect to find it once.
  - As we have just found a valid element from $L^{0,1}$, it determines the differences in the AES states $S_{23}[1,0]$ and $S_{23}[2,0]$ that were not fixed yet. Now, we need to check if, for those differences in S_{23}, the corresponding elements in the four lists $L^i_{1,0}$, $L^i_{2,0}$ for $i \in [2,3]$ that match with the differences fixed in the diagonals 2 and 3 of S_{15} [6], satisfy the values in S_{15} that were also determined by the lists L^i_A. This occurs with probability 2^{-64}.

All in all, the time complexity of this algorithm is $2^{64} \cdot (2^{64}+2^{64}) = 2^{129}$ with a memory requirement of 2^{64}. The resulting expected number of valid pairs is $2^{64} \cdot 2^{64} \cdot 2^{64} \cdot 2^{-64} \cdot 2^{-64} = 2^{64}$.

4.2 Finding pairs between S_{30} and S_{47}

In quite the same way as the previous section, we can find solutions for the yellow part with an average cost of 2^{64}. To do so, we take into account the fact that the MixColumns

[5] i is either 0 or 1 because we are just considering the two first diagonals.

[6] We expect one match per list.

and BigMixColumns transformations commute. So, if we exchange their positions between states S_{39} and S_{40}, we only have one active AES state in S_{39}. We fix the differences in S_{47} and in two AES states, say $S_{32}[0,0]$ and $S_{32}[1,1]$, and we still have 2^{32} possible differences for each of the two remaining active AES states in S_{32}. Then, the lists L_A^i are generated from the end and contain values and differences from S_{40}. Similarly, the lists $L_{j,j}^i$ contain values and differences from S_{38}. We can apply the same algorithm as before and obtain 2^{64} solutions with a cost of 2^{128} in time and 2^{64} in memory.

4.3 Merging solutions

In this section, we explain how to get a solution for the whole path. As explained in our Section 4.1, we can find 2^{64} solutions for the blue part, that have the same difference for the active AES states of columns 0 and 1 in S_{23}. We obtain 2^{64} solutions from a fixed value for the differences in S_8 and the AES states $S_{23}[0,0]$ and $S_{23}[3,1]$. Repeating this process for the 2^{32} possible differences in S_8, we obtain in total 2^{96} solutions for the blue part with the same differences in the columns 0 and 1 in S_{23}. The cost of this step is 2^{160} in time and 2^{96} in memory.

The same way, using the algorithm explained in Section 4.2, we can also find 2^{96} solutions for the yellow part, that have the same difference value for the AES active states of columns 0 and 1 in S_{32} (we fix the difference value of this two columns in S_{32}, and we try all the 2^{32} possible values for the difference in S_{47}). The cost of this step is also 2^{160} in time and 2^{96} in memory.

Now, from the partial solutions obtained in the previous steps, we want to find a solution that verifies the whole differential path. For this, we want to merge the solutions from S_{23} with the solutions from S_{32}. We know that the differences of the columns 0,1 of S_{24} and S_{31} are fixed. Hence, from S_{24} to S_{31}, there are four AES states for which we know the input difference and the output difference, as they are fixed[7]. We can then apply a variant of the SuperSBox [3,5] technique in these four AES states: it fixes the possible values for the active diagonals of those states.

The differences in the other four AES states in S_{24} that are fixed are associated to other differences that are not fixed [8]. There are 2^{64} possible differences, each one associated to 2^{32} solutions for S_{32}-S_{47} given by the solutions that we found in the second step. For each one of these 2^{64} possible differences, one possible value is associated by the SuperSBox. When computing backwards these values to state S_{24}, as we have also the values for the other four AES states of the columns 0 and 1 that are also fixed (in the third step), we can compute the values for these two columns in S_{23}, and we need 32×2 bit conditions to be verified on the values. So for each one of the 2^{64} possible differences in S_{31}, we obtain $2^{96-64} = 2^{32}$ that verify the conditions on S_{23}. In total, we have $2^{64+32} = 2^{96}$ possible partial matches.

