Improved Collisions for Reduced ECHO-256

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 a collision attack on 5/8 rounds of the ECHO-256 hash function with a complexity of 2^{112} in time and $2^{85.3}$ memory. In this work, we further show that the merge inbound phase can still be solved in the case of hash function attacks on ECHO. As correctly observed by Jean et al., the merge inbound phase of previous hash function attacks succeeds only with a probability of 2^{-128}. The main reason for this behavior is the low rank of the linear SuperMixColumns transformation. However, since there is enough freedom in ECHO we can solve the resulting linear equations with a complexity much lower than 2^{128}. On the other hand, also this low rank of the linear SuperMixColumns transformation allows us to extend the previous collision attacks by one more round. Additionally, we present a collision attack on 6 rounds of the compression function of ECHO-256 and show that a subspace distinguisher is still possible for 7 out of 8 rounds of the compression function of ECHO-256. Both attacks have a complexity of 2^{160} with memory requirements of 2^{128} and chosen salt.

Keywords: hash functions, SHA-3 competition, ECHO, cryptanalysis, truncated differential path, rebound attack, collision attack

1 Introduction

Many new and interesting hash function designs have been proposed in the NIST SHA-3 competition [7]. In this paper, we improve the cryptanalysis of 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 standard AES rounds are used. The best result on the hash function of ECHO-256 is a collision attack for 4 rounds and a subspace distinguisher for 5 rounds [8]. In this work, we show how to extend these attacks to get collisions for 5 rounds of the ECHO-256 hash function.

As correctly observed by Jean et al. in [3], the merge inbound phase of [8] succeeds only with a probability of 2^{-128}. The main reason for this behavior is the low rank of the linear SuperMixColumns transformation. Jean et al. also present a solution to efficiently solve the merge inbound phase but are able to attack 4/8 rounds of the ECHO-256 compression function. They present a semi-free-start collision attack on 4/8 rounds with complexity 2^{52} and 2^{16} memory and have implemented a semi-free-start near-collision attack on 4/8 rounds. In this work, we show how to solve the merge inbound phase for up to 7 rounds of the compression function, as well as for the *hash* function of ECHO-256. Since in the attacks on ECHO we have enough freedom and many parts can be fulfilled independently, the resulting linear equations can still be fulfilled with a complexity much lower than 2^{128}. Additionally, the low rank of the linear SuperMixColumns transformation allows us to extend the previous collision attacks on the hash function by one more round.

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]. ECHO 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, followed by the 16-byte message length.

The compression function of ECHO uses one internal 2048-bit permutation P which manipulates 128-bit words similar as 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 4×4 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 also tweaked by the counter c_i and salt s) are the previous chaining variable H_{i-1} and the current message block M_i, 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_{MC} but applied to those bytes with equal position inside the AES states.

For an easier description of the following attacks, 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 results in the two super-round transformations SuperBox and SuperMixColumns which are separated just by byte shuffling operations. For more details on these transformations and their differential properties we refer to [8].

3 Collisions for 5 Rounds of the ECHO-256 Hash Function

Three improvements are needed to get a collision for 5 rounds. Firstly, we split the merge inbound phase into two parts. In the first part, we efficiently fulfill the 128-bit condition raised in [3] such that a hash function attack is still possible. Note that this also changes the inbound phases slightly. Secondly, we show that the resulting differences in the subspace of the hash function output collide with a probability of 2^{-64}. Finally, we improve the second part of the merge inbound phase which determines the remaining white bytes to get an average complexity of $2^{21.3}$ to compute one such solution. Then, the total complexity to get a collision for 5 rounds of ECHO-256 is about 2^{112} with memory requirements of $2^{85.3}$. For this attack, we use exactly the same truncated differential path as in [8], except that we get a collision at the output (see Fig. 1).

3.1 Solving and Merging the Inbound Phases

To find input pairs according to the truncated differential path given in Fig. 1, we use a rebound attack [6] with multiple inbound phases [4, 5, 8]. 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). In the yellow inbound and green outbound phase we construct (partial) pairs such that the truncated differential path in round 3, 4 and 5 is fulfilled. In the red inbound phase we search for many pairs conforming to the red part. Additionally, we ensure the 128-bit condition such that the yellow and red part can be connected. Then, we merge (connect) the resulting pairs of the red and yellow inbound phase with the chaining input (blue) and padding (cyan). Note that for each found pair in these phases, the white bytes are still free to choose. We find values which also satisfy the remaining white bytes in the subsequent sections.

