Analysis of Reduced-SHAvite-3-256 v2*

Marine Minier[1], María Naya-Plasencia[2], and Thomas Peyrin[3]

[1] Université de Lyon, INRIA, CITI, F-69621, France
[2] FHNW, Windisch, Switzerland
[3] Nanyang Technological University, Singapore

Abstract. In this article, we provide the first independent analysis of the (2^{nd}-round tweaked) 256-bit version of the `SHA-3` candidate `SHAvite-3`. By leveraging recently introduced cryptanalysis tools such as rebound attack or Super-Sbox cryptanalysis, we are able to derive chosen-related-salt distinguishing attacks on the compression function on up to 8 rounds (12 rounds in total) and free-start collisions on up to 7 rounds. In particular, our best results are obtained by carefully controlling the differences in the key schedule of the internal cipher. Most of our results have been implemented and verified experimentally.

Keywords: rebound attack, Super-Sbox, distinguisher, `SHAvite-3`, `SHA-3`.

1 Introduction

In cryptography hash functions are one of the most important and useful tools. An n-bit cryptographic hash function H is a function taking an arbitrarily long message as input and outputting a fixed-length hash value of size n bits. One wants such a primitive to be collision resistant and (second)-preimage resistant: it should be impossible for an attacker to obtain a collision (two different messages hashing to the same value) or a (second)-preimage (a message hashing to a given challenge) in less than $2^{n/2}$ and 2^n computations respectively. However, in many protocols hash functions are used to simulate the behavior of a random oracle [1] and the underlying security proof often requires the hash function H to be indistinguishable from a random oracle. This security notion naturally extends to a fixed-input length random oracle with a compression function.

In recent years, we saw the apparition of devastating attacks [23,22] that broke many standardized hash functions [20,14]. The National Institute of Standards and Technology (NIST) launched the `SHA-3` competition [16] in response

* This work was partially supported by the French National Agency of Research: ANR-06-SETI-013. The second author is supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322. The third author is supported by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03.

A. Joux (Ed.): FSE 2011, LNCS 6733, pp. 68–87, 2011.

to these attacks and in order to keep an appropriate security margin considering the increase of the computation power or potential further cryptanalysis improvements. The outcome of this competition will be a new hash function standard, to be selected in 2012. Among the 64 candidates originally submitted and the 14 selected for the 2nd round of the competition, one can observe that a non negligible proportion are AES-based proposals (reuse of some parts of the AES block cipher [15,5] or mimicry of its structure), as SHAvite-3 [3] for example.

This fact motivated the academic community to improve its knowledge concerning the security of the AES block cipher or AES-like permutations in the setting of hash functions [17,8,13,11,10,12,7]. For an attacker, one of the major distinction between cryptanalyzing a block cipher and a hash function is that in the latter he has full access to the internal computation and thus he can optimize its use of the freedom degrees. In particular, the recent SHA-1 attacks where made possible thanks to an improvement of the use of the freedom degrees. In the case of AES-based hash functions, the rebound attack [13,10], Start-from-the-middle [12] or Super-Sbox cryptanalysis [7] are very handy tools for a cryptanalyst.

During the first round of the SHA-3 competition, SHAvite-3 was first analyzed by Peyrin [18] who showed that an attacker could easily find chosen-counter chosen-salt collisions for the compression function. This weakness led to a tweaked version of the algorithm for the second round. Then, recently new cryptanalysis results [4,6] on the 512-bit version of SHAvite-3 were published, in particular a chosen-counter chosen-salt preimage attack on the full compression function of SHAvite-3-512.

Our contributions. In this paper, we give the first cryptanalysis results on the tweaked 256-bit version of SHAvite-3-256. By using the rebound attack or Super-Sbox cryptanalysis, we are able to derive distinguishers for a reduced number of rounds of the internal permutation of SHAvite-3-256. Those results can then be transformed into distinguishers or can be used to mount a free-start collision attack for reduced versions of the compression function. The number of rounds attacked can be further extended by authorizing the attacker to fully control the salt values. The results are summarized in Table 1. We emphasize that most of the attacks have been implemented and verified experimentally.

Table 1. Summary of results for the SHAvite-3-256 compression function

rounds	comput. complexity	memory complexity	type	section
6	2^{80}	2^{32}	free-start collision	sec. 3
7	2^{48}	2^{32}	distinguisher	sec. 3
7	2^{7}	2^{7}	chosen-related-salt distinguisher	sec. 4.1
7	2^{25}	2^{14}	chosen-related-salt free-start near-collision	sec. 4.2
7	2^{96}	2^{32}	chosen-related-salt semi-free-start collision	ext. vers.
8	2^{25}	2^{14}	chosen-related-salt distinguisher	sec. 4.2

2 The SHAvite-3-256 Hash Function

SHAvite-3-256 is the 256-bit version of SHAvite-3 [3], an iterated hash function based on the HAIFA framework [2]. We describe here the tweaked version of the algorithm. The message M to hash is first padded and then split into ℓ 512-bit message blocks $M_0 \| M_1 \| \ldots \| M_{\ell-1}$. Then, the 256-bit internal state (initialized with an initial value IV) is iteratively updated with each message block using the 256-bit compression function C_{256}. Finally, when all the padded message blocks have been processed, the output hash is obtained by truncating the internal state to the desired hash size n as follows:

$$h_0 = IV, \;\; h_i = C_{256}(h_{i-1}, M_{i-1}, salt, cnt), \;\; hash = trunc_n(h_i)$$

Internally, the 256-bit compression function C_{256} of SHAvite-3-256 consists of a 256-bit block cipher E^{256} used in classical Davies-Meyer mode. The input of the compression function C_{256} consists of a 256-bit chaining value h_{i-1}, a 512-bit message block M_{i-1}, a 256-bit salt (denoted $salt$) and a 64-bit counter (denoted cnt) that represents the number of message bits processed by the end of the iteration. The output of the compression function C_{256} is given by:

$$h_i = C_{256}(h_{i-1}, M_{i-1}, salt, cnt) = h_{i-1} \oplus E^{256}_{M_{i-1} \| salt \| cnt}(h_{i-1})$$

where $\oplus$ denotes the XOR function.