For each of the 2^{64} possible differences in S_{31}, its associated 2^{32} possible partial matches also need to verify the 128-bit condition in S_{30}-S_{32} at the SuperMixColumns layer [4] and the remaining 2×32 bit conditions on the values of S_{23}. Since for each of the 2^{64} differences we have 2^{32} possible associated values in S_{32}, the probability of finding a good pair is $2^{96-128-64+32} = 2^{-64}$.

If we repeat this merging procedure 2^{64} times, namely for 2^{32} differences in the columns 0 and 1 of S_{23} and for 2^{32} differences in the columns 0 and 1 of S_{32}, we should find a solution. We then repeat the procedure for the cross product of the 2^{32} solutions for each

[7] $S_{24}[0,0]$, $S_{24}[0,1]$, $S_{24}[1,1]$, $S_{24}[3,0]$ correspond to $S_{31}[0,0]$, $S_{31}[0,1]$, $S_{31}[1,0]$, $S_{31}[3,1]$, respectively.
[8] $S_{24}[1,0]$, $S_{24}[2,0]$, $S_{24}[2,1]$, $S_{24}[3,1]$ correspond to $S_{31}[1,3]$, $S_{31}[2,2]$, $S_{31}[2,3]$, $S_{31}[3,2]$.

side. As we do not want to compute them each time that we use them, as it would increase the time complexity, we can just store the $2^{64+32+32} = 2^{128}$ solutions for the first part and use the corresponding ones when needed, while the second part is computed in sequence. The complexity would be: $2^{192} + 2^{192} + 2^{96+64}$ in time and 2^{128} in memory. So far, we have found a partial solution for the differential part for rounds from S_6 to S_{48}. We still have the passive bytes to determine and the condition to pass from S_{50} to S_{51} to verify. This can be done exactly as in the second and third part of the merge inbound phase of Section 3.4 with no additional cost.

Moreover, since we can find x solutions with complexity $\max\{x, 2^{96}\}$ in time and 2^{96} memory for the (independent) merge inbound phase, we can get $x < 2^{193}$ solutions with time complexity $2^{193} + \max\{x, 2^{96}\} \sim 2^{193}$ and 2^{128} memory. We need only 2^{96} of these solutions to pass the probabilistic propagation in the last round from S_{50} to S_{51}. Hence, we can find a complete solution for the whole path with a cost of about 2^{193} in time and 2^{128} in memory. Furthermore, with a probability of 2^{-128}, the input and output differences in S_0 and S_{48} collide in the feed-forward and BigFinal transformation. Therefore, we can also generate free-start collisions for 6 rounds of the compression function with a time complexity of $2^{193} + 2^{128} \sim 2^{193}$ and 2^{128} memory.

5 Conclusions

In this work, we have presented new results on the second-round candidate of the SHA-3 competition ECHO-256 that improve considerably the previous published cryptanalysis. Our analysis are based on multi-inbound rebound attacks and are summarized in Table 1. The main results are a 5-round collision of the hash function and a 7-round distinguisher of its compression function. All of our results take into account the condition observed in [4], which is needed to merge the results of multiple inbound phases, and satisfy it. The 7-round distinguisher on the compression function uses the stop-in-the-middle algorithms proposed in [10].

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), http://crypto.rd.francetelecom.com/echo/doc/echo_description_1-5.pdf
2. Daemen, J., Rijmen, V.: Understanding Two-Round Differentials in AES. In: Prisco, R.D., Yung, M. (eds.) SCN. LNCS, vol. 4116, pp. 78–94. Springer (2006)
3. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In: Hong, S., Iwata, T. (eds.) FSE. LNCS, vol. 6147, pp. 365–383. Springer (2010)
4. Jean, J., Fouque, P.A.: Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function. In: FSE. pp. 107–128. LNCS (2011)
5. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: ASIACRYPT. LNCS, vol. 5912, pp. 126–143. Springer (2009)
6. 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. LNCS, vol. 3373, pp. 128–141. Springer (2004)
7. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full lane compression function. In: Matsui, M. (ed.) ASIACRYPT. LNCS, vol. 5912, pp. 106–125. Springer (2009)
8. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Fast Software Encryption - FSE 2009. LNCS, vol. 1008. Springer (5665)