Yellow Inbound and Green Outbound. We start the attack by choosing a difference in state S_{16} such that the truncated differential path of SuperMixColumns between state S_{14} and S_{16} is fulfilled. Then, we can find one pair for the yellow and green part with a complexity of 2^{96} and memory requirements of 2^{64}. Since we do not change this part of the attack, we refer to [8] for more details on how to find such a pair. Note that this step determines the values of all yellow and black bytes in state S_{16}.

Red Inbound. In the red inbound phase, we search for many pairs according to the truncated differential path between state S_7 and S_{14}. The difference in S_{14} is already fixed due to the yellow inbound phase but we can still choose from 2^{32} differences for each active AES state in S_7. As shown in [8] and previous rebound attacks [6], we can find one pair on average for each starting difference in an inbound phase. Note that we can independently search for pairs for each BigColumn of state S_7 since the four BigColumns stay independent until they are mixed by the following BigMixColumns transformation between state S_{15} and S_{16}. Hence, we can independently iterate through all 2^{32} starting differences for the 1st, 2nd and 3rd column, and through all 2^{64} starting differences for the

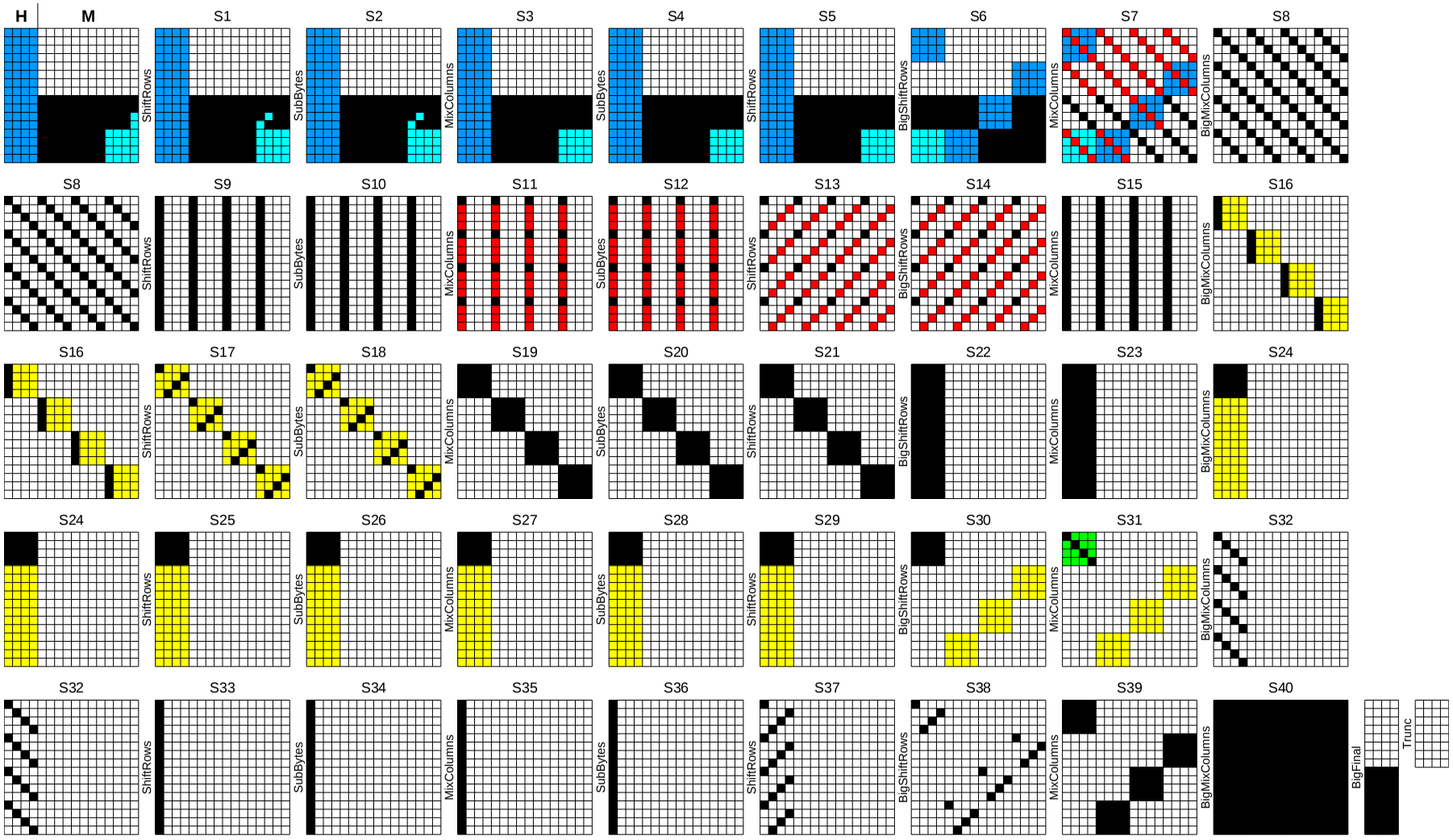

Fig. 1. The truncated differential path to get a collision for 5 rounds of ECHO-256. The differences in the output subspace collide with a probability of 2^{-64}. Black bytes are active, blue and cyan bytes are determined by the chaining input and padding, red bytes are values computed in the red inbound phase, yellow bytes in the yellow inbound phase and green bytes in the outbound phase.

4th column of state S_7. Then, we get 2^{32} pairs for each of the first three columns and 2^{64} pairs for the 4th column. The total complexity to find all these pairs is 2^{64} and determined by the last column (also see [8]). For each pair, the red and black bytes in state S_{14} are determined.

Additional Conditions at SuperMixColumns. 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 and we get for the first column-slice the following relation

$$M_{SMC} \cdot \left[a_0 * * * a_1 * * * a_2 * * * a_3 * * *\right]^T = \left[b_0\, b_1\, b_2\, b_3 * * * * * * * * * * * *\right]^T,$$