2.1 The Block Cipher E^{256}

The internal block cipher E^{256} of the SHAvite-3-256 compression function is composed of 12 rounds of a classical 2-branch Feistel structure. The chaining variable input h_{i-1} is first divided into two 128-bit chaining values (A_0, B_0). Then, for each round, the Feistel construction computes:

$$(A_{i+1}, B_{i+1}) = (B_i, A_i \oplus F_i(B_i)), \;\; i = 0, \ldots, 11$$

where F_i is a non-linear function composed of three full AES rounds. More precisely, if one considers that $AESr$ denotes the unkeyed AES round (i.e. SubBytes SB, ShiftRows ShR and MixColumns MC functions in this order), then F_i is defined by (see Figure 1) :

$$F_i(x) = AESr(AESr(AESr(x \oplus k_i^0) \oplus k_i^1) \oplus k_i^2) \tag{1}$$

where k_i^0, k_i^1 and k_i^2 are 128-bit local keys generated by the message expansion of the compression function C^{256} (it can also be viewed as the key schedule of the internal block cipher E^{256}). We denote by $RK_i = (k_i^0, k_i^1, k_i^2)$ the group of local keys used during round i of the block cipher.

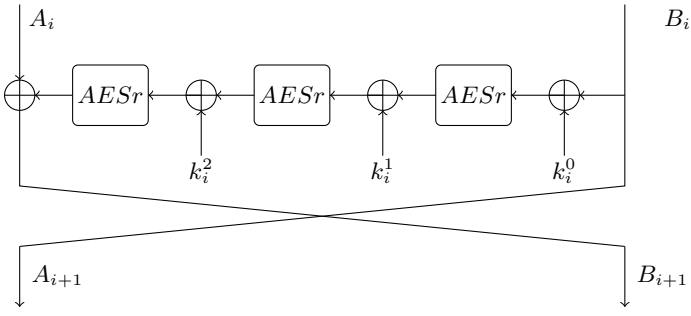

Fig. 1. Round i of the state update of SHAvite-3-256 compression function

2.2 The Message Expansion

The message expansion of C_{256} (the key schedule of E^{256}) takes a 512-bit message block M_i, the 256-bit salt ($salt$) and the 64-bit counter (cnt) as inputs. The 512-bit message block M_i is represented as an array of sixteen 32-bit words $(m_0, m_1, \ldots, m_{15})$, the 256-bit salt as an array of eight 32-bit words $(s_0, s_1, \ldots, s_7)$ and the counter as an array of two 32-bit words (cnt_0, cnt_1). 36 128-bit AES local subkeys k_i^j (with $0 \leq i \leq 11$ and $0 \leq j \leq 2$) are generated, seen as 144 words of 32 bits each (one word standing for one AES column), represented in an array $rk[0...143]$:

$$\begin{aligned}(k_i^0, k_i^1, k_i^2) = &(rk[12 \cdot i], rk[12 \cdot i + 1], rk[12 \cdot i + 2], rk[12 \cdot i + 3]),\\ &(rk[12 \cdot i + 4], rk[12 \cdot i + 5], rk[12 \cdot i + 6], rk[12 \cdot i + 7]),\\ &(rk[12 \cdot i + 8], rk[12 \cdot i + 9], rk[12 \cdot i + 10], rk[12 \cdot i + 11])\end{aligned}$$

The first 16 values of the array rk are initialized with the message block m_i, i.e. $rk[i] = m_i$ with $0 \leq i \leq 15$. Then, the rest of the array is filled by repeating four times the step described in Figure 2. During one step, sixteen 32-bit words are first generated using parallel AES rounds and a subsequent linear expansion step L_1 (the salt words are XORed to the internal state before applying the AES rounds). Note that during each step the two counter words cnt_0 and cnt_1 (or a complement version of them) are XORed with two particular 32-bits words at the output of the AES rounds. Then, sixteen more 32-bit words are computed using only another linear layer L_2. For more details on the message expansion, we refer to the submission document of the tweaked version of SHAvite-3 [3].

3 Rebound and Super-Sbox Analysis of SHAvite-3-256

Before describing our distinguishing and free-start collision attacks on reduced versions of the SHAvite-3-256 compression function, we first explain what are the main tools we are going to use.

