# Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function

Martin Schläffer

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria

martin.schlaeffer@iaik.tugraz.at

**Abstract.** In this work we present the first results for the ECHO hash function. We provide a subspace distinguisher for 5/8 rounds, near-collisions on 4.5/8 rounds and collisions for 4/8 rounds of the ECHO-256 hash function. The complexities are $2^{96}$ compression function calls for the distinguisher and near-collision attack, and $2^{64}$ for the collision attack. The memory requirements are $2^{64}$ for all attacks. Furthermore, we provide improved compression function attacks on ECHO-256 to get a distinguisher on 7/8 rounds and near-collisions for 6.5/8 rounds with chosen salt. The compression function attacks also apply to ECHO-512. To get these results, we consider new and sparse truncated differential paths through ECHO. We are able to construct these paths by analyzing the combined MixColumns and BigMixColumns transformation. Since in these sparse truncated differential paths at most 1/4 of all bytes of each ECHO state are active, missing degrees of freedom are not a problem. Therefore, we are able to mount a rebound attack with multiple inbound phases to efficiently find according message pairs for ECHO.
**Keywords:** hash functions, SHA-3 competition, ECHO, cryptanalysis, truncated differential paths, rebound attack, subspace distinguisher, near-collisions, collision attack

## 1 Introduction

Many new and interesting hash function designs have been proposed in the NIST SHA-3 competition [15]. In this paper, we analyze the hash function ECHO [1], which is one of 14 Round 2 candidates of the competition. ECHO is a wide-pipe, AES based design which transforms 128-bit words similar as AES transforms bytes. Inside these 128-bit words, two AES rounds are used to transform bytes. So far, most cryptanalytic results of ECHO were limited to the internal permutation [7, 11]. Recently, reduced round attacks on the wide-pipe compression function of ECHO have been published [16], which cover up to 4/8 rounds for ECHO-256 and 6/10 rounds of ECHO-512. However, a drawback of attacks on building blocks (such as compression functions or permutations) is that they cannot be used to compare SHA-3 candidates due to their great design variety and different requirements for building blocks.

Therefore, in this work we analyze the ECHO hash function and present results for up to 5/8 rounds of ECHO-256. We use the subspace distinguisher [8, 9] to compare our distinguishing attacks with the generic complexity on ideal hash functions. Our results greatly improve upon the previous results on the compression function, which are attacks on a similar number of rounds. Furthermore, we present attacks on the compression function with chosen salt for up to 7/8 rounds of ECHO-256 which also extend to ECHO-512. The main improvement is to consider a new type of sparse truncated differential paths by placing only a single active byte in the ECHO state with 16 active AES states. In all previous paths, the full active ECHO states also had full active AES states. The construction of such paths is possible by combining the last MixColumns transformation of the second AES round with the BigMixColumns transformation of an ECHO round to a SuperMixColumns transformation.

The attack itself is a rebound attack [12] with multiple inbound phases. Similar attacks have been applied to the SHA-3 candidate LANE [10] and the hash function Whirlpool [8]. Since the truncated differential paths are very sparse, we have plenty degrees of freedom to merge

the solutions of these multiple inbound phases. Note that using multiple inbound phases, we can control more distant parts of much longer truncated differential paths than in a start-from-the-middle attack [11] or Super-Sbox analysis [7, 8, 13] where the controlled rounds are limited to only the middle rounds. To merge independent solutions of multiple inbound phases, we use a technique based on the generalized birthday attack [17]. A summary of our main results on the ECHO-256 hash function and both ECHO compression functions are given in Table 1. More details and the respective generic complexities, especially for near-collisions and distinguishers, are given in the sections describing the attacks.

Table 1: Summary of our results and related analysis for ECHO.

| Hash Function | Target | Rounds | Time | Memory | Type | Reference |
|---|---|---|---|---|---|---|
| ECHO-256 | hash function | 5/8 | $2^{96}$ | $2^{64}$ | distinguisher | Sect. 4.1 |
| | | 4.5/8 | $2^{96}$ | $2^{64}$ | near-collision | Sect. 4.2 |
| | | 4/8 | $2^{64}$ | $2^{64}$ | collision | Sect. 4.3 |
| | compression function | 7/8 | $2^{107}$ | $2^{64}$ | distinguisher* | Sect. 5.3 |
| | | 6.5/8 | $2^{96}$ | $2^{64}$ | free-start near-collision* | Sect. 5.4 |
| | | 4/8 | $2^{64}$ | $2^{64}$ | distinguisher | [16] |
| | | 3/8 | $2^{64}$ | $2^{64}$ | semi-free-start collision | [16] |
| ECHO-512 | compression function | 7/10 | $2^{106}$ | $2^{64}$ | distinguisher* | Sect. A.1 |
| | | 6.5/10 | $2^{96}$ | $2^{64}$ | free-start near-collision* | Sect. A.2 |
| | | 6/10 | $2^{96}$ | $2^{64}$ | distinguisher | [16] |
| | | 3/10 | $2^{96}$ | $2^{64}$ | semi-free-start collision | [16] |

* with chosen salt (without chosen salt the complexity increases by a factor of $2^{128}$)

## 2 Description of ECHO

In this section we briefly describe the AES based SHA-3 candidate ECHO. For a detailed description of ECHO we refer to the specification [1]. Since ECHO heavily uses AES round transformations, we describe the AES block cipher first.

### 2.1 The AES Block Cipher

The block cipher Rijndael was designed by Daemen and Rijmen and standardized by NIST in 2000 as the Advanced Encryption Standard (AES) [14]. The AES follows the wide-trail design strategy [4, 5] and consists of a key schedule and state update transformation. Since ECHO does not use the AES key schedule, we just describe the state update here.

In the AES, a $4 \times 4$ state of 16 bytes is updated using the following 4 round transformations, with 10 rounds for AES-128, 12 rounds for AES-192 and 14 rounds for AES-256:

- The non-linear layer SubBytes (SB) applies the AES S-Box to each byte of the state independently.
- The cyclical permutation ShiftRows (SR) rotates the bytes of row $j$ to the left by $j$ positions.
- The linear diffusion layer MixColumns (MC) multiplies each column of the state by a constant MDS matrix $M_{\mathsf{MC}}$.
- AddRoundKey (AK) adds the 128-bit round key $K_i$ to the AES state.

Note that a round key is added prior to the first round and the MixColumns transformation is omitted in the last round of AES. For a detailed description of the AES we refer to [14].

### 2.2 The ECHO Hash Function

The ECHO hash function is a SHA-3 candidate submitted by Benadjila et al. [1]. It is a double-pipe iterated hash function and uses the HAIFA [2] domain extension algorithm. More precisely, a padded $t$-block message $M$ and a salt $s$ are hashed using the compression function $f(H_{i-1}, M_i, c_i, s)$, where $c_i$ is a bit counter, IV the initial value and $trunc(H_t)$ a truncation to the final output hash size of $n$ bits:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, M_i, c_i, s) \quad \text{for } 1 \leq i \leq t \\ h &= trunc_n(H_t). \end{aligned}$$

The message block size is 1536 bits for ECHO-256 and 1024 bits for ECHO-512, and the message is padded by adding a single 1 followed by zeros to fill up the block size. Note that the last 18 bytes of the last message block always contain the 2-byte hash output size and the 16-byte message length.

The compression function of ECHO uses one internal 2048-bit permutation $P$ which manipulates 128-bit words similar as the AES manipulates bytes. The permutation consists of 8 rounds in the case of ECHO-256 and has 10 rounds for ECHO-512. The internal state of the permutation $P$ can be modeled as a $4 \times 4$ matrix of 128-bit words. We denote one ECHO state by $S_i$ and each 128-bit word or AES state is indexed by $[r, c]$, with rows $r \in \{0, ..., 3\}$ and columns $c \in \{0, ..., 3\}$ of the ECHO state.

The 2048-bit input of the permutation (which is tweaked by the input counter $c_i$ and the salt $s$) are the previous chaining variable $H_{i-1}$ and the current message block $M_i$, which are concatenated to each other. After the last round of the permutation, a feed-forward (FF) is applied to get the preliminary output $V$:

$$V = P_{c_i,s}(H_{i-1}||M_i) \oplus (H_{i-1}||M_i). \tag{1}$$

To get the 512-bit chaining variable $H_i$ for ECHO-256, all columns of the ECHO output state $V$ are XORed. In the case of ECHO-512, the 1024-bit chaining variable $H_i$ is the XOR of the two left and the two right columns of $V$. The feed-forward together with the compression of columns is called the BigFinal (BF) operation. To get the final output of the hash function, the lower half is truncated in the case of ECHO-256 and the right half is truncated for ECHO-512.