where 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 marked by $*$ are free to choose. We get a similar relation for all other 16 column-slices. As correctly observed by Jean et al. [3], we only get a solution with probability 2^{-8} for each column-slice due to the low rank of the M_{SMC} matrix. In total, each pair of the red and yellow inbound phase imposes a 128-bit condition on the whole state due to the SuperMixColumns transformation between state S_{14} and S_{16}. In detail, we get for each of the 16 column-slices one linear 8-bit conditions on the 8 values of the red, yellow and black bytes. The 8-bit condition for the first column-slice is then

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

and similar conditions exist for all other columns.

1st Part of the Merge Inbound Phase. At this point, we have constructed one pair for the yellow inbound phase and in total, $2^{32} \cdot 2^{32} \cdot 2^{32} \cdot 2^{64} = 2^{160}$ 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 on SuperMixColumns between state S_{14} and S_{16}. In the following, we show how to find all these 2^{32} pairs with a complexity below 2^{128}. First, we combine the $2^{32} \cdot 2^{32} = 2^{64}$ pairs determined by the first two columns of state S_7 in a list L_1, and the $2^{32} \cdot 2^{64} = 2^{96}$ pairs determined by the last two columns of state S_7 in a list L_2. Note that the pairs in these two lists are independent. Hence, we can use a birthday attack to find those pairs which satisfy the 128-bit condition imposed by SuperMixColumns. This way, we get $2^{64} \times 2^{96} \times 2^{-128} = 2^{32}$ pairs with a total complexity of 2^{96}. 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 instead. In detail, we first separate Equation 2 into terms determined by L_1 and terms determined by L_2:

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

Then, 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. Finally, we just iterate through all elements of L_2 and collect the 2^{32} pairs which satisfy the 128-bit condition. These 2^{32} pairs are then valid partial pairs for the combined red and yellow inbound phase. Note that the complexity of this part can probably be further reduced using the techniques proposed in [3].