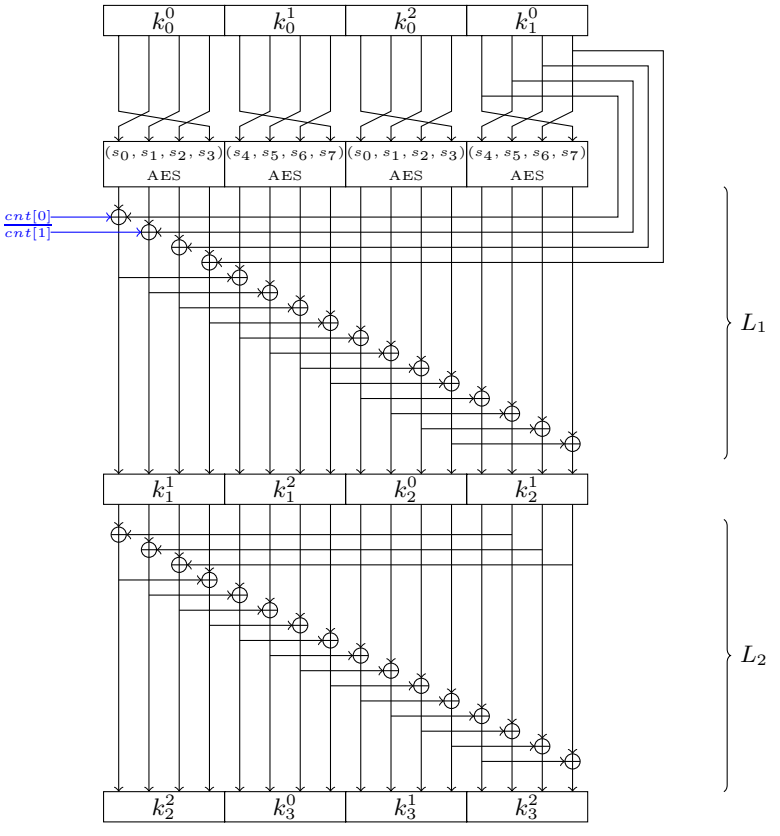

Fig. 2. The first step of the message expansion of the SHAvite-3-256 compression function. The salt words are XORed to the internal state before the parallel AES rounds application. The counters are XORed several times at different positions.

3.1 The Cryptanalyst Tool 1: The Truncated Differential Path

When cryptanalyzing AES-based hash functions (or more generally byte-oriented primitives), it has been shown [17] that it is very handy to look at truncated differences [9]: instead of looking at the actual difference value of a byte, one only checks if a byte contains a difference (active byte) or not (inactive byte). In addition to simplifying the analysis, the direct effect is that the differential behavior through the non-linear Sboxes becomes deterministic. On the other hand, the differential transitions through the linear MixColumns layer will be verified probabilistically.

More precisely, the matrix multiplication underlying the AES MixColumns transformation has the interesting property of being a Maximum Distance Separable (MDS) mapping: the number of active input and output bytes for one column is always greater or equal to 5 (unless there is no active input and output byte at all). When picking random input values, the probability of success for

a differential transition that meets the MDS constraints through a MixColumns layer is determined by the number of active bytes in the output: if such a differential transition contains k active bytes in one column of the output, its probability of success will approximatively be equal to $2^{-8\times(4-k)}$. For example, a $4 \mapsto 1$ transition for one column has success probability of approximatively 2^{-24}. Note that the same reasoning applies when dealing with the invert function as well.

In the following, we will use several different types of truncated differential masks for a given 128-bit AES state. We reuse the idea from [19] and we restrict ourselves to four types of byte-wise truncated differential words F, C, D and 1, respectively a fully active state, one fully active column only, one fully active diagonal only and one active byte only. Considering those 4 types of differential masks seems natural because of the symmetry and diffusion properties of an AES round.

We are especially interested in the truncated differential transitions through 3 rounds of the AES since it is the main basic primitive used in the round function of the SHAvite-3-256 compression function. We would like to know what is the probability to go from one truncated differential mask to another (both forward and backward) and the corresponding differential path. First, we can compute the approximate probability of success for a one-round transition between the four types of truncated differential states for both forward and backward directions. Those probabilities are simply obtained by studying the MixColumns transitions for one AES round. For example, one can easily check that when computing forward, going from D to F with the trail $\mathtt{D} \mapsto \mathtt{1} \mapsto \mathtt{C} \mapsto \mathtt{F}$ happens with probability 2^{-24} with randomly selected input values and active bytes difference values. The same probability holds for the inverse trail in the backward direction.

3.2 The Cryptanalyst Tool 2: The Freedom Degrees

The second very important tool for a hash function cryptanalyst are the freedom degrees. The rebound attack [13] uses a local meet-in-the-middle-like technique in which the freedom degrees are consumed in the middle part of the differential path, right where they can improve at best the overall complexity. More precisely, the rounds in the middle are controlled (the controlled rounds) and will be verified with only a few operations on average, while the rest of the path both in forward and backward direction is fulfilled probabilistically (the uncontrolled rounds). This method provides good results [11,10], but the controlled part is limited to two rounds only. In [12], this technique is generalized to start-from-the-middle attacks, allowing to control 3 rounds in the middle part, without increasing the complexity (i.e. only a few operations on average). However, this technique is more complex to handle and only works for differential paths for which the middle part does not contain too many active bytes. Finally, the Super-Sbox cryptanalysis (independently introduced in [10] and [7]) can also control 3 rounds in the middle of the differential trail with only a few operations on average and works for any differential path. The idea is that one can view two rounds of an AES-like permutation as the parallel application of a layer of big Sboxes, named Super-Sboxes, preceded and followed by simple affine transformations.

This technique can find several solutions for an average cost of 1, but there is a minimal cost to pay in any case: the complexity of the attack in the case of the AES permutation is $\max\{2^{32}, k\}$ computations and 2^{32} memory, where k is the number of solutions found verifying the controlled rounds.