The round transformations of the ECHO permutation are very similar to AES rounds, except that 128-bit words are used instead of bytes. One round is the composition of the following three transformations in the given order:

- The non-linear layer BigSubWords (BSW) applies two AES rounds to each of the 16 128-bit words of the internal state. The first round key consists of a counter value initialized by $c_i$ and increased for every AES state and round of ECHO. The second round key consists of the 128-bit salt $s$.
- The cyclical permutation BigShiftRows (BSR) rotates the 128-bit words of row $j$ to the left by $j$ words.
- The linear diffusion layer BigMixColumns (BMC) mixes the AES states of each ECHO column by the same MDS matrix $M_{\mathsf{MC}}$ but applied to those bytes with equal position inside the AES states.

## 3 Improved Truncated Differential Analysis of ECHO

In this section we describe the main concepts used to attack the ECHO hash function. We first describe the improved truncated differential paths which have a very low number of active S-boxes. These sparse truncated differential paths are the core of our attacks and for a better

description of the attacks, we reorder the ECHO round transformations. This reordering gives two combined building blocks of ECHO, the SuperMixColumns and SuperBox transformations. We then show how to efficiently find both differences and values through these functions for a given truncated differential path.

### 3.1 Sparse Truncated Differential Paths for ECHO

In this section we construct truncated differential paths with a low number of active bytes. Since ECHO has the same properties for words as AES has for bytes, at least 25 AES states are active in each 4-round differential path. However, we can reduce the number of active S-boxes in each AES state to get a sparse 4-round truncated differential path with only 245 active S-boxes. Note that the truncated differential paths of the previously best analysis of ECHO have already 320 active S-boxes in a single round [16] and a trivial lower bound [1] of active S-boxes for 4 rounds is 125.

The AES structure of ECHO ensures that the minimum number of active words for 4 rounds has the following sequence of active words:

$$1 \rightarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1$$

Also, the same sequence of active bytes holds for 4 rounds of AES. In previous analysis of ECHO, truncated differential paths have been used with 16 active bytes in the AES states where the ECHO state has 16 active words. Hence, in these attacks always one full active state with 256 active S-boxes was used. In the following, we show how to construct sparse truncated differential paths with only 64 active S-boxes by looking inside the AES rounds.

The main idea is to place the AES states with only one active byte in the ECHO rounds with 16 active words. This way, the number of total active bytes (or S-boxes) can be greatly reduced. Since one round of ECHO consists of two AES rounds, we have to place the full active AES states in those rounds with 4 active words. Finally, the ECHO state with only one active word contains only one active byte in the active AES state as well. The resulting 4-round truncated differential path of ECHO is given in Fig. 1 and consists of only 245 active S-boxes. Note that in the attacks on ECHO, we use this truncated differential path with small modifications to improve the overall complexity of the attacks.

### 3.2 An Equivalent ECHO Round Description

For an easier description of our attack, we use an equivalent description of one ECHO round. 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. This way, we get two new super-round transformations separated just by byte shuffling operations: SuperMixColumns and SuperBox. These functions with adjacent byte shuffling operations are shown in Fig. 2. In the following subsections we show how to efficiently find differences and pairs for both transformations according to a given truncated differential path.

### 3.3 SuperMixColumns

The SuperMixColumns transformation combines the four MixColumns transformations of the second AES round with the 4 MixColumns transformations of BigMixColumns in the same $1 \times 16$ column slice of the ECHO state (see Fig. 2). We denote by column slice the 16 bytes of the same

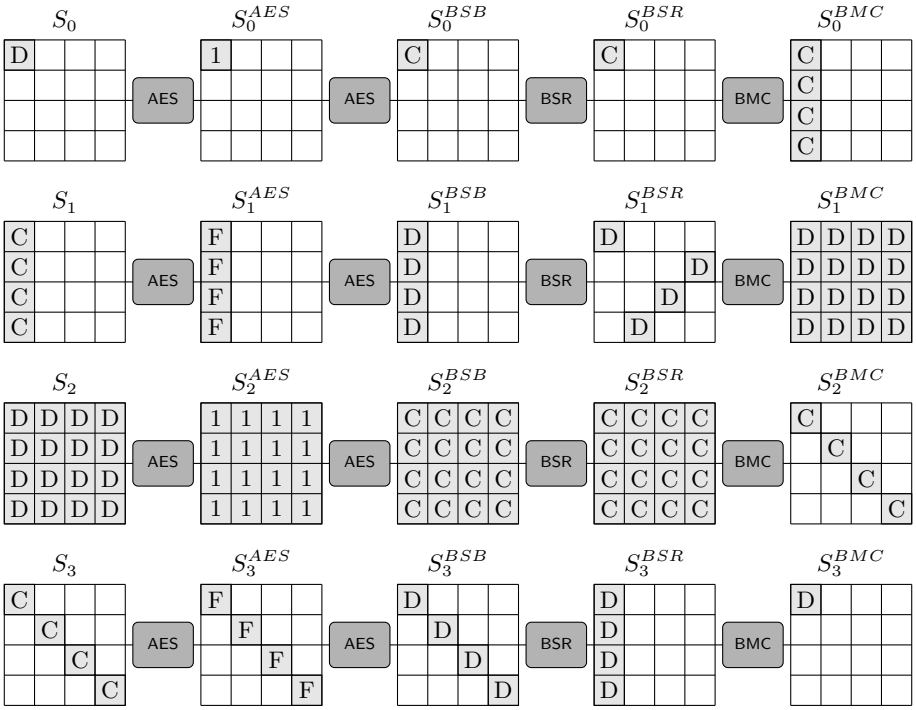


Fig. 1: The sparse truncated differential path for 4 rounds of ECHO. By 1, D, C, F we denote the pattern and number of active bytes in each AES state (also see [7]). A 1 denotes an AES state with only one active byte, a D an active diagonal (4 active bytes), a C an active column (4 active bytes) and an F denotes a full active state (16 active bytes). Note that the path has only one active byte in each AES state of ECHO state $S_2^{AES}$.

1-byte wide column of the $16 \times 16$ ECHO state. Note that the BigMixColumns transformation consists of $16 \times 4$ parallel MixColumns transformations. Each of these MixColumns transformations mixes those four bytes of an ECHO state column, which have the same position in the four AES states. Using the alternative description of ECHO (see Fig. 2), it is easy to see that four MixColumns operations of AES work on the same column slice as four MixColumns operations of BigMixColumns. We combine these eight MixColumns transformations to get a SuperMixColumns transformation on a 1-byte wide column slice of ECHO.

We have determined the $16 \times 16$ matrix $M_{\mathsf{SMC}}$ of the SuperMixColumns transformation which is applied to the ECHO state instead of MixColumns and BigMixColumns. This matrix can be computed by the Kronecker product of two MixColumns MDS matrices $M_{\mathsf{MC}}$ and is given as follows:

$$M_{\mathsf{SMC}} = M_{\mathsf{MC}} \otimes M_{\mathsf{MC}} = \begin{bmatrix} 2&3&1&1 \\ 1&2&3&1 \\ 1&1&2&3 \\ 3&1&1&2 \end{bmatrix} \otimes \begin{bmatrix} 2&3&1&1 \\ 1&2&3&1 \\ 1&1&2&3 \\ 3&1&1&2 \end{bmatrix} = \begin{bmatrix} 4&6&2&2&6&5&3&3&2&3&1&1&2&3&1&1 \\ 2&4&6&2&3&6&5&3&1&2&3&1&1&2&3&1 \\ 2&2&4&6&3&3&6&5&1&1&2&3&1&1&2&3 \\ 6&2&2&4&5&3&3&6&3&1&1&2&3&1&1&2 \\ 2&3&1&1&4&6&2&2&6&5&3&3&2&3&1&1 \\ 1&2&3&1&2&4&6&2&3&6&5&3&1&2&3&1 \\ 1&1&2&3&2&2&4&6&3&3&6&5&1&1&2&3 \\ 3&1&1&2&6&2&2&4&5&3&3&6&3&1&1&2 \\ 2&3&1&1&2&3&1&1&4&6&2&2&6&5&3&3 \\ 1&2&3&1&1&2&3&1&2&4&6&2&3&6&5&3 \\ 1&1&2&3&1&1&2&3&2&2&4&6&3&3&6&5 \\ 3&1&1&2&3&1&1&2&6&2&2&4&5&3&3&6 \\ 6&5&3&3&2&3&1&1&2&3&1&1&4&6&2&2 \\ 3&6&5&3&1&2&3&1&1&2&3&1&2&4&6&2 \\ 3&3&6&5&1&1&2&3&1&1&2&3&2&2&4&6 \\ 5&3&3&6&3&1&1&2&3&1&1&2&6&2&2&4 \end{bmatrix}$$