Merge Chaining Input. Next, we need to merge the 2^{32} results of the previous phase 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 $5 \cdot 4 = 20$ bytes. This results in a 160-bit conditions on the overlapping blue/cyan/red bytes. We use 2^{112} randomly generated first message blocks to find a pair according to this condition. Additionally, we repeat from the yellow inbound phase with 2^{16} different starting points. This way, we get $2^{16} \cdot 2^{32} = 2^{48}$ pairs for the combined yellow and red inbound phases which also satisfy the 128-bit condition of SuperMixColumns between state S_{14} and S_{16}. Next, we do a birthday match on the overlapping 160-bits and get $2^{112} \times 2^{48} \times 2^{-160} = 1$ final pair. If we compute the 2^{112} blocks online, the complexity of this step is 2^{112} with memory requirements of 2^{48}. For the resulting pair, all differences (black) between state S_4 and state S_{33}, and all colored values (blue, cyan, red, yellow, green) between state S_0 and state S_{31} are determined.

3.2 Colliding Subspace Differences

As shown in [8], we can combine the linear MixColumns and BigMixColumns transformations with the BigFinal function and the final output truncation. Note that in all these transformations, the resulting one-byte columns of the output hash value can be computed independently of each other. Further, column $i \in \{0, 1, 2, 3\}$ of the output hash value depends only on columns $i \cdot 4$ 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 4 active differences in columns 0, 4, 8, and 12 of state S_{38} which we denote by a, b, c, d. Using M_{comb} of the first output column (see [8, Section 3.3]), we get the following linear system of equations:

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

Since we cannot directly control the differences a, b, c, d in the attack, we need to solve 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 2, 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 each of the 4 output column matrices is 2 as well, we get a collision at the output of the hash function with a total probability of 2^{-64} for any random difference in state S_{38} of Fig. 1.

3.3 Improved Merge Inbound Phase

To get a collision attack with a complexity below 2^{128} for 5 rounds, we need to improve the merge inbound phase further. We start exactly as in the merge inbound phase of the hash function attack in [8]. First, we choose random values for the white bytes in the first two BigColumns of state S_7 and propagate the resulting pairs forward to state S_{14}. Note that we again need to ensure that the conditions on the linear SuperMixColumns transformation are fulfilled. We can solve this part with a complexity of at most 2^{64} in

time and memory and we refer to [8] for a detailed description of this part. At this point, all gray values of Fig. 2 are determined and we know that for these values a solution according to SuperMixColumns exists.

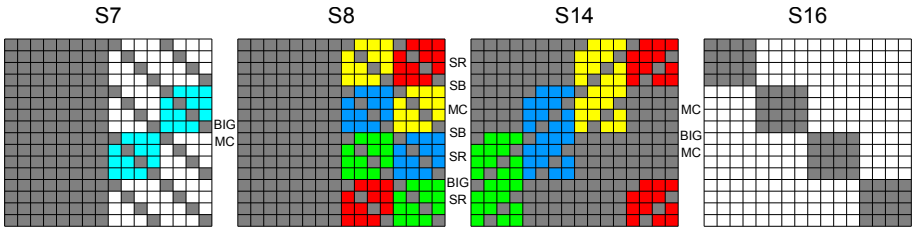

Fig. 2. States used to merge the two inbound phases with the chaining values. Gray bytes show values already determined. Green, blue, yellow and red bytes show independent values used in the generalized birthday attack and cyan bytes represent values with the target conditions.

After this part, we can choose from up to 2^{384} values for each of the green, blue, yellow and red column bytes in state S_{14} (see [8]). In the last part of the merge inbound phase, we do a generalized birthday attack to find values which also match the 24 cyan bytes (a 192-bit condition) in state S_7 of Fig. 2. First, we choose 2^{64} values for each of the green, blue, yellow and red columns in state S_{14} and independently compute them backward to state S_8. Since we get 4 independent lists with 2^{64} values in state S_8, we can use the generalized birthday attack [10] to find one solution with a complexity of $2^{192/3} = 2^{64}$ in time and memory.