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.) Selected Areas in Cryptography. LNCS, vol. 5867, pp. 16–35. Springer (2009)
10. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Advances in Cryptology - CRYPTO 2011, 31th Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. LNCS, Springer (2011), to appear.
11. Naya-Plasencia, M.: How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607 (2010), (extended version). http://eprint.iacr.org/
12. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. LNCS, vol. 6223, pp. 370–392. Springer (2010)
13. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-Full-Active Super-Sbox Analysis Applications to ECHO and Grøstl. In: ASIACRYPT. LNCS (2010), to appear
14. Schläffer, M.: Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. LNCS, vol. 6544, pp. 369–387. Springer (2010)
15. National Institute of Standards, 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
16. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO. LNCS, vol. 2442, pp. 288–303. Springer (2002)

A Chosen-Salt Attacks on the ECHO Compression Function

In this section, we show how to get a collision attack for 6 rounds and a subspace distinguisher for 7 rounds of the ECHO-256 compression function in the chosen-salt model. For both attacks, we get a complexity of 2^{160} compression function evaluations with memory requirements of 2^{128}.

The attacks on the hash functions of ECHO can be extended to the compression function in a straightforward way. In this case, instead of the chaining value, a 512-bit value of another inbound phase is merged with the first inbound phase. In fact, we can continue with a similar 3-round path in backward direction as we have in the hash function case in forward direction. Then, the full active ECHO state is located in the middle round and we can construct attacks for up to 7 rounds for the compression functions of ECHO-256 (see Figure 5).

A.1 The Truncated Differential Path

We use the 7-round truncated differential path given in Figure 5. Black bytes are active and colored bytes show the different inbound and outbound phases. Since this path is sparse, we are able to find many right pairs conforming to the path. We can already compute the expected number of right pairs by considering the MixColumns and SuperMixColumns transformations. At the input, we can freely choose the 256-byte values, the 16-byte difference between the values and the 16-byte salt. We get a reduction of pairs at the first MC and SMC of round 1, the second MC of round 3, the first MC and SMC of round 4, the BMC of round 5 and the second MC of round 6. The differential probability (in base-2 logarithm) for the path is given as follows:

$$8 \times (-12 - 3 - 48 - 48 - 12 - 48 - 12) = -8 \times 183.$$

To summarize, the expected number of pairs conforming to this 7-round truncated differential path is

$$2^{8\times(256+16+16)} \times 2^{-8\times 183} = 2^{800},$$

which corresponds to 800 degrees of freedom. Note that this is much more than for the paths given in [9] and [12].