Note that the optimal branch number of a $16 \times 16$ matrix is 17, which could be achieved by an MDS matrix. Using Magma we have computed the branch number of SuperMixColumns which is 8. Hence, it is possible to find differential paths in SuperMixColumns such that the sum of active bytes at input and output is only 8. An according truncated differential path through MixColumns and BigMixColumns has the following sequence of active bytes:

$$4 \xrightarrow{\mathsf{MC}} 16 \xrightarrow{\mathsf{BMC}} 4$$

An example for a valid SuperMixColumns differential according to this truncated differential path is given as follows:

$$\mathsf{SMC}([\mathsf{E000\ 9000\ D000\ B000}]^{\mathsf{T}}) = [\mathsf{2113\ 0000\ 0000\ 0000}]^{\mathsf{T}}$$

However, the probability for a truncated differential path from $4 \rightarrow 4$ active bytes (with fixed position) through SuperMixColumns is $2^{-24}$. Hence, only $2^8$ (out of $2^{32}$) such differentials for the given position of active bytes exist. To improve the complexity of our attacks, we simply choose the differences accordingly. In the sparse truncated differential path of Fig. 1, this $4 \rightarrow 4$ transition through SuperMixColumns occurs in the second and forth round.

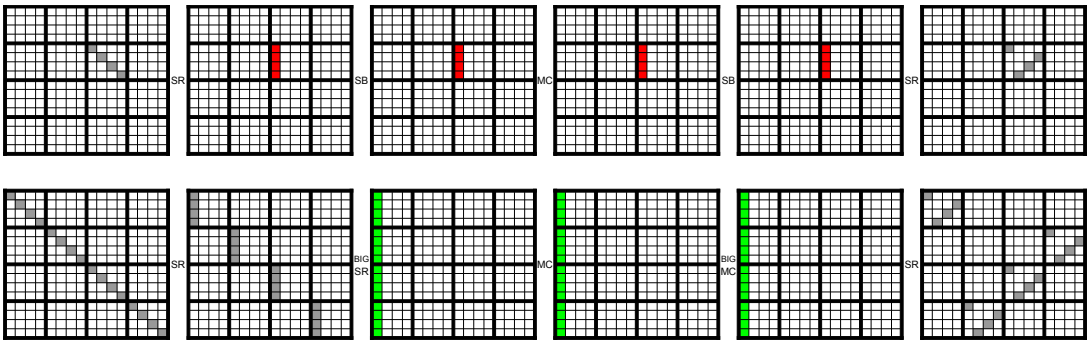


Fig. 2: The two super-round transformations of ECHO: SuperBox (top, red) and SuperMixColumns (bottom, green) with adjacent byte shuffling operations (ShiftRows and BigShiftRows).

### 3.4 SuperBox

The SuperBox has first been used by the designers of AES in the differential analysis of two AES rounds [6]. Since one round of ECHO also consists of two consecutive AES rounds we use this concept in our analysis as well. Using SuperBoxes, we can represent two rounds of AES using a single non-linear layer and two adjacent linear layers. Since we can swap the SubBytes and ShiftRows operation of the first AES round, we get a sequence of SB-MC-SB transformations with independent columns in the middle. One such column is called a SuperBox and consists of 4 parallel S-boxes, one MixColumns operation and another 4 parallel S-boxes (see Fig. 2). Hence, a SuperBox is in fact a 32-bit non-linear S-box.

This separation of two AES rounds into parallel 32-bit SuperBoxes allows to efficiently find pairs for a given (truncated) differential. In a theoretical attack on ECHO or if we do not care about memory, we can simply pre-compute and store the whole differential distribution table (DDT) of the AES SuperBox with a time and memory complexity of $2^{64}$. The DDT stores which input/output differentials of the SuperBox are possible and also stores all input values for which the differentials are fulfilled. Note that in ECHO, each SuperBox is keyed in the middle by

the counter value. Hence, we also need different DDTs for all SuperBoxes with different keys. To reduce the memory requirements and the maximum time to find values for given SuperBox differentials, a time-memory trade-off with average complexity one and memory requirements of $2^{32}$ can be used. This method has first been proposed in the analysis of the hash function Whirlpool [8, Appendix A] and applied to Grøstl in [13]. The same technique has independently been discovered in [7].

### 3.5 Expected Number of Pairs

At this point, we can already compute the expected number of pairs conforming to the 4-round truncated differential path given in Fig. 1. The resulting number of solutions determine the degrees of freedom we have in the attack. At the input of the path, we have a 2048-bit value and differences in 4 bytes. Therefore, the total number of possible inputs pairs (excluding the 128-bit salt) is about

$$2^{2048} \cdot 2^{8 \cdot 4} = 2^{8 \cdot 260} = 2^{2080}.$$

In general, the probability for a random pair to follow a truncated differential path from $a$ to $b$ active bytes (with $a + b \geq 5$) through MixColumns is $2^{-8 \cdot (4-b)}$. An exception is the propagation from $4 \rightarrow 16 \rightarrow 4$ bytes through SuperMixColumns, which has a probability of $2^{-24}$ (see Sect. 3.3). Multiplying all probabilities through MixColumns and SuperMixColumns gives the approximate probability for a random input pair to follow the whole truncate differential path. For the path given in Fig. 1, we get a probability less than one for all MixColumns or SuperMixColumns transformation where a reduction in the number of active bytes occur. This happens in the 1st MC of round 1, the 2nd MC of round 2, the 1st MC and SMC of round 3, and the 2nd MC and BMC of round 4. We then get for the total probability of the truncated differential path (in base 2 logarithm):

$$-8 \cdot (3 + 4 \cdot 12 + 16 \cdot 3 + 4 \cdot 3 + 4 \cdot 12 + 3 \cdot 4) = -8 \cdot 171$$

So in total, the expect number of solutions for the given path is

$$2^{8 \cdot 260} \cdot 2^{-8 \cdot 171} = 2^{8 \cdot 89} = 2^{712}$$

and we still have 712 degrees of freedom in this 4-round truncated differential path.

## 4 Attacks on the ECHO-256 Hash Function

In this section we use the sparse truncated differential path and properties of SuperMixColumns to get attacks for up to 5 rounds of the ECHO-256 hash function. We first describe our main result, the subspace distinguisher for 5 rounds of ECHO-256 in detail. Then, we briefly show how to get near-collisions for 4.5 rounds and collisions for 4 rounds of ECHO-256.

### 4.1 Subspace Distinguisher for 5 Rounds

In this section we show that ECHO-256 reduced to 5 rounds can be distinguished from an ideal hash function. We are able to construct a large set of output differences which fall into a vector space of fixed dimension. But when does this result in a distinguisher on the hash function? An attacker could have chosen the vector space specifically to fit a previously computed set of differences. Also, finding up to $x$ differences in subspace of dimension $x$ is trivial, even for ideal functions. But once a subspace has been chosen, finding additional differences in this subspace should again have the generic complexity. We have a similar situation in preimage attacks:

finding a preimage for an image chosen by the attacker is trivial if the attacker already knew the preimage. Note that in most distinguishing attacks, the generic complexity also depends on the number of found solutions. To compare distinguishers with generic attacks, differential $q$-multicollisions have been used in the distinguishing attacks on AES [3]. More general, to analyze the complexity of finding differences in a vector space of fixed dimension, the subspace distinguisher has been introduced in the analysis of Whirlpool [8, 9]. Before we describe the subspace distinguisher for 5 rounds of ECHO-256 in detail, we give an overview of the truncated differential path and provide a brief outline of the attack.

**The Truncated Differential Path.** For 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 Fig. 3. We use colors (red, yellow, green) to describe different phases of the attack and to denote their resulting solutions. Active bytes are denoted by black color and all AES states are active which 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 as described in Sect. 3.1. Except for the beginning and end, at most 1/4 of the ECHO state is active and therefore, we have enough freedom to find many solutions. Since the lower half of the state is truncated, we have most differences in the lower half of the message and there are no differences in the chaining input (blue). The padding of the second (and last) message block is denoted by cyan bytes. The last 16 bytes (one AES state) of the padding contain the message length, and the two bytes above contain the 2-byte value with the hash size. 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.