3.3 Super-Sbox Attacks for Reduced SHAvite-3-256

Having introduced our cryptanalyst tools, we will derive distinguishing attacks for the 7-round reduced SHAvite-3-256 compression function, or even free-start collision attacks (collision for which the incoming chaining variable is fully controlled by the attacker) for the 6-round reduced version. We start with the 6-round truncated differential path built by removing the last round of the 7-round differential path depicted in Figure 3. First, one can check that this path is valid as it contains no impossible MixColumns transitions. Moreover, a simple analysis of the amount of freedom degrees (as it is done in [7]) shows that we have largely enough of them in order to obtain at least one valid solution for the whole differential path: we have a probability of about 2^{-48} that a valid pair for the differential path exists when the Δ and the subkeys values are fixed. Randomizing those values provides much more than the 2^{48} freedom degrees required.

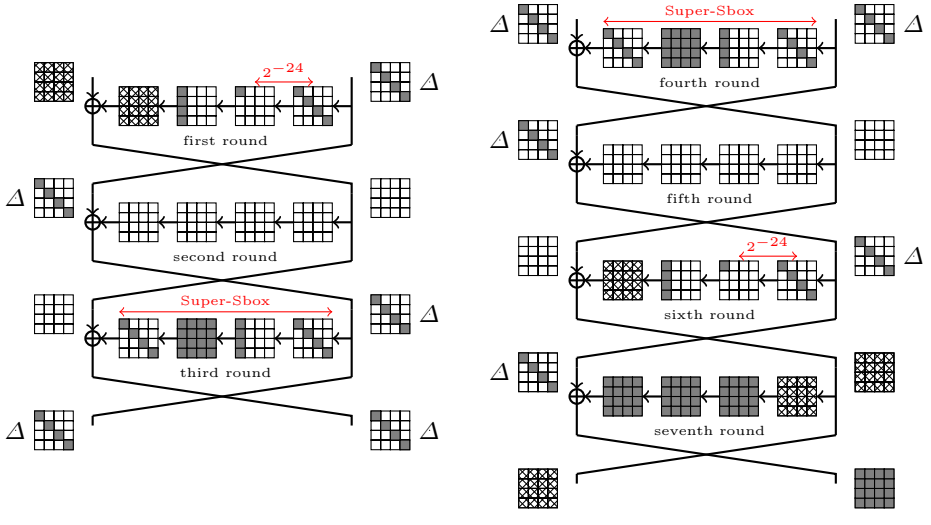

Fig. 3. The 7-round truncated differential path. The left part is the three first rounds and the right part the four last ones. Each gray cell stands for an active byte. A hatched state denotes a fully active state obtained by applying the MixColumns function on only one active byte per column. All D 128-bit words contain the difference Δ.

The most costly part is obviously located in the middle, during the third and fourth rounds where we have fully active AES states (F-type). Thus, we will use the available freedom degrees at those particular rounds precisely. Let Δ be one

of the 2^{32} possible D-type difference values. With the Super-Sbox technique and by using the freedom degrees available on the message input, we will find a valid 128-bit pair for the fourth round, mapping the difference Δ to the very same difference through the 3 AES rounds. Then, we will do the same for the third round.

The subkeys used during the fourth round are k_3^0, k_3^1, k_3^2. We first choose a random value for k_3^2. Then, using the Super-Sbox technique, for a cost of $\max\{2^{32}, 2^{32}\} = 2^{32}$ computations and 2^{32} memory, one can generate 2^{32} pairs of 128-bit states verifying the truncated differential path for the 3 AES rounds in the right branch of the fourth round: $\mathtt{D} \mapsto \mathtt{C} \mapsto \mathtt{F} \mapsto \mathtt{D}$. At the present time, we did not fix k_3^0 nor k_3^1, because we were only looking at truncated differences: for each 128-bit solution pair found, the 3 AES rounds truncated differential path will be verified whatever is the value of k_3^0 or k_3^1 (more precisely k_3^1 will only have an incidence on the exact D-type difference value on the input of the pairs, while k_3^0 will have no incidence on the difference at all). Note that the Super-Sbox technique allows us to directly force the exact difference value Δ at the output of the 3 AES rounds. Indeed, one can observe that the output of the 3 AES rounds is only a linear combination of the 4 Super-Sboxes outputs. However, the exact difference value on the four active bytes at the input of the 3 AES rounds is unknown because of the SubBytes layer of the first AES round. In order to get the desired difference value Δ on the input as well, for each solution pair we choose accordingly the value of k_3^1. More precisely, only the first column of k_3^1 must be accommodated (only the first column is active when incorporating k_3^1) and this can be done byte per byte independently. Exhausting all the AES Sbox differential transitions during a 2^{16} operations precomputation phase allows us to perform this step with only four table lookups. At this moment, for a cost of 2^{32} computations and memory, we found 2^{32} pairs of 128-bit states that map Δ to Δ through the 3 AES rounds of the fourth SHAvite-3-256 round. For each pair, the value of k_3^2 and the first column of k_3^1 are fixed, but the rest of the message is free to choose.

We perform exactly the same method in order to find 2^{32} pairs of 128-bit states that map Δ to Δ through the 3 AES rounds of the third SHAvite-3-256 round. The only difference is that we will have to fix k_2^2 and choose the first column of k_2^1. This can be done independently from the previously fixed values of k_3^2 and the first column of k_3^1 (by setting the second column of k_3^1, see Figure 2). We are left with two sets, each of 2^{32} pairs, verifying **independently** the third and fourth rounds. The subkey material that remains free is set to a random value and the second and fifth rounds of the differential path is verified with probability 1.