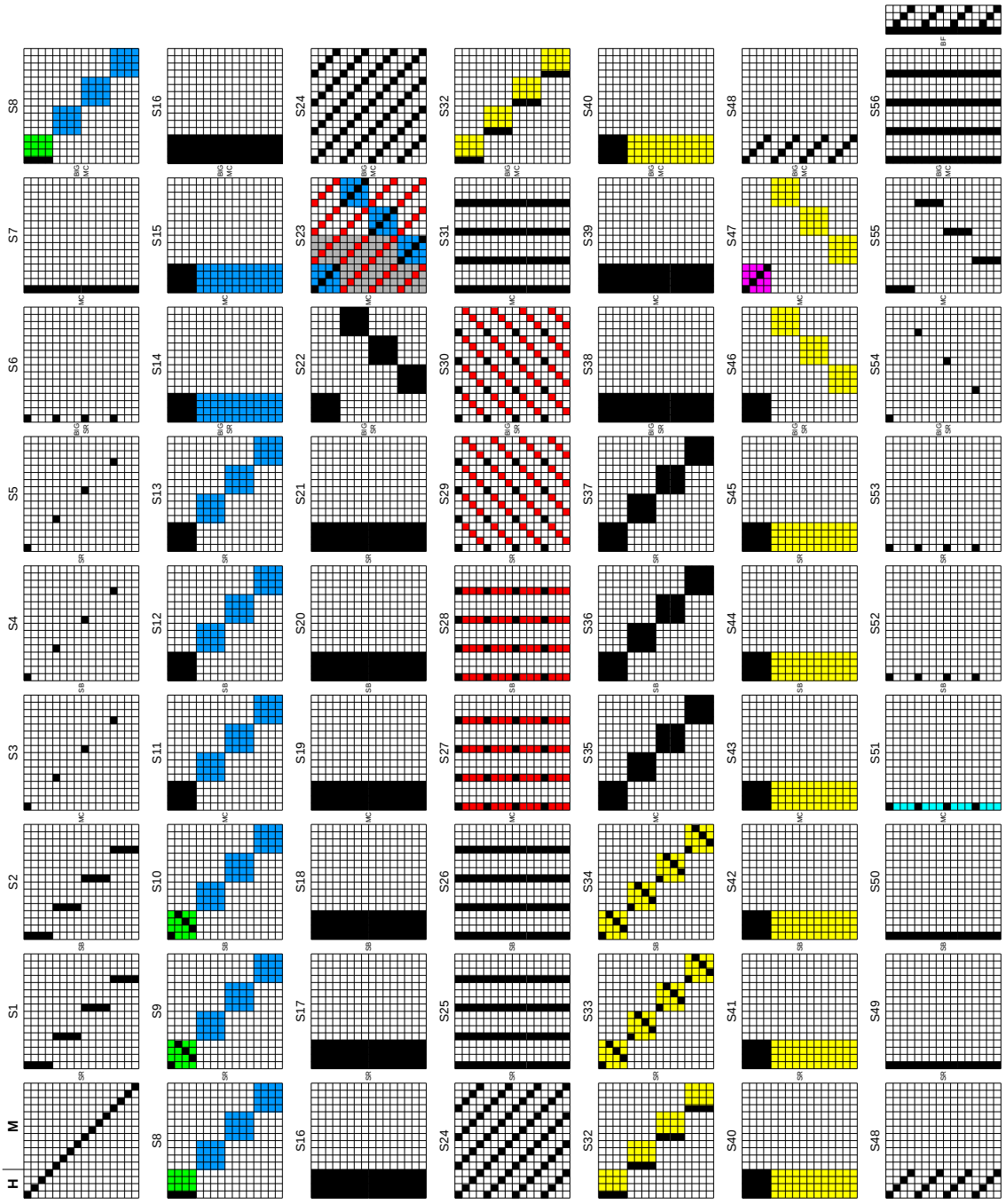

Figure 5: The truncated differential path to get collisions for 6 rounds and near-collisions for 7 rounds of the ECHO-256 compression function. Black bytes are active, red bytes are values computed in the first inbound phase, yellow bytes in the second, blue bytes in the third and green bytes in the fourth inbound or second outbound phase, and cyan bytes in the third outbound phase. Purple bytes are determined in the first outbound phase and gray bytes are chosen in the merge inbound phase.

A.2 Outline of the Attack

The main idea of the attack is to find solutions for the forward and backward parts independently for fixed differences at the same layer between states S_{30} and S_{32}. For the yellow/purple part, we can find 2^{128} pairs with a complexity of 2^{128} by choosing the salt value. For the green/blue/red part, we detail how to find 2^{128} pairs as well, but with a

complexity of 2^{160} by constructing the salt. Then, we just need to match the 128-bit salt values of the forward and backward parts and fulfill the 128-bit condition on the input (red) and output (yellow) values of SuperMixColumnsfor the merge to be possible. Since we get 2^{128} independent pairs for both the forward and backward parts, we can fulfill the resulting 256-bit condition by merging the two resulting lists.

A.3 Finding Right Pairs for Sparse Paths of the Permutation

In this section, we show how to find a pair for the first 6 rounds of the 7-round truncated differential path of Figure 5. We detail how to find such a right pair in time 2^{160} with 2^{128} memory. We use this path in the chosen-salt model to get a collision for 6 rounds and a distinguisher for 7 rounds of the ECHO-256 compression function.