**Attack Outline.** To find input pairs according to this path we use the rebound attack [12] with multiple inbound phases [8, 10]. 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. For the attack on 5 rounds of ECHO-256 we use an inbound phase in round 2 (red) and another inbound phase in round 3 (yellow). The 1st inbound phase finds values and differences for the red bytes which we merge (connect) with the chaining input (blue) and padding (cyan). Then, we compute the solutions of the 2nd inbound phase forwards in the outbound phase (green) to insure the propagation according to the truncated differential path until the end. Finally, we merge the solutions of the two inbound phases by determining the remaining (white) values using a generalized birthday attack on 4 independent columns of the state. Note that in some cases the probability to find one solution is only close to one. However, for simplicity reasons of describing the attack we assume it is one, since we have enough freedom in the attack to repeat all phases with different starting points to get one solution on average.

**1st Inbound.** We start the 1st inbound phase with a random difference according to the truncated differential path through SuperMixColumns between state $S_{14}$ and state $S_{16}$ (see Sect. 3.3). For these differences, we compute backward to get the output differences of the SuperBoxes in state $S_{12}$. For each column in state $S_7$ we choose $2^{32}$ random differences for the given active bytes. We compute these differences forward through BigMixColumns to the input of the SuperBoxes. Note that for the last column we could choose up to $2^{64}$ (8 active bytes) differences, whereas in all other columns we have only $2^{32}$ (4 active bytes each) possible differences.

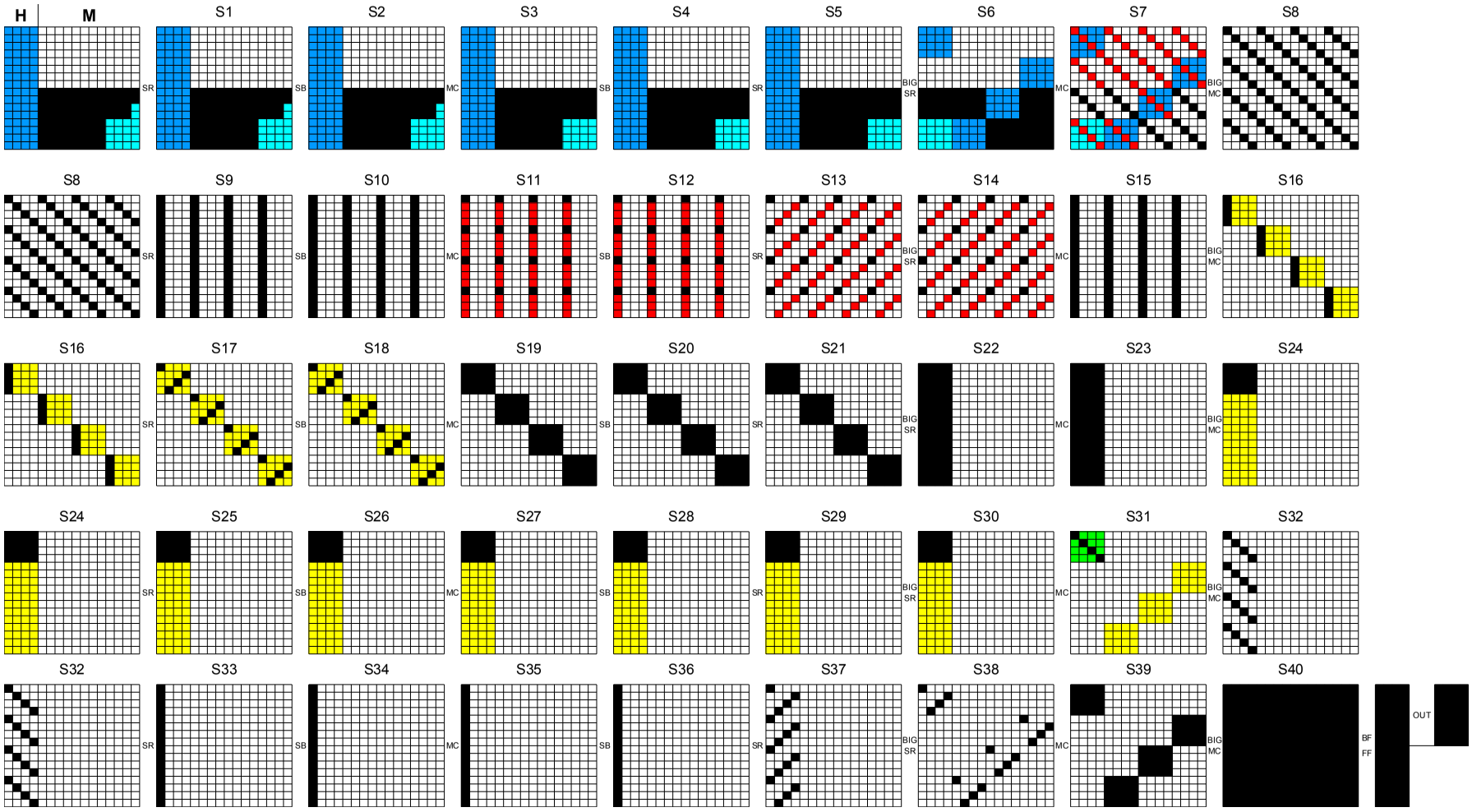


Fig. 3: The truncated differential path to get a subspace distinguisher for 5 rounds of ECHO-256. Black bytes are active, blue and cyan bytes are determined by the chaining input and padding, red bytes are values computed in the 1st inbound phase, yellow bytes in the 2nd inbound phase and green bytes in the outbound phase.

As described in Sect. 3.4, we find values according to the SuperBox differentials with an average complexity of 1 by using their DDT. Note that for some differentials no solutions exist, but for each possible differential we get more pairs which out-weight the non-existing ones (for more details we refer to [12]). For a full active AES state, one out of approximately $2^{16}$ differences gives a differential match and then, provides at least $2^{16}$ solutions. Hence, in the following it is reasonable to assume that for each differential, we get one solution with average complexity one.

Hence, after the 1st inbound phase we get $2^{32}$ independent solutions for each of the three columns in state $S_7$ (red and black bytes) with complexity $2^{32}$ in time and memory. These solutions (or pairs) consist of differences and values for the black bytes, and values for the red bytes in $S_7$. Note that for each solution (and arbitrary choice of white bytes in $S_7$) the truncated differential path from state $S_3$ to state $S_{23}$ is already fulfilled.

**Merge Chaining Input.** Next, we need to merge the solutions of the 1st inbound phase with the chaining input and bytes fixed by the padding. Therefore, we choose $2^{32}$ random first message blocks and compute the resulting chaining value after one compression function call of ECHO. Note that each AES state can be independently computed forward to state $S_7$ until the first BigMixColumns transformation. We do this for the chaining values (blue) and the AES state containing the message length (cyan). Note that we match the two remaining bytes and one bit of the padding at a later step.

We merge the $2^{32}$ chaining values with the solutions of the 1st inbound phase column by column. We start with column 0 where we need to match the padding state as well. Since we match 64 bits of overlapping red and blue/cyan bytes, the expected number of solutions is $2^{32} \times 2^{32} \times 2^{-64} = 1$. For all other columns, we need to match only 4 red bytes in each blue AES state and we get $2^{32} \times 2^{-32} = 1$ solution as well. Since we only merge lists of size $2^{32}$ the complexity of this step is $2^{32}$ in time and memory.

After this step, we have found solutions where the values of all blue, cyan and red bytes, as well as the values of the black bytes between state $S_7$ and state $S_{14}$ are determined. Furthermore, all differences (black bytes) from state $S_4$ up to state $S_{17}$ can be computed.

**2nd Inbound.** In the 2nd inbound phase, we search for values and differences such that the truncated differential path in round 3 is fulfilled (yellow). Remember that the differences in state $S_{17}$ have already been fixed due to the 1st inbound phase. We start with $2^{64}$ differences of state $S_{24}$ and compute backwards to state $S_{20}$, the output of the SuperBoxes. Note that we have 16 independent SuperBoxes for the yellow AES states between state $S_{17}$ and $S_{20}$. Again, we use the DDT of the SuperBoxes to find the according values for the given differentials. For 4 full active AES states, the probability of a possible differential is about $2^{-4 \cdot 16}$ and we get one possible differential. Note that for each possible differential, the expected number of solutions for the 2nd inbound phase is $2^{64}$. For each of these pairs, differences and values of all yellow and black bytes in round 3 are determined.