To improve the average complexity of this generalized birthday attack, we can start with $2^{85.3}$ values for the green, blue, yellow and red columns in state S_{14}. Since we need to match a 192-bit condition, we get $2^{3 \cdot 85.3} \times 2^{-192} = 2^{64}$ solutions with a complexity of $2^{85.3}$ in time and memory, or with an average complexity of $2^{21.3}$ per solution (see [10] for more details). Note that we can even find solutions with an average complexity of 1 by starting with lists of size 2^{96}. Each of these 2^{64} solution of the generalized birthday match results in a valid pair conforming to the whole 5-round truncated differential path. According to the previous section, among these 2^{64} pairs we expect to find one pair which collides at the output of the hash function. The maximum time complexity is determined by merging the chaining input and the memory complexity by the generalized birthday attack. In total, the complexity to find a collision for 5 rounds of the ECHO-256 hash function is 2^{112} compression function evaluations with memory requirements of $2^{85.3}$.

4 Compression Function Attacks for Reduced ECHO-256

In this section we show how to get a collision attack for 6 and a subspace distinguisher for 7-rounds of the ECHO-256 compression function with chosen salt. Note that in these attacks we also fulfill the 128-bit conditions on SuperMixColumns observed by Jean et al. in [3]. For both attacks we get a complexity of 2^{160} with memory requirements of 2^{128}.

4.1 Finding Pairs for 6 Rounds of the Truncated Differential Path

First, we show how to find a pair for the first 6 rounds of the 7-round truncated differential path given in [9]. This 6-round truncated differential path (with a slightly different 7th

round) is shown in Fig. 3. In this figure, black bytes are active and colored bytes show the different inbound and outbound phases. The complexity to find one such pair for the first 6 rounds of the truncated differential path is 2^{160} in time with 2^{128} memory. We use this path to get a collision for 6 rounds and a distinguisher for 7 rounds of the ECHO-256 compression function with chosen salt.

Outline of the Attack. The main idea of the attack is to find solutions for the forward and backward part independently for a fixed differences between state 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 can also find 2^{128} pairs but with a complexity of 2^{160} and chosen salt. Then, we just need to match the 128-bit salt value of the forward and backward part and fulfill the 128-bit condition on the input (red) and output (yellow) values of SuperMixColumns. Since we get 2^{128} independent pairs for both the forward and backward part, we can again fulfill the resulting 256-bit condition using a meet-in-the-middle approach.

Yellow Inbound Phase. Again we start the attack with the SuperMixColumns transformation between the yellow and red part. 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 do an inbound phase between state S_{32} and S_{40}. Since we get one solution on average and with average complexity one, we can 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.

Purple Outbound Phase. 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 choose the 128-bit salt value accordingly. Hence, we get 2^{128} pairs with complexity 2^{128} according to the truncated differential path from state S_{32} to S_{48}.

Red Inbound Phase. The red inbound phase is the same as in the hash function attack. We start with the difference between state S_{30} and S_{32}, which has been chosen in the yellow inbound phase. Then, we do 4 independent inbound phases for each BigColumn in state S_{23}. Since we can start with 2^{32} differences for each column in S_{23}, we also get 2^{32} pairs for each column with a total complexity of 2^{32}.

Blue Inbound Phase. 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 do the blue inbound phases independently for each active AES state in backward direction. For each column, we get 2^{32} pairs with a complexity of 2^{32}.

Merge Blue and Red Inbound Phase. When merging the solutions of the blue and red inbound phase, 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} solutions. Furthermore, we can choose the salt value again. We start by matching the differences in the overlapping 4 bytes of each column. Since we have 2^{32} solutions for each of the blue and the red part,