Inbound between S_{30} and S_{40}. We choose a difference for state S_{32} such that the truncated differential path of SuperMixColumns between state S_{30} and S_{32} is fulfilled. Then, for each of the 2^{128} differences in state S_{40}, we perform an inbound phase between states S_{32} and S_{40}. In average, we get one solution with average complexity one: we can then compute 2^{128} pairs for the yellow inbound phase with complexity 2^{128}. We store these pairs sorted by their difference in state S_{40} in list L_1.

Outbound between S_{40} and S_{47}. We continue to find pairs which also satisfy the truncated differential path until state S_{47}. We choose 2^{128} random pairs for the AES state in S_{47} (according to the given truncated differential path) and compute backwards to state S_{40}. For each resulting difference in S_{40}, we lookup the matching difference in list L_1. To match also the values, we can construct the 128-bit salt value accordingly. Thus, we get 2^{128} pairs with complexity 2^{128} according to the truncated differential path from state S_{32} to state S_{48}.

Inbound between S_{23} and S_{30}. The red inbound phase is the same as in the hash function attack: we start with the difference between states S_{30} and S_{32}, which has been chosen in the yellow inbound phase. Then, we do four independent inbound phases for each BigColumn in state S_{23}. Since we can start with at least 2^{32} differences for each column in state S_{23}, we also get 2^{32} pairs for each column with a time complexity of 2^{32}.

Inbound between S_{15} and S_{23}. Independently from the previous step, in the blue inbound phase, we start with a fixed difference in state S_{15} and compute this difference forward to state S_{17}. Again, we can choose all 2^{32} differences for each BigColumn of state S_{23} and perform the blue inbound phases independently for each active AES state in the backward direction. For each column, we get 2^{32} pairs with a complexity of 2^{32}.

Merge Inbounds. When merging the solutions of the blue and red inbound phases, we want to get one pair with average complexity one. Note that for each inbound phase and each column of state S_{23}, we have 2^{32} right pairs. Moreover, we are allowed to set the salt value. We then start by matching the differences in the overlapping four bytes of each BigColumn. Since we have 2^{32} solutions for each of the blue and the red part, we get $2^{32} \times 2^{32} \times 2^{-32} = 2^{32}$ pairs with matching differences but non-matching values.

To match also these 4-byte values, we only set the four diagonal bytes of the salt value. For each of the 2^{32} pairs with matching differences, we compute the diagonal bytes of the

salt such that the values also match. We sort the resulting list according to the 4-byte salt value and repeat the same for all four BigColumns of state S_{23}. Then, we just need to iterate through all four lists and search for matching salt values. Note that for some salt values, we will get no solution, but for some we will get more than one solution. On average, we expect to get 2^{32} matching pairs with a complexity of 2^{32} with chosen diagonal bytes of the salt.

Inbound between S_6 and S_{15}. To find a pair of states conforming to the green part, we first choose a difference verifying the truncated differential path between state S_6 and state S_8. The second starting point for the green inbound phase is the difference in state S_{15}, which has been chosen in the blue inbound phase. Again, we get one pair on average for each starting differential. This pair needs to be connected with the solutions of the blue inbound phase. To do so, we first match the values in the diagonal bytes of state S_{15}. Remember that in the previous phases, we have constructed 2^{32} pairs for a single difference in state S_{15}. Among these pairs, we expect to find one such that the diagonal 4-byte values between the green and blue inbound phase match. To connect the other 12 bytes, we can simply set the remaining 12 bytes of the salt value. Hence, we get one solution for the combined green, blue and red part with an average complexity of 2^{32}.