Then, the rest of the path (the uncontrolled rounds) is verified probabilistically: we have one $\mathtt{D} \mapsto \mathtt{1} \mapsto \mathtt{C} \mapsto \mathtt{F}$ transition in the first round and another one in the sixth round. As already demonstrated, this happens with probability $2^{-2*24} = 2^{-48}$. Overall, one can find a valid candidate for the whole 6-round truncated differential path with 2^{48} computations and 2^{32} memory.

If the very last branch switching of the Feistel structure is removed (at the end of the sixth round) and due to the Davies-Meyer construction, one can obtain a free-start collision if the active byte differences on the input of the first round are equal to the active byte differences on the output of the sixth round. This happens with probability 2^{-32} because we have the right 128-bit word of the internal state containing the difference Δ on both the input and output. The left one is fully active but its differences belong to a 4-byte subspace since only one MixColumns linear application away from a 4-byte difference (hatched states in Figure 3). Finally, by repeating the process, one can find a free-start collision for 6-round reduced `SHAvite-3-256` with $2^{48+32} = 2^{80}$ computations and 2^{32} memory.

We can now move to the full 7-round path depicted in Figure 3. One solution for the entire path can still be obtained with 2^{48} computations and 2^{32} memory since the differential trail in the last round is verified with probability 1. A solution pair will have a difference Δ on its right word input and a difference maintained in a subspace of roughly 2^{32} elements on its left word input (as denoted with hatched cells in Figure 3). Concerning the output, the differences on the left 128-bit output word is kept in the same subspace of 2^{32} elements, while the right output word will have a random difference. Overall, after application of the feed-forward, we obtain a compression function output difference maintained in a subspace of at most 2^{160} elements, while the input difference is maintained in a subspace of 2^{32} elements (since the message/salt/counter inputs contain no difference and the value of Δ is fixed). This is equivalent to mapping a fixed 672-bit input difference (224 bits for the chaining variable, 256 bits for the non active incoming message chunk, 128 and 64 bits for the non active incoming salt and counter chunks respectively) to a fixed 96-bit output difference through a 704-bit to 256-bit compression function. According to the limited-birthday distinguishers [7], this should require 2^{64} computations in the ideal case.[1] Thus, one can distinguish a 7-round reduced version of the SHAvite-3-256 compression function with 2^{48} computations and 2^{32} memory.

4 Chosen-Related-Salt Distinguishers

While the previous section takes a full advantage of the Super-Sbox techniques in its analysis, this section presents an attack that uses rebound attack principle fully exploiting the message expansion. This leads to 7-round and 8-round chosen-related-salt distinguishers on the `SHAvite-3-256` compression function with a complexity of 2^7 and 2^{25} operations respectively. Also, one can find chosen-related-salt semi-free-start collisions on 7 rounds of the `SHAvite-3-256` compression function with a complexity of 2^{96} operations (see extended version of this article). In this section, we will insert differences in the message and in

[1] As shown in [7], if we denote by i (resp. j) the number of fixed-difference bits in the input (resp. in the output) and by t the total number of input bits of the compression function, the equivalent complexity to find such a structure for a random compression function is $2^{i+j-t} = 2^{672+96-704} = 2^{64}$ computations.

the salt input of the compression function. The main principle of those analyses relies on correcting the differences at the end of each round so that they are not spread afterwards. The 8-round differential path used is given in Figure 4, while the 7-round version is obtained by removing the seventh round. All the differences in the successive internal states will be canceled from state 1 to state 7. This could be done by considering differences in the first four 32-bit words of the *salt* denoted (s_0, s_1, s_2, s_3), in eight 32-bit message words (m_0, m_1, m_2, m_3) and $(m_8, m_9, m_{10}, m_{11})$ and differences in A_0, the left part of the initial chaining value, the other parameters being taken without any difference. The notations used are the ones of Section 2: A_i denotes the left part of the state at round i (where i goes from 0 to 8), and B_i the right part of the internal state at round i.

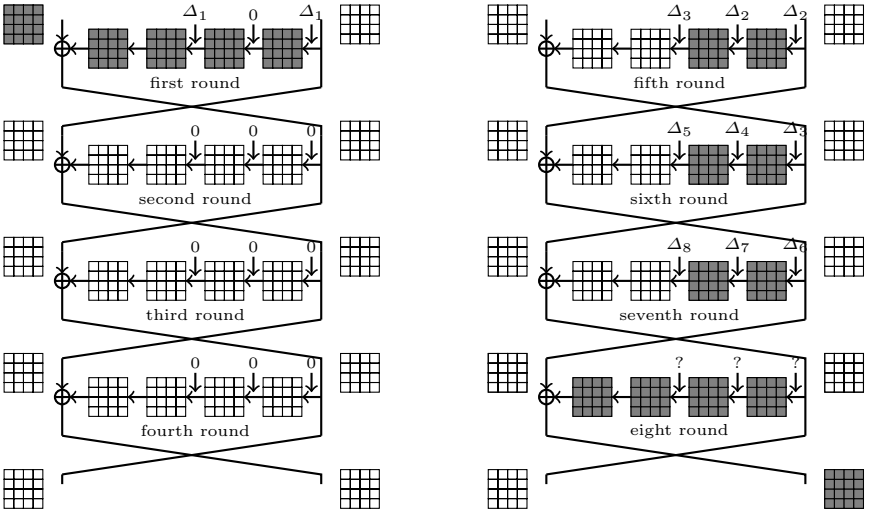

Fig. 4. The 8-round truncated differential path. The left part represents the four first rounds and the right part the four last ones. Each gray cell stands for an active byte. The Δ's in each round denote the differences incorporated by the subkeys. For the 7-round differential path, we remove the differences control in the seventh round.