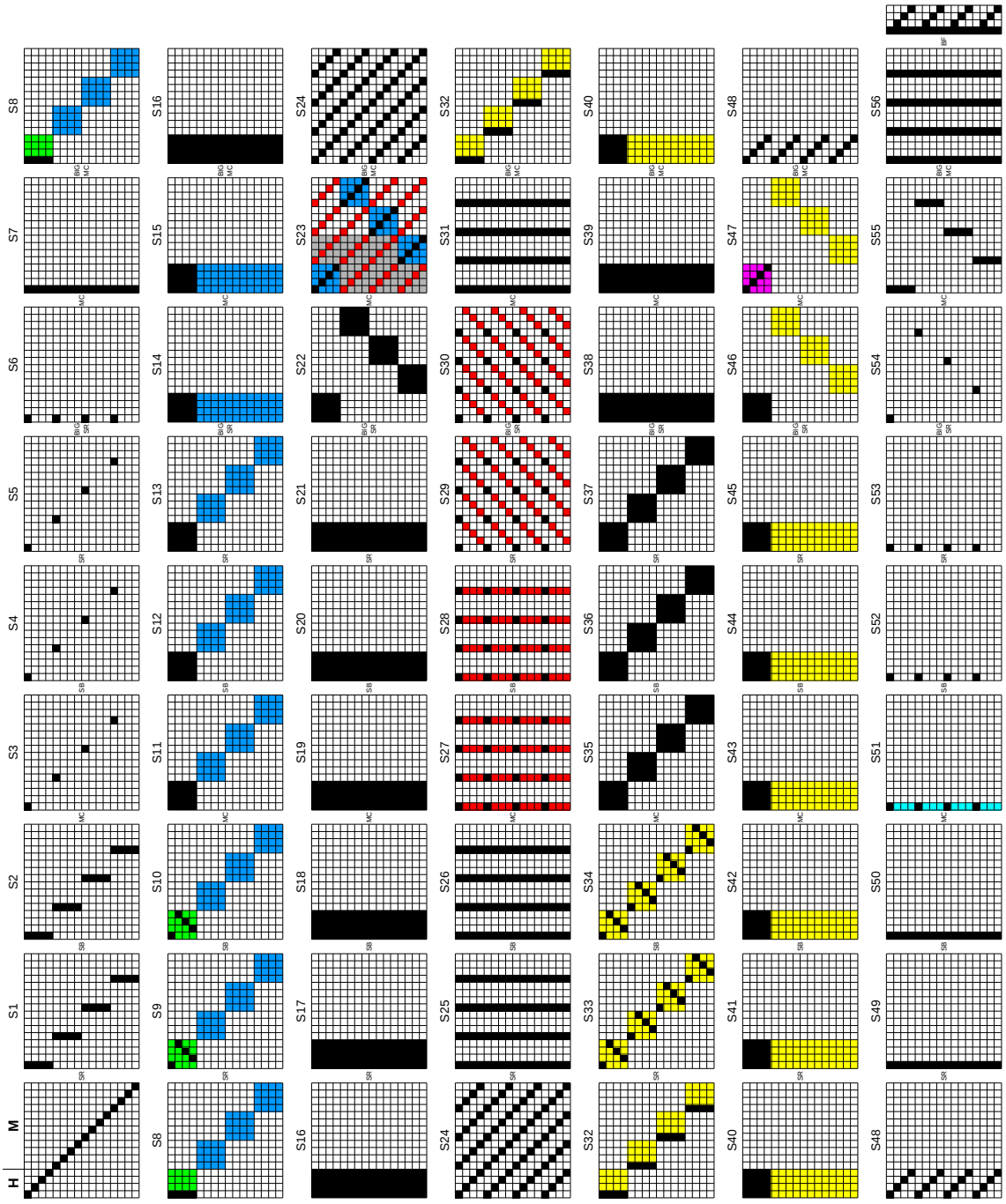

Fig. 3. The truncated differential path to get collisions for 6 rounds and near-collisions for 7 rounds of the ECHO-256 compression function. Black (and brown) bytes are active, red bytes are values computed in the 1st inbound phase, yellow bytes in the 2nd, blue bytes in the 3rd and green bytes in the 4th inbound or 2nd outbound phase, and cyan bytes in the 3rd outbound phase. Brown bytes in state S_{15} and S_{23} are active and show the position where the salt is chosen in the chosen-salt attack. Purple bytes are determined in the 1st outbound phase and gray bytes are chosen in the merge inbound phase.

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 choose (only) the diagonal 4 bytes of the salt value. For each of the 2^{32} pairs with matching difference, we compute the diagonal bytes of the salt such that the values match. We sort the resulting list according to the 4-byte salt value and repeat the same for all 4 BigColumns of state S_{23}. Then, we just need to iterate through all 4 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.

Green Inbound Phase. To find a pair also for the green part, we first choose a difference according to the truncated differential path between state S_6 and S_8. The second starting point for the green inbound phase is the difference of 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. First, we 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 pair such that the diagonal 4-byte values between the green and blue inbound phase match. To match the other 24 bytes, we can simply choose the remaining 24 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}.