**Outbound Phase.** Next, we compute the $2^{64}$ differences and values of state $S_{24}$ forward to $S_{31}$. With a probability of $2^{-96}$ we get 4 active bytes after MixColumns in state $S_{31}$. Hence, we have to repeat the 2nd inbound phase $2^{32}$ times to find one solution for the outbound phase as well and we get a total complexity of $2^{96}$. After this step, the complete truncated differential path is fulfilled (except for two cyan bytes in the first states). Furthermore, all differences (black bytes) from state $S_4$ until state $S_{33}$ are already determined. Also, the values of the yellow, red, blue, green and cyan bytes, and the values of the black bytes from state $S_7$ to $S_{31}$ except for

state $S_{15}$ are determined. What remains is to find values for the white bytes such that the results of the two inbound phases (blue/cyan/red and yellow bytes) can be connected.

subsubsectionMerge Inbound.

To merge the two inbound phases, we need to find according values for the white bytes. We first choose random values for all remaining bytes of the first two columns in state $S_7$ and compute them forward to state $S_{14}$. Note that we need to try $2^{2\cdot 8+1}$ 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 $S_0[2,3]$. To illustrate all further steps, we use only states and colors shown in Fig. 4. Note that all gray bytes have already been determined either by an inbound phase, chaining value, padding or by just choosing random values for the remaining free bytes of the first two columns of $S_7$. Also the cyan bytes are fixed already. However, all white, red, green, yellow and blue bytes are still free to choose.

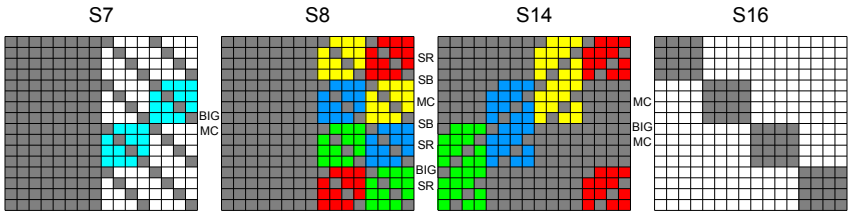


Fig. 4: States used to merge the two inbound phases with the chaining values. Gray bytes show values already determined. Green, blue, yellow and red bytes show independent values used in the generalized birthday attack and cyan bytes represent values with the target conditions.

By taking a look at the linear SuperMixColumns transformation, we observe that in each column slice, 14 out of 32 input/output values are already fixed. Note that choosing any value of two more bytes uniquely determines all remaining 16 bytes, which we can compute by solving a linear system of equations. We choose in total $2^{64}$ such values for the green, blue, yellow and red columns in state $S_{14}$ and compute them independently backward to $S_8$.

Next, 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 4 independent lists with $2^{64}$ values in state $S_8$, we can use the generalized birthday attack [17] to find one solution with a complexity of $2^{192/3} = 2^{64}$ in time and memory. In detail, we need to match values after the BigMixColumns transformation in backward direction. Hence, we first multiply each byte of the 4 independent lists by the 4 multipliers of the InvMixColumns transformation. Then, we get 24 equations containing only XOR conditions on bytes between the target value and elements of the 4 independent lists, which can be solved using a generalized birthday attack.

After this step, all values and differences are determined and we can compute the input message pair, as well as the output differences for ECHO-256 reduced to 5 rounds. By simply repeating the merge inbound phase $2^{32}$ times, we can find at least $2^{32}$ solutions for the whole truncated differential path. The total complexity is then $2^{96}$ compression function evaluations and memory requirements of $2^{64}$.

**Subspace Distinguisher.** Note that one message pair resulting in one output differences does not give a distinguisher. We need to find many output differences in a subspace with a complexity less than in the generic case. To determine the generic complexity of finding output differences in a vector space and the resulting advantage of our attack we use the subspace distinguisher. In general, the size of the output vector space is define 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 $(16+0)\cdot 8 = 128$. However, a weakness in the SuperMixColumns transformation together with the BigFinal and output truncation reduces the vector space to a dimension of 64 at the output of the hash function for the given truncated differential path.

Note that we can move the BigFinal function prior to SuperMixColumns, since BigFinal is a linear transformation and the same linear transformation $M_{\mathsf{SMC}}$ is applied to all columns in SuperMixColumns. Hence, we get 4 active bytes in each column slice at the same position in each AES state. To each (active) column slice $C_{16}$, we first apply the SuperMixColumns matrix multiplication with $M_{\mathsf{SMC}}$ and then, a matrix multiplication with $M_{trunc}$ 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 $4 \times 8$ matrix $M_{comb}$ applied to the reduce input $C_4$, which contains only the 4 active bytes of $C_{16}$:

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

The multiplication with zero differences of $C_{16}$ removes 12 columns of $M_{\mathsf{SMC}}$ while the truncation removes 8 rows of $M_{\mathsf{SMC}}$. An example for the first active column slice is given as follows:

$$\begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 4&6&2&2&6&5&3&3&2&3&1&1&2&3&1&1\\ 2&4&6&2&3&6&5&3&1&2&3&1&1&2&3&1\\ 2&2&4&6&3&3&6&5&1&1&2&3&1&1&2&3\\ 6&2&2&4&5&3&3&6&3&1&1&2&3&1&1&2\\ 2&3&1&1&4&6&2&2&6&5&3&3&2&3&1&1\\ 1&2&3&1&2&4&6&2&3&6&5&3&1&2&3&1\\ 1&1&2&3&2&2&4&6&3&3&6&5&1&1&2&3\\ 3&1&1&2&6&2&2&4&5&3&3&6&3&1&1&2\\ 2&3&1&1&2&3&1&1&4&6&2&2&6&5&3&3\\ 1&2&3&1&1&2&3&1&2&4&6&2&3&6&5&3\\ 1&1&2&3&1&1&2&3&2&2&4&6&3&3&6&5\\ 3&1&1&2&3&1&1&2&6&2&2&4&5&3&3&6\\ 6&5&3&3&2&3&1&1&2&3&1&1&4&6&2&2\\ 3&6&5&3&1&2&3&1&1&2&3&1&2&4&6&2\\ 3&3&6&5&1&1&2&3&1&1&2&3&2&2&4&6\\ 5&3&3&6&3&1&1&2&3&1&1&2&6&2&2&4 \end{bmatrix} \cdot \begin{bmatrix} a\\0\\0\\0\\b\\0\\0\\0\\c\\0\\0\\0\\d\\0\\0\\0 \end{bmatrix} = \begin{bmatrix} 4&6&2&2\\ 2&3&1&1\\ 2&3&1&1\\ 6&5&3&3\\ 2&4&6&2\\ 1&2&3&1\\ 1&2&3&1\\ 3&6&5&3 \end{bmatrix} \cdot \begin{bmatrix} a\\b\\c\\d \end{bmatrix}$$

Analyzing the resulting matrix $M_{comb}$ for all 4 active column slices shows that in each case, the rank of $M_{comb}$ is 2 instead of 4. This reduces the dimension of the vector space in each active column slice from 32 to 16. Since we have 4 active columns, the total dimension of the vector space at the output of the hash function is 64. We use [8, Corollary 1] and [8, Equation (19)] to compute the complexity of a generic distinguishing attack on the ECHO-256 hash function. We get the parameters $N = 256$ (hash function output size), $n = 64$ (dimension of 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 64 is about $2^{111.8}$ compression function evaluations. Remember that in our attack on ECHO we also get $2^{32}$ pairs in a vector space of the same dimension. Hence, the total complexity for a subspace distinguisher on 5 rounds of the ECHO-256 hash function is about $2^{96}$ compression function evaluations with memory requirements of $2^{64}$.

### 4.2 Near-Collisions for 4.5 Rounds

We can also use the truncated differential path for the subspace distinguisher on 5 rounds to get a near-collision attack for 4.5 rounds. If we remove the last MixColumns, BigShiftRows and BigMixColumns transformations we get $2^{32}$ 64-bit near-collisions (8 active bytes at the output) with a total complexity of $2^{96}$ and $2^{64}$ memory. Note that the generic complexity to find one

such near-collision is $2^{96}$ while the complexity to find $2^{32}$ near-collisions is $2^{128}$ compression function evaluations.