The difference values in the salt words and in the message words are chosen to be identical (Δ_1), so that they cancel for the subkeys generated after the first round of the message expansion. Moreover, the subkeys involved in rounds 2, 3 and 4 will not contain any difference (as shown on Figure 5). We concentrated our analysis on the active rounds in the middle of the trail, i.e. rounds 5, 6 (and 7 in the case of the 8-round distinguisher). The probability of success for the rest of the differential path is one since in the first round the differences can spread freely.

We start by finding a valid pair that verifies the path for the rounds 5, 6 and 7. Let us remember that at the beginning, we have just established the truncated differential path, i.e. the actual difference values in the bytes are unknown. Thus,

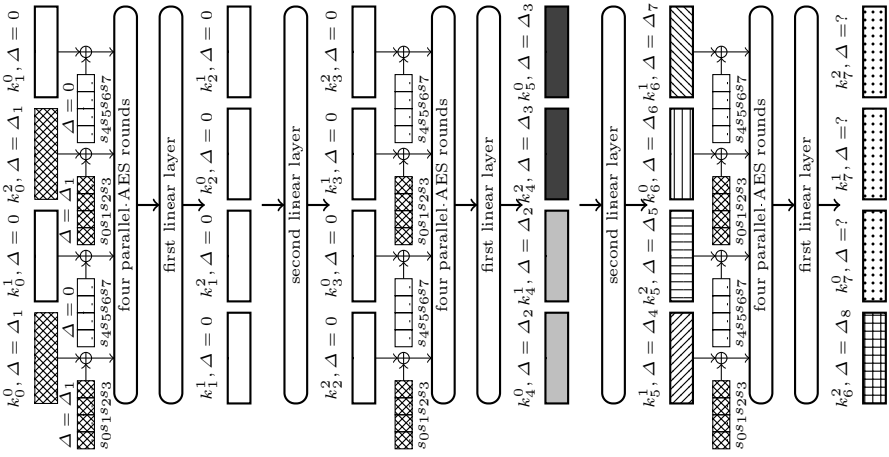

Fig. 5. Differential cancellation and differential equalities in the message expansion for the 7-round and 8-round chosen-related-salt distinguishers on the SHAvite-3-256 compression function. $\tilde{\Delta}$ represents the difference Δ after application of the column switching layer just before the salt incorporation.

when required, we will fix the difference values of the active bytes and also the values themselves. We will insert differences in the first four salt words (s_0, s_1, s_2, s_3), all their bytes being active. As before, when we refer to words, we denote the AES column 32-bit words in the message expansion.

For a better understanding of the distinguisher procedure, we discriminate 5 distinct kinds of ways in order to determine values and differences:

- Type a means that we directly choose the values and the differences. This can be done when there is no previous restriction.
- Type b are determined by the linear part of the message expansion. In this case, a linear relation of previously fixed differences or values completely determines the remaining ones.
- Type c are determined by the non-linear part of the message expansion. Here, a non linear relation of previously fixed differences and values determines the remaining ones.
- Type d are produced by some previous conditions on the Feistel path. That is, for example, if the value of $B_i \oplus k^0_{i+1}$ is fixed and then this subkey is determined, we automatically deduce the value of B_i from this equation. This will directly determine A_{i+1} since $A_{i+1} = B_i$.
- Type e are fixed by the AES rounds. This basically represents the conditions associated to the controlled rounds part.

From now on and for a better clarity, we indicate its type in brackets for each determination. When omitted, the default type is a. We will first describe the distinguisher on 7 rounds and then the one on 8 rounds.

4.1 7-Round Distinguisher with 2^7 Computations

As partially shown on Figure 4, the aim of the 7-round distinguisher is to find a pair of plaintext/ciphertext values for the internal block cipher of SHAvite-3-256 such that there is no difference on the right part of the plaintext and on the left part of the ciphertext. As shown in [7], the corresponding complexity to find such a structure for a random permutation is equal to 2^{64} computations. We show in this section how to find a pair of inputs that verifies the path with a time and memory complexity of 2^7 (by omitting the last branch swapping of the Feistel construction, the attack also applies to the compression function SHAvite-3-256 with the same practical and ideal complexities). In order to build this distinguisher, we would like to find values and differences of the subkeys such that the difference in the 3 AES rounds during SHAvite-3-256 rounds 5 and 6 are auto-erased. The differences generated by the subkeys in the seventh round are let completely free for the 7-round distinguisher and it will not be the case for the 8-round distinguisher. We first give in Table 2 how to fix the degrees of freedom in order to avoid any impossibility.