1st Part of the Merge Inbound Phase. To merge the inbound phases, we first compute 2^{128} pairs for the yellow/purple part with a total complexity of 2^{128} and store these pairs in a list L_2. We also compute 2^{128} pairs for the green/blue/red part. Since the complexity to compute one solution for this part is 2^{32}, the total complexity to compute all 2^{128} pairs is 2^{160}. To connect the resulting pairs between state S_{30} and S_{32} we need to satisfy two 128-bit conditions. First of all, we need to satisfy the linear 128-bit SuperMixColumns condition observed by Jean et al. in [3]. 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 total, this gives a 256-bit condition which we can satisfy using the birthday effect and get $2^{128} \times 2^{128} \times 2^{-256} = 1$ pair which satisfies the whole 6-round truncated differential path. The time complexity is 2^{160} and with a memory complexity if 2^{128}.

2nd Part of the Merge Inbound Phase. In the second part of the merge inbound phase, we need to find values for the remaining white bytes again. This part of the attack is again the same as in the improved hash function attack on ECHO-256 (see Section 3). The only difference is that we change the time-memory trade-off slightly to get an average complexity of one 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} (also compare with Fig. 2). Since we have a 192-bit condition in state S_{23}, we get $2^{3 \cdot 96} \times 2^{-192} = 2^{96}$ solutions with a complexity of 2^{96} in time and memory, or with an average complexity of 1 per solution (see [10] for more details). It follows, that we can find one pair, as well as 2^{160} pairs for the 6-round truncated differential path with a total complexity of 2^{160} and memory requirements of 2^{128} with chosen salt.

4.2 Collisions for 6 Rounds

To get a collision for 6 rounds of the 512-bit compression function of ECHO-256 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^{128} pairs for the truncated differential path with a complexity of 2^{160}, we can get a collision at the output of the compression function after 6 rounds with a complexity of 2^{160} and memory requirements of 2^{128} with chosen salt.

4.3 Subspace Distinguisher for 7 Rounds

To get a distinguisher for 7 rounds of the compression function of ECHO-256, we use the truncated differential path given in Fig. 3. Note that the truncated differential path in the last round is followed 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 3rd 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 of 1, we can find one pair which also satisfies the conditions in the last round with a complexity of 2^{128} in time and memory. 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 [4, 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. Hence, with chosen salt we get a 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 the 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, this time we 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 and with chosen salt.

5 Conclusion

In this work, we have shown how to efficiently solve the condition observed by Jean et al. in [3] for the hash function of ECHO as well. This condition occurs due to the low rank of the SuperMixColumns transformation in ECHO. However, using the large degrees of freedom and a meet-in-the-middle approach, this 128-bit condition can be solved with a complexity much lower than 2^{128}. On the other hand, the low rank of the SuperMixColumns transformation allows us to extend the collision attack on ECHO-256 by one more round. We have shown that collision for 5/8 rounds of the ECHO-256 hash function can be constructed with a complexity of 2^{112} and memory requirements of $2^{85.3}$. Furthermore, we have corrected the subspace distinguisher for 7 out of 8 rounds on the compression function of ECHO-256 and get a complexity of 2^{160} with memory requirements of 2^{128} with chosen salt. We expect that the attacks on ECHO can be further improved by combining our ideas with those of Jean et al.

Acknowledgements

We would like to thank Jérémy Jean and María Naya-Plasencia for their comments and useful discussions. The work described in this paper has been supported by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

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. Jérémy Jean and Pierre-Alain Fouque. Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function. Cryptology ePrint Archive, Report 2010/569, 2010. `http://eprint.iacr.org/`.
4. 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.
5. 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.
6. 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.
7. 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 online: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
8. Martin Schläffer. Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. In *Selected Areas in Cryptography*, 2010. to appear.
9. Martin Schläffer. Subspace distinguisher for 5/8 rounds of the echo-256 hash function. Cryptology ePrint Archive, Report 2010/321, 2010. `http://eprint.iacr.org/`.
10. David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002. Extended version available online at `http://www.eecs.berkeley.edu/~daw/papers/genbday.html`.