We can also change the truncate differential path slightly to find one 48-bit near-collision with only 6 active bytes at the output (generic complexity $2^{104}$). Instead of computing $2^{32}$ solutions after the merge inbound phase, we search for one pair according the truncated differential path of Fig. 5 in the last round. The probability for the propagation through MixColumns in state $S_{35}$ is $2^{-32}$ and we get one 48-bit near-collision with a complexity of $2^{96}$ compression function evaluations and $2^{64}$ memory.

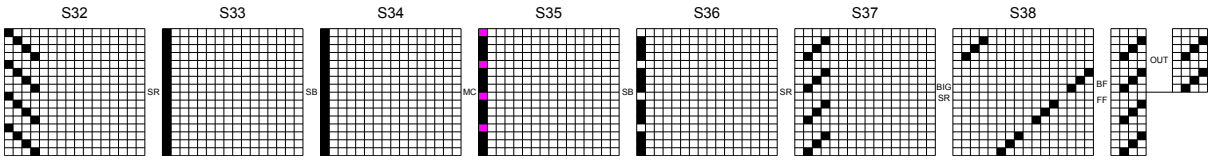


Fig. 5: The truncated differential path for the last round to get a near-collision for 4.5 rounds of ECHO-256. Black bytes are active and purple bytes are determined in the 2nd outbound phase.

Additionally, we can improve the complexity of a near-collision attack by choosing the salt value. In this case, we extend the attack by a 3rd inbound phase in round 4 instead of the outbound phase. Since the salt has to be determined first, we need to compute the 2nd and 3rd inbound phase before the 1st inbound phase. Note that the 128-bit salt value is added after the second AES MixColumns transformation in each round of ECHO. Since the same salt value is added in every AES state and BigMixColumns is a linear operation, we can move the XOR-addition of the salt after BigMixColumns. Then, the whole 2nd and 3rd inbound phase is independent of the salt value and we use the freedom in the salt to connect the two inbound phases.

In detail, we first compute the resulting difference of the 2nd inbound phase forward to state $S_{25}$ and one difference of state $S_{31}$ backward to state $S_{28}$. Then, we use the DDT of the SuperBoxes to find values according to the given differential or repeat with another difference of state $S_{31}$. This step determines values and differences of the green and black bytes of round 4. Since the differences between the 2nd and 3rd inbound phase already match we just need to choose the salt such that the values in the active AES state match as well. Since we do not reduce the number of active bytes in the outbound phase, the total complexity to find a 64-bit near-collision with chosen salt (8 active bytes at the output) is $2^{64}$ in time and memory.

### 4.3 Collisions for 4 Rounds

Finally, we are able to construct collisions for 4 rounds of the ECHO-256 hash function. The attack and truncated differential path is similar as for the subspace distinguisher on 5 rounds. We use a two-block message and the truncated differential path for the second block is given in Fig. 6. Again, we start with the 1st inbound phase, merge the chaining input and continue with the 2nd inbound and outbound phase. To get a collision at the output we use differences in the feed-forward and do a 3rd inbound phase in two AES states in round 1. Finally, we merge the solutions of the two inbound phases to determining the remaining values. In the following, we only describe parts of the attack which are new or have been changed.

**1st Inbound.** The 1st inbound phase is the same as for the subspace distinguisher, except that in state $S_7$ we choose $2^{64}$ random differences for the active bytes of column 0 and 1, and

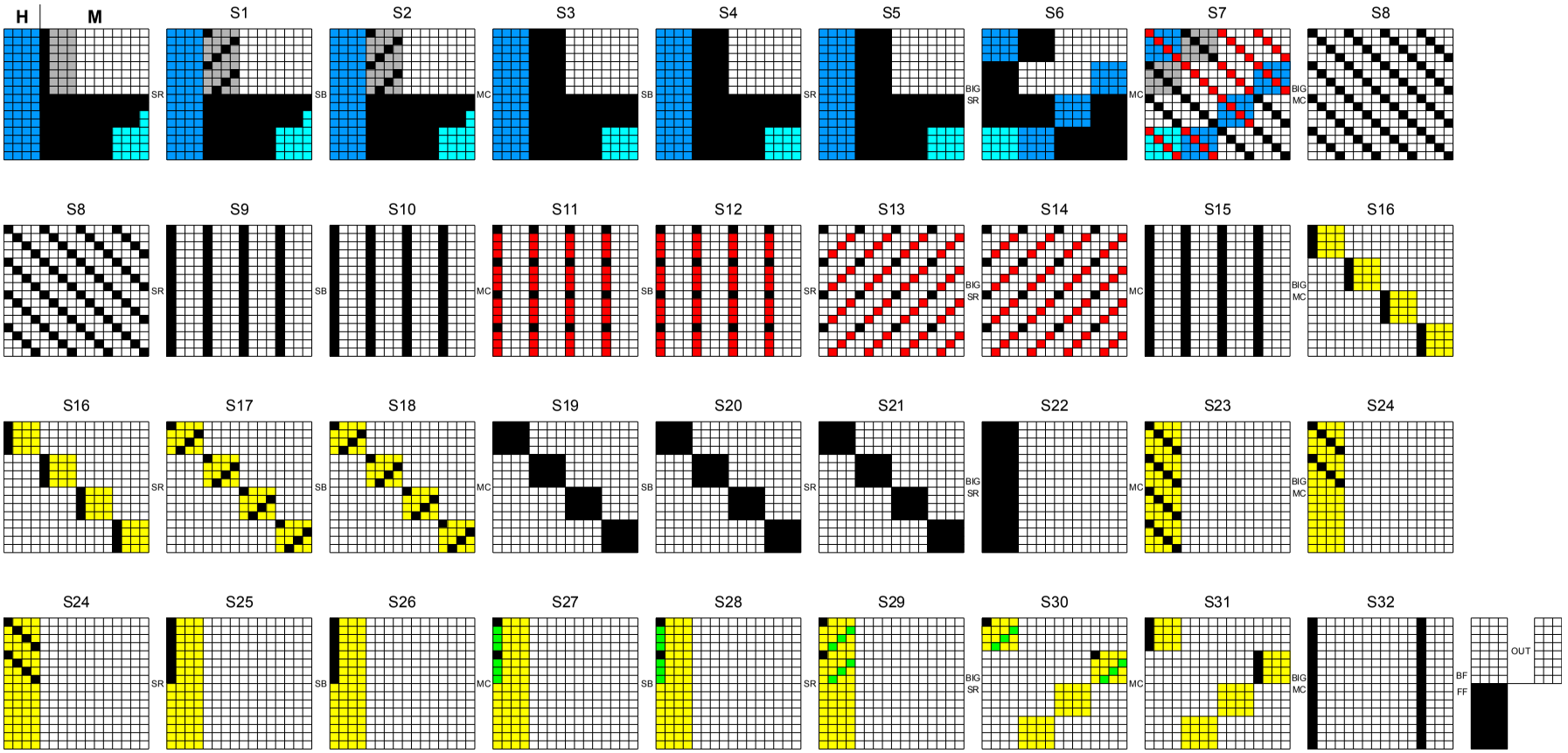


Fig. 6: The truncated differential path to get a collision on 4 rounds of ECHO-256. Black bytes are active, blue and cyan bytes are determined by the chaining input and padding, red bytes are values computed in the 1st inbound phase and yellow bytes in the 2nd inbound phase, green bytes in the outbound phase and gray bytes in the input inbound phase.

$2^{32}$ random differences for column 2 and 3. Hence, after the 1st inbound phase we get $2^{64}$ independent solutions for each of the first two columns, and $2^{32}$ solutions for each of the last two columns in state $S_7$ (red and black bytes).

**Merge Chaining Input.** Again, we choose $2^{32}$ random first message blocks and merge them with the solutions of the 1st inbound phase column by column. We start with column 0 where we need to match the padding state as well. Since we match 64 bits of overlapping red and blue/cyan bytes, the expected number of solutions is $2^{32} \times 2^{64} \times 2^{-64} = 2^{32}$. For all other columns, we need to match only the 4 red bytes in each blue AES state. For column 1 we get $2^{32}$ solutions since we have computed $2^{64}$ results in the 1st inbound phase. For column 2 and 3, we have $2^{32}$ solutions from the 1st inbound phase and get one match on the overlapping 4 bytes.

**2nd Inbound.** To get the first solution for the 2nd inbound phase we need to try $2^{64}$ differences of state $S_{24}$, since the probability of a differential match in 4 full active AES states is only about $2^{-4 \cdot 16}$. However, the expected number of solutions for the 2nd inbound phase is $2^{64}$ but we only need $2^{48}$ of them to continue.