Table 2. Order and conditions for fixing values and differences. Δ_2 and Δ_3 are chosen such that Δ_2 and $\Delta_2 \oplus \Delta_3$ can be both generated from an AES layer with the same difference in the inputs (for randomly chosen Δ_2 and Δ_3, this is verified with very high probability). When going through all these steps followed in the given order, we are left with the degrees of freedom associated to the values of the words of the salt s_0, s_1, s_2, s_3. Thus, we can pick a random value for those remaining freedom degrees and finally compute the valid pair for the 7-round differential path. * represents the steps that are not executed for the distinguisher over 8 rounds.

instant	fixed (type a)	implies	type	cost
begin	$\Delta k_4^0 = \Delta_2$ $\Delta k_4^2 = \Delta_3$	$\Delta k_4^1 = \Delta_2$ $\Delta k_3^0 = \Delta_3$ $\Delta k_5^1 = \Delta_2 \oplus ((\Delta_3 \wedge (2^{96}-1)) \| \| ((\Delta_3 >> 96) \oplus \Delta_2) \wedge (2^{32}-1))$ $\Delta k_5^2 = \Delta_2 \oplus ((\Delta k_5^1 \wedge (2^{96}-1)) \| \| ((\Delta k_5^1 >> 96) \oplus \Delta_2) \wedge (2^{32}-1))$ $\Delta k_6^0 = \Delta_3 \oplus ((\Delta k_5^2 \wedge (2^{96}-1)) \| \| ((\Delta k_5^2 >> 96) \oplus \Delta_3) \wedge (2^{32}-1))$ $\Delta k_6^1 = \Delta_3 \oplus ((\Delta k_6^0 \wedge (2^{96}-1)) \| \| ((\Delta k_6^0 >> 96) \oplus \Delta_3) \wedge (2^{32}-1))$	b b b b b b	1 1 1 1 1 1
round 5	$B_4 \oplus k_4^0$	$k_4^1 : AES_r(AES_r(B_4 \oplus k_4^0) \oplus k_4^1) \oplus AES_r(AES_r(B_4 \oplus k_4^0 \oplus \Delta_2) \oplus k_4^1 \oplus \Delta_2) = \Delta_3$	e	2^6
round 6	$B_5 \oplus k_5^0$	$k_5^1 : AES_r(AES_r(B_5 \oplus k_5^0) \oplus k_5^1) \oplus AES_r(AES_r(B_5 \oplus k_5^0 \oplus \Delta_3) \oplus k_5^1 \oplus \Delta k_5^1) = \Delta k_5^2$ $k_5^2 = k_4^1 \oplus ((k_5^1) \wedge (2^{96}-1)) \| \| ((k_5^1 >> 96) \oplus k_4^1) \wedge (2^{32}-1)$ $k_4^0 \wedge (2^{32}-1) = (k_5^1 \wedge (2^{32}-1)) \oplus (((k_5^1) >> 96) \wedge (2^{32}-1))$ $B_4 \wedge (2^{32}-1) = (k_4^0 \wedge (2^{32}-1)) \oplus (k_4^0 \oplus B_4) \wedge (2^{32}-1)$	e b b b	2^6 1 1 1
end	$\delta s_0 \ldots \delta s_3 = \Delta_1$	$k_4^2 : AES_r^{-1}(k_4^2 \oplus k_4^1) \oplus AES_r^{-1}(k_4^2 \oplus k_4^1 \oplus \Delta_2 \oplus \Delta_3) = \Delta_1$ $k_6^0 = k_4^2 \oplus ((k_5^2 \wedge (2^{96}-1)) \| \| ((k_5^2 >> 96) \oplus k_4^2) \wedge (2^{32}-1))$	c b	1 1
*	k_5^0	$(k_4^0 >> 32) \wedge (2^{96}-1) = k_5^0 \wedge (2^{96}-1)$ $k_6^1 = k_5^0 \oplus (k_6^0 \wedge (2^{96}-1)) \| \| ((k_6^0 >> 96) \oplus k_5^0) \wedge (2^{32}-1)$ $s_4 \ldots s_7 : AES_r^{-1}(k_4^0 \oplus cnt \oplus AES_r^{-1}(k_5^0 \oplus k_4^2) \oplus s_4 \| \| s_5 \| \| s_6 \| \| s_7) \oplus AES_r^{-1}(k_4^0 \oplus ct \oplus AES_r^{-1}(k_5^0 \oplus k_4^2) \oplus s_4 \| \| s_5 \| \| s_6 \| \| s_7 \oplus \Delta_2) = \Delta_1$ $B_4 = k_4^0 \oplus (k_4^0 \oplus B_4)$ $B_5 = k_5^0 \oplus (B_5 \oplus k_5^0)$	b b c d d	1 1 1 1 1

Fifth round: As described in Figure 4, after the fourth round, the inputs of the fifth do not contain any active byte. The differences will be injected during the fifth round through the message expansion. The idea is to erase those differences directly during the fifth round. This will be achieved by carefully choosing the actual values of the bytes.

We choose (type a) the difference in k_4^0 to be Δ_2 and this sets (type b) the difference in k_4^1 to be Δ_2 as well. Then we pick a random difference value for k_4^2 that we denote Δ_3. Note that due to the message expansion, Δ_2 and $\Delta_3 \oplus \Delta_2$ must be 128-bit output differences that can be obtained from the same input difference after application of one `AES` round. Indeed, as shown in Figure 5, Δ_2 and $\Delta_3 \oplus \Delta_2$ are constructed from the Δ_1 difference inserted by the salt after application of one `AES` round. In fact, for randomly chosen Δ_2 and Δ_3 values, this is verified with very high probability. Those particular differences also fix (type b) the difference values of the 9 subkeys used in rounds 5, 6 and 7. However, note that k_6^2 is determined after one another `AES` non-linear round (as shown in Figure 5).