First Part in Merging Inbound Phases. To merge the inbound phases, we first compute 2^{128} pairs for the yellow/purple part with a time and memory complexity of 2^{128} and store these pairs in a list L_2. Similarly, we compute 2^{128} pairs for the green/blue/red part. Since the complexity to compute one solution for this part is 2^{32}, the time complexity to compute all 2^{128} pairs is 2^{160}. To connect the resulting pairs between states S_{30} and S_{32}, we need to satisfy two 128-bit conditions. First, we need to verify the linear 128-bit SuperMixColumns condition observed by Jean and Fouque in [4]. Second, since each solution of the yellow/purple and green/blue/red part has also a different salt value, we need to match the 128-bit salt as well. In the end, this leads to a 256-bit condition, which we can be satisfied by merging the two lists L_1 and L_2 under that condition to produce $2^{128} \times 2^{128} \times 2^{-256} = 1$ right pair, which satisfies the whole 6-round truncated differential path. The time complexity of this step is 2^{160} with memory requirements of 2^{128}.

Second Part in Merging Inbound Phases. In the second part of the merge inbound phase, we need to find values for the two first columns of Figure 3. This part of the attack is the same as in the hash function attack on ECHO-256 (see Section 3.4).

Third Part in Merging Inbound Phases. The only difference in the third part of the merge inbound phase is that we change the time-memory trade-off slightly to get an average complexity of 1 for each solution. Again, we do a generalized birthday attack but this time, we start with 2^{96} independent values for each column of state S_{30} (Figure 3). Since we have a 192-bit condition in state S_{23}, we get $2^{3\times 96} \times 2^{-192} = 2^{96}$ solutions with a complexity of 2^{96} in time and memory, or with an average complexity of 1 per solution [16]. It follows that we can find up to 2^{160} right pairs for the 6-round truncated differential path with a total complexity of 2^{160} and memory requirements of 2^{128}.

A.4 Chosen-Salt Collision Attack for 6 Rounds

To get a collision for 6 rounds of the 512-bit compression function of ECHO-256 in the chosen-salt, we need to ensure that the differences in the feed-forward cancel the output

differences of the permutation: this happens with a probability of 2^{-128}. Since we can find 2^{160} pairs for the truncated differential path with a complexity of 2^{160}, we expect to find 2^{32} collisions at the output of the 6-round compression function with a time complexity of 2^{160} and memory requirements of 2^{128}.

A.5 Chosen-Salt Distinguisher for 7 Rounds

To get a chosen-salt distinguisher for 7 rounds of the compression function of ECHO-256, we use the whole truncated differential path given in Figure 5. Note that the last round of this truncated differential path is verified with a probability of 2^{-96}. Furthermore, with an additional 32-bit condition on the active bytes in state S_{52}, we can fix the difference at the output of the permutation, prior to the feed-forward. In this case, only the 16-byte differences in the diagonal bytes of the output of the compression function change for each additional found pair. In other words, the difference vector space at the output of the compression function reduces to a dimension of 128. We use a third outbound phase to satisfy these conditions in the last round. Since we can find one solution for the white bytes of the 6-round path with an average complexity one, we can find one pair which also satisfies the conditions in the last round with a time and memory complexity 2^{128}. Note that we can find up to 2^{32} such pairs with a total complexity of 2^{160} in time and 2^{128} memory.

Again, we use [5, Equation 19] to compute the complexity of a generic distinguishing attack on the ECHO-256 compression function. We get the parameters $N = 512$ (compression function output size), $n = 128$ (dimension of output difference vector space) and $t = 2^{32}$ (number of outputs in vector space) for the subspace distinguisher. Then, the generic complexity to construct 2^{32} elements in a vector space of dimension 128 is about $2^{207.8}$ compression function evaluations. All in all, we get a chosen-salt distinguisher for 7 rounds of the ECHO-256 compression function with a complexity of 2^{160} in time and 2^{128} memory.

Note that we can use almost the same attack to construct 2^{32} near-collisions with a zero difference in the same 320 bits. Again, we need to satisfy the 96-bit condition in the cyan bytes in the last round. However, we now require that the overlapping 4-byte differences in the feed-forward cancel each other. This 32-bit condition ensures that we get only $4 \times 6 = 24$ active bytes at the output of the compression function for 2^{32} pairs with a total complexity of 2^{160} in time and 2^{128} memory in the chosen-salt model.