**Outbound Phase.** In the outbound phase we compute these $2^{48}$ differences and values of state $S_{24}$ forward to $S_{27}$. With a probability of $2^{-48}$ we get one active byte after MixColumns in each active state of $S_{27}$. After this step, the complete truncated differential path (except for the three first states) is fulfilled. What remains is to determine differences in the first state to get a collision at the output and to find values for the white bytes.

**3rd Inbound.** To get a collision at the output, we use two additional active AES states in round 1. In $S_0[0,1]$ and $S_0[1,1]$, only the first column should be active such that the active bytes overlap with the active bytes at the output. For these active bytes at the input, we choose the same differences as in the output in $S_{32}[0,0]$ and $S_{32}[1,0]$. Then, these differences cancel each other by the feed-forward and we get a collision. In a 3rd inbound phase, we determine the remaining values of the gray and black bytes such that the given truncated differential for these two AES states in round 1 is satisfied. Again, we can find such values and differences with a complexity of about 1 using the DDT of the SuperBoxes, and compute $2^{32}$ solutions for each AES state. Since we still have $2^{32}$ solutions for each of column 0 and column 1 due to the 1st inbound phase, we expect to find a match for both differences and values of the overlapping 4 diagonal bytes of AES state $S_7[0,1]$ and $S_7[1,0]$.

**Merge Inbound.** Finally, we merge the 1st and 2nd inbound phases as in the previous attacks. Then, all values and differences are determined and we can compute the input message pair which results in a collision for ECHO-256 reduced to 4 rounds. The total complexity is $2^{64}$ in time and memory.

## 5 Attacks on the ECHO-256 Compression Function

The attack on the hash functions of ECHO can be extended to the compression function almost in a straight-forward way. In this case, instead of the chaining value, a 512-bit value of another inbound phase is merged with the 1st inbound phase. In fact we can repeat 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 get in total an attack

on 7 rounds of the compression function of ECHO-256. We get similar results also for ECHO-512 (see App. A). To reduce the complexity of the attacks, we also choose the salt input of the compression function.

### 5.1 The Truncated Differential Path

We use the 7-round truncated differential path given in Fig. 7. Black bytes are active and colored bytes are determined by the different inbound and outbound phases. Since this path is sparse, we do not have problems with missing degrees of freedom and are able to find many pairs according to this path. In more detail, we can compute this number already by considering the reduction of pairs in the MixColumns transformations. Remember that a truncated differential path through the SuperMixColumns transformation from $4 \rightarrow 16 \rightarrow 4$ active bytes has a probability of $2^{-24}$. At the input, we start with $16 \cdot 16$ byte values, 16 byte differences and a 128-bit salt. We get a reduction of pairs at the 1st MC and SMC of round 1, the 2nd MC of round 3, the 1st MC and SMC of round 4, the BMC of round 5 and the 2nd MC of round 6. Hence, we get the following expected number of pairs (in base 2 logarithm) for the whole path:

$$8 \cdot (16 \cdot 16 + 16 + 16 - 12 - 3 - 48 - 48 - 12 - 48 - 12) = 8 \cdot 105 \tag{2}$$

In other words, the expected number of pairs conforming to this truncated differential path is $2^{8 \cdot 105} = 2^{800}$, which is much more than for the paths given in [11] and [16].

### 5.2 Outline of the Attack

The attack on the compression function consists of two more inbound phases in round 2 and 3. Instead of merging with the chaining input, we merge the results of the 1st inbound phase with the results of these new inbound phases. Hence, we start with the 1st inbound phase in round 4, compute the 3rd and 4th inbound phase and then, merge the results of these inbound phases. We continue with the 2nd inbound phase, the outbound phase and finally, merging all inbound phases by finding solutions for the remaining white bytes, similar as in the hash function attack. Note that we slightly change the 1st inbound phase. We choose only one difference for the active bytes in state $S_{23}$ but $2^{32}$ differences for the active bytes in state $S_{30}$. According to the SuperMixColumns transformation in round 4, $2^{32}$ such differences exist.

**3rd Inbound.** In the 3rd inbound phase, we search for values and differences such that the truncated differential path in round 3 is fulfilled. Furthermore, we need to ensure that the results can be connected with the solutions of the 1st inbound phase. We have $2^{32}$ solutions for the 1st inbound phase and choose $2^{32}$ values of the salt corresponding to the active diagonal bytes (red an black bytes) in state $S_{23}$. Hence, we get in total $2^{64}$ values with equal differences for the red and black bytes of state $S_{23}$. Next, we compute the difference of $S_{23}$ backward to state $S_{20}$, choose a random difference in state $S_{15}$ and compute forwards to state $S_{17}$. Again, we have 16 independent active SuperBoxes in round 3 and can use the DDT to find according values for the given differentials with an average complexity of 1. We need to match the overlapping values of the 16 black bytes in state $S_{23}$. We repeat the previous step with $2^{96}$ random differences of state $S_{15}$ to get $2^{96} \times 2^{64} \times 2^{-128} = 2^{32}$ solutions.

**4th Inbound.** The 4th inbound phase is applied in round 2. We first choose a difference according to the truncated differential path through SuperMixColumns in round 1 and compute this difference forward to state $S_9$. Furthermore, we compute the differences of the $2^{32}$ solutions

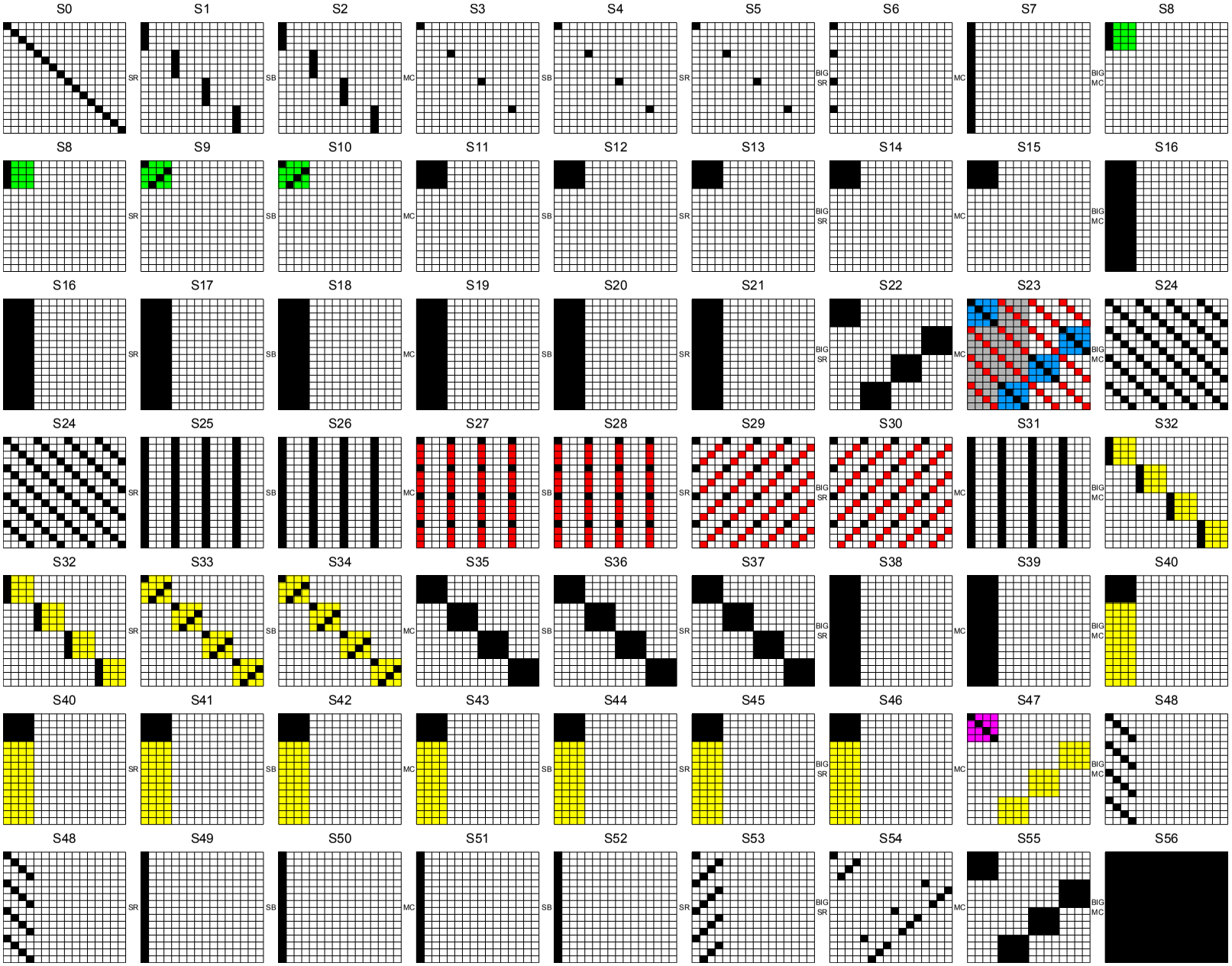


Fig. 7: The truncated differential path to get a subspace distinguisher for 7 rounds of the ECHO-256 and ECHO-512 compression function with chosen salt. Black bytes are active, red bytes are values computed in the 1st inbound phase, yellow bytes in the 2nd, blue in the 3rd and green bytes in the 4th inbound phase. Purple bytes are determined in the outbound phase and gray bytes are chosen randomly. Note that by omitting the last round, we get near-collisions for the compression function.