B Collision Attack on the 4-round ECHO-256 Hash Function

In this section, we describe a way to extend the compression function collision attack presented in [4] into a collision attack on the 4-round hash function ECHO-256 with a time and memory complexity of 2^{64}. That published attack finds a message pair (M_1, M_1') and a chaining value h such that $f(h, M_1) = f(h, M_1')$, where f is the ECHO-256 compression function. To get a collision in the hash function, the difference then consists in finding a message block M_0 which verifies $f(IV, M_0) = h$. This way, the collision in the hash function is the result of an internal collision in the compression function: consequently, we do not need to take care of the padding in that scenario. For any message M, we would have: $\text{ECHO}_{4R}(IV, M_0||M_1||M) = \text{ECHO}_{4R}(IV, M_0||M_1'||M)$, where $||$ denotes the concatenation of messages and ECHO_{4R} the 4-round ECHO-256 hash function.

Description of the attack. As in the other attacks suggested in this paper, we use the rebound technique with multiple inbound phases to find a valid message pair. The path

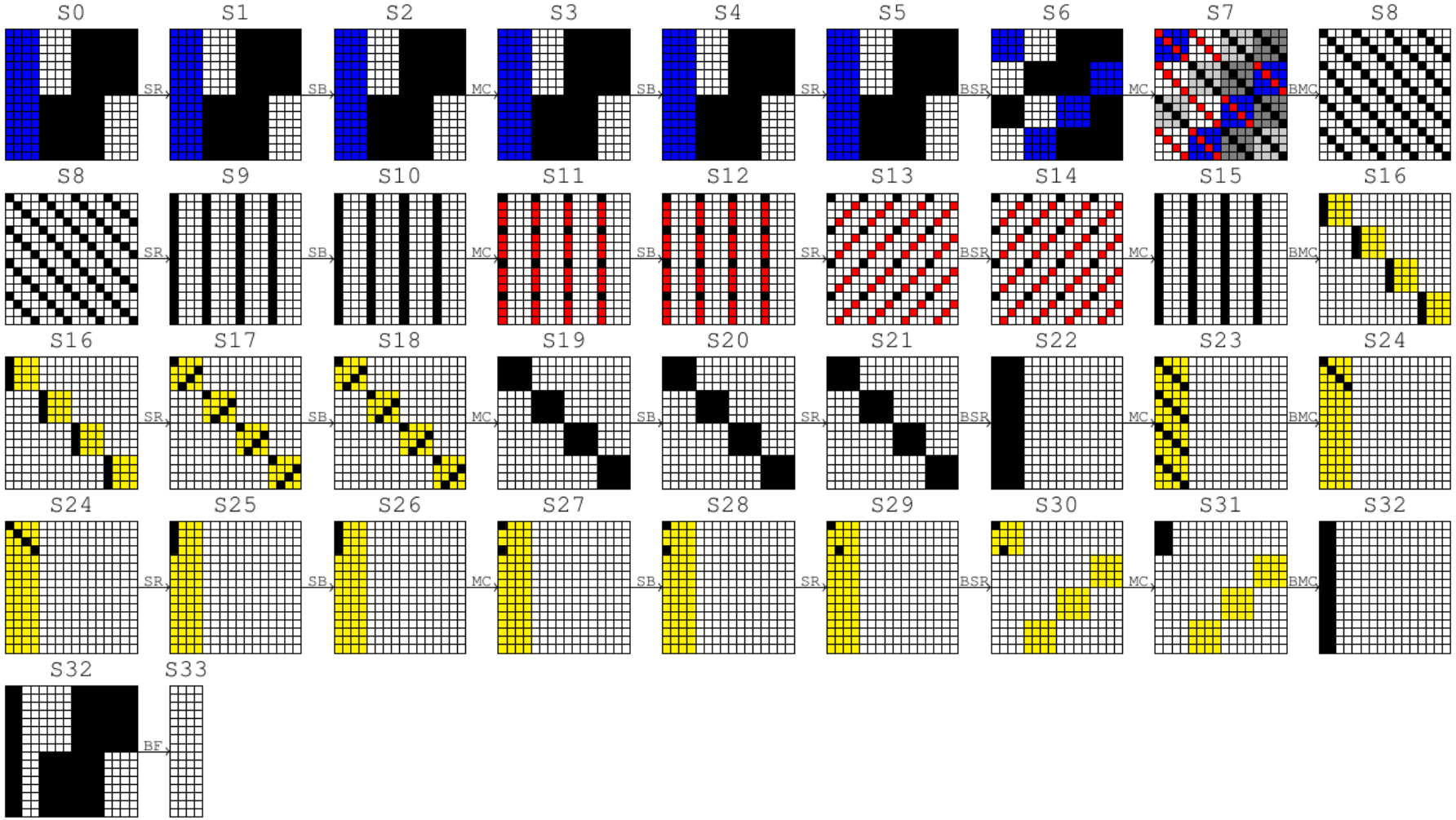

Figure 6: The truncated differential path used in the 4-round collision attack on the ECHO-256 hash function.

used here is depicted in Figure 6. The first inbound is located in the second round between states S_7 and S_{14} and can be done in parallel on the four BigColumns and the second one is in the third round between states S_{16} and S_{24}. We extend the latter by a probabilistic outbound phase to filter out some of its valid pairs in order to reduce the number of active bytes in the final fourth round. In the sequel, we consider the alternative description of the permutation (see 2.1).

At first, we precompute and store the Differential Distribution Table Δ of the SuperSBox in time and memory 2^{64}. Then, we randomize the differences around the merging point: the SuperMixColumns at the end of the second round. Using Δ, we find a pair of internal states conforming to the second inbound and the outbound phase. This can be done in the same way as in [4] in less than 2^{64}. We then find a valid pair of BigColumns for the first BigColumn in the first inbound: this fixes the value of $\mathrm{diag}(S_7[0,0])$ overlapping with an AES state directly coming from the previous chaining value yet to be determined.