of the 3rd inbound phase backwards to state $S_{12}$. We use the DDT of the SuperBoxes to get $2^{32}$ solutions for the 4th inbound phase. Next, we need to connect the values of the 3rd and 4th inbound in state $S_{15}$. Among the $2^{32}$ solutions of the 4th inbound phase, we expect to find one match for the 4 bytes of the diagonal of state $S_{15}[0,0]$. For all other bytes, we simply choose the remaining 12 bytes of the salt accordingly.

**Outbound Phase.** In the 2nd inbound phase, we compute $2^{96}$ solutions and propagate each of them outwards to state $S_{47}$. With a probability of $2^{-96}$ we get 4 active bytes after MixColumns in state $S_{47}$ and find one such pair. The remaining part of the truncated differential path holds with probability 1. Hence, for this resulting pair, the complete 7-round truncated differential path is fulfilled.

**Merge Inbound.** What remains is to find values for all white bytes. We can choose random values for the gray bytes in round 3 and continue with merging the inbound phases as in Sect. 4.1 of the hash function attack. Hence, the overall complexity to find one pair according to the truncated differential path for the permutation is $2^{96}$ with $2^{64}$ memory.

### 5.3 Subspace Distinguisher for 7 Rounds

To distinguish 7 rounds of the ECHO-256 compression function from an ideal compression function, we use the complete 7-round truncated differential path to get a subspace distinguisher. Since the output is not truncated, the size of the vector space is define by the number of active bytes prior to the MixColumns and BigMixColumns transformations in the last round (16 active bytes), and the number of active bytes at the input due to the feed-forward (16 active bytes). Since in total 32 active bytes determine the vector space we get a dimension of $32 \cdot 8 = 256$. Again, we use [8, 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), $n = 256$ (dimension of vector space) and $t = 2^{11}$ (number of outputs in vector space) for the subspace distinguisher. Then, the generic complexity to construct $2^{11}$ elements in a vector space of dimension 256 is about $2^{117.3}$ compression function evaluations. Note that we also need to repeat our attack on the ECHO-256 compression function about $2^{11}$ times to get enough differences in the vector space. Hence, the total complexity for the subspace distinguisher on the compression function is about $2^{107}$ with memory requirements of $2^{64}$ with chosen salt.

### 5.4 Near-Collision for 6.5 Rounds

Due to the truncating output transformation of ECHO, near-collisions are also interesting for the compression function. To construct near-collisions for up to 6.5 rounds of the compression function, we use the same truncated differential path as before but remove the last MixColumns, BigShiftRowsand BigMixColumns transformation. We get a 128-bit near-collision for the 512-bit compression function of ECHO-256 (4 diagonals with 4 bytes are active) and 6 rounds, or a 192-bit near-collision (6 bytes in 4 AES states are active) and 6.5 rounds with a complexity of $2^{96}$ and $2^{64}$ memory with chosen salt. Because of the padding, this near-collision attack does not apply to the last compression function call of ECHO-256.

## 6 Conclusion

In this work we have presented the first analysis of the ECHO hash function. We give a subspace distinguisher for 5/8 rounds, near-collisions for 4.5/8 rounds and collisions for 4/8 rounds of the

ECHO-256 hash function. These results greatly improve upon the previous results which are only on the (double-pipe) compression function of ECHO and for less rounds. Note that near-collisions are a requirement [15] for a future SHA-3 and due to the distinguisher, ECHO-256 reduced to 60% of its rounds cannot be considered as an ideal function. We have also analyzed the compression functions of ECHO to get a distinguisher for 7/8 rounds and near-collisions for up to 6.5/8 rounds of ECHO-256 with chosen salt. We get similar compression function results also for ECHO-512.

The main idea of our improvement is to combine the last MixColumns with the following BigMixColumns to a SuperMixColumns transformation, which allows to construct very sparse truncated differential paths. In these paths, at most 1/4 of the bytes are active throughout the whole computation of ECHO. This behavior is not known from the AES or AES based hash functions which strictly follow the wide-trail design strategy. Future work will include to search for even sparser truncated differential paths and the improvement of the given attacks, especially for ECHO-512.

### Acknowledgements

We thank all members of the IAIK Krypto group for useful discussions, and the designers of ECHO for their comments. This work was supported in part by the Austrian Science Fund (FWF), project P21936, by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II, and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

## References

1. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST, 2008. Available online: http://crypto.rd.francetelecom.com/echo.
2. Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org/.
3. Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.
4. Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001.
5. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer, 2002.
6. Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *LNCS*, pages 78–94. Springer, 2006.
7. Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations. FSE, 2010. to appear.
8. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 126–143. Springer, 2009.
9. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. Cryptology ePrint Archive, Report 2010/198, 2010. http://eprint.iacr.org/.
10. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolic, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full Lane Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 106–125. Springer, 2009.
11. Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *LNCS*, pages 16–35. Springer, 2009.
12. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.

13. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *LNCS*, pages 350–365. Springer, 2010.
14. National Institute of Standards and Technology (NIST). FIPS PUB 197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197, U.S. Department of Commerce, November 2001. Available online: `http://www.itl.nist.gov/fipspubs`.
15. National Institute of Standards and Technology (NIST). Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
16. Thomas Peyrin. Improved Differential Attacks for ECHO and Grøstl. Cryptology ePrint Archive, Report 2010/223, 2010. `http://eprint.iacr.org/`.
17. David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.

## A Results on the ECHO-512 Compression Function

The truncated differential path to analyze the ECHO-256 compression function can also be used for the compression function of ECHO-512. The truncated differential path of the permutation is exactly the same, but the active bytes for the near-collision and the complexities for the subspace distinguisher change. The complexity to find one solution for the permutation is $2^{96}$ with $2^{64}$ memory (with chosen salt). In the case of ECHO-512, we can also find one solution for the permutation without chosen salt by repeating the attack $2^{128}$ times and we get a complexity of $2^{224}$ with $2^{64}$ memory. Note that we have enough different starting points in state $S_{23}$ and $S_{15}$. Alternatively, we can also fix the 144-bit padding of the last compression function call. Then, the total complexity to find one solution is $2^{240}$ and $2^{64}$ memory with chosen salt.

### A.1 Subspace Distinguisher for 7 Rounds

For the compression function of ECHO-512, the parameters of the subspace distinguisher are $N = 1024$ (compression function output size), $n = 256$ (dimension of vector space) and $t = 2^{10}$ (number of outputs in vector space). The generic complexity for a subspace distinguisher is $2^{292.9}$ compression function evaluations. With chosen salt, we get for the total complexity of the subspace distinguisher on the ECHO-512 compression function about $2^{106}$ with memory requirements of $2^{64}$, and without chosen salt we get $2^{234}$ and $2^{64}$ memory.

### A.2 Near-Collision for 6.5 Rounds

In the case of ECHO-512, we get a 192-bit near-collision for the 1024-bit compression function for 6 rounds (6 diagonals with 4 bytes are active). If we remove the last MixColumns, BigShiftRows and BigMixColumns transformation, we get a 224-bit near-collision for 6.5 rounds (2 AES states with 6 bytes and 4 AES states with 4 bytes are active). With chosen salt, the complexity is $2^{96}$ and $2^{64}$ memory, and without chosen salt we get a complexity of $2^{224}$ and $2^{64}$ memory.