By generating 2^{64} message M_0, we obtain a list L of 2^{32} chaining values sharing the same value on $\mathrm{diag}(S_7[0,0])$. We now generate all the 2^{32} possible BigColumns pairs for the second BigColumn in the first inbound. For each pair, we compute the value of $\mathrm{diag}(S_7[3,1])$: we expect one element of L to share this value because L contains 2^{32} elements. Consequently, we can update L by completing each of its entry by the associated pair for the second BigColumn and get 2^{32} pairs for L again. For the third BigColumn, we compute 2^{64} pairs conforming to the path among the 2^{96} possible ones. With the same argument, for any element of L, we expect to find 2^{32} pairs where the value of $\mathrm{diag}(S_7[2,2])$ matches the one dictated by that particular chaining value. We link each of the 2^{32} elements of L with a set E of 2^{32} valid pairs for the third BigColumn. In other words, we get 2^{64} pairs where the three first BigColumns match the respective chaining value.

At this point, each of the 2^{32} entries of L consists of a chaining value $h = f(IV, M_0)$, a pair of BigColumns for the two first BigColumns and a set E of 2^{32} valid pairs for the third BigColumn, all conforming to the first inbound. As demonstrated in [4], the 128-bit condition to merge the two inbound phases is deported to the fourth BigColumn in the first inbound. More precisely, once we know a pair for each of the three first BigColumns, we can deduce the fourth one so that the merge around the SuperMixColumns layer is possible. For each entry in L and for each associated set E, we are in that case so that we can deduce the pair for the fourth BigColumn. Finally, we check if the truncated path is verified for that BigColumn and if the value in $\mathrm{diag}(S_7[3,1])$ matches the expected one. The two events occur with probability 2^{-64} but since we can repeat the check $2^{32} \times 2^{32} = 2^{64}$ times, we should find one M_0 and one pair for each of the four BigColumns.

Finally, we can finish the attack as in [4] by ensuring that the differences cancel out in the feed-forward and by merging the two partial solutions. We note that the merging is slightly different since we need to take care of the known AES states in S_7 but is still feasible in time 2^{64}.