Once Δ_2 and Δ_3 differences fixed, we need to set the values themselves in this fifth round. To do so, we choose random value for $B_4 \oplus k_4^0$ (see Figure 6) and we can compute forward the differences just before the SubBytes layer of the second `AES` round. We can also propagate the differences backwards from the insertion of Δ_3 up to the output of this SubBytes layer. Due to the `AES` Sbox differential property, with a probability of 2^{-16}, the differences before and after the second SubBytes can be matched (2^{-1} per Sbox). Thus, we will have to try 2^{16} values of $B_4 \oplus k_4^0$ before finding a match. Note that this cost can be reduced to $4 \times 2^4 = 2^6$ by attacking all the columns independently. When such a solution is found, we directly pick (type e) a value of k_4^1 that makes this match happen.

Sixth round: We deal with the sixth round in a very same way. The differences in the subkeys k_5^0, k_5^1 and k_5^2 are already fixed by the message expansion. Thus, we would like to find the corresponding values that make the cancellation of differences possible in the sixth round computation. As before, we choose an appropriate value of $B_5 \oplus k_5^0$ such that we have a possible match between the input and output differences of the second SubBytes layer. When such a solution is found, we directly pick (type e) a value of k_5^1 that makes this match happen.

The final step: Now, if we randomly fix the values and differences δs_0, δs_1, δs_2, δs_3, and k_5^0, the values of k_4^2, k_6^0, k_4^0, k_6^1, s_4, s_5, s_6, s_7, B_4 and B_5 will also be determined (as shown in Table 2). At this point, we have obtained some coherent values that verify the differential path of rounds 5 and 6, and the only degrees of freedom left are the values of s_0, s_1, s_2 and s_3. We can just pick up one and compute backward and forward, and we obtain the whole path, where the inputs have just the left part of the state active, and the output, before the Davies-Meyer, the right one.

The total cost for the distinguisher is driven by the two first steps, that is $2 \times 2^6 = 2^7$ operations in order to find one valid candidate. This distinguisher has been implemented and verified experimentally. We provide in the extended version of this article an example of such a structured input/output pair (which should not be generated with less than 2^{64} operations in the ideal case).

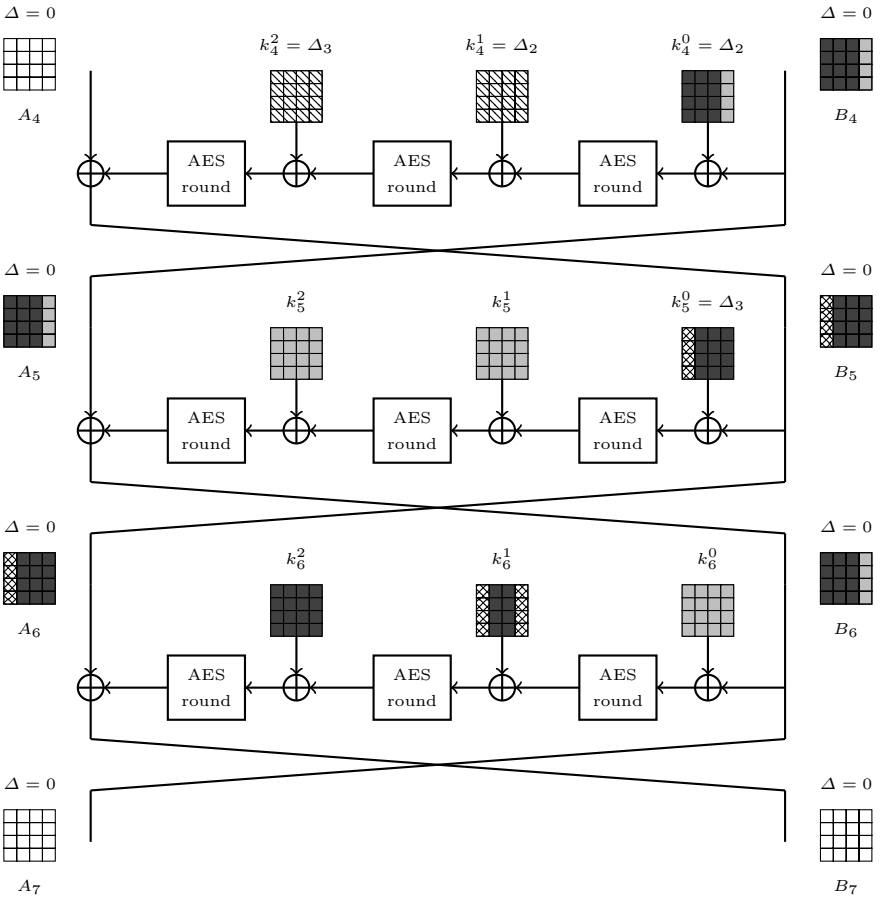

Fig. 6. Details of rounds 5, 6 and 7 of the 8-round chosen-related-salt distinguisher on the `SHAvite-3-256` compression function. The bytes denoted with north-west lines are fixed during the fifth round. The light gray bytes are fixed during the sixth round. The dark gray bytes are fixed at the beginning of the seventh round whereas the bytes denoted with hatched cells are fixed at the end of the seventh round.

4.2 8-Round Distinguisher with 2^{25} Computations

In this section we describe the 8-round distinguisher. For rounds 5 and 6 the procedure is the same as the one for the 7-round distinguisher using the equations described in Table 2. However, now we would like also that the differences inserted during the seventh round cancel themselves, whereas the differences inserted during the eighth round can freely spread. In order to fulfill this requirement, after having handled the sixth round, instead of randomly choosing the value of k_5^0 one first chooses the values of s_0, s_1, s_2 and s_3, which will allow us to determine the difference in k_6^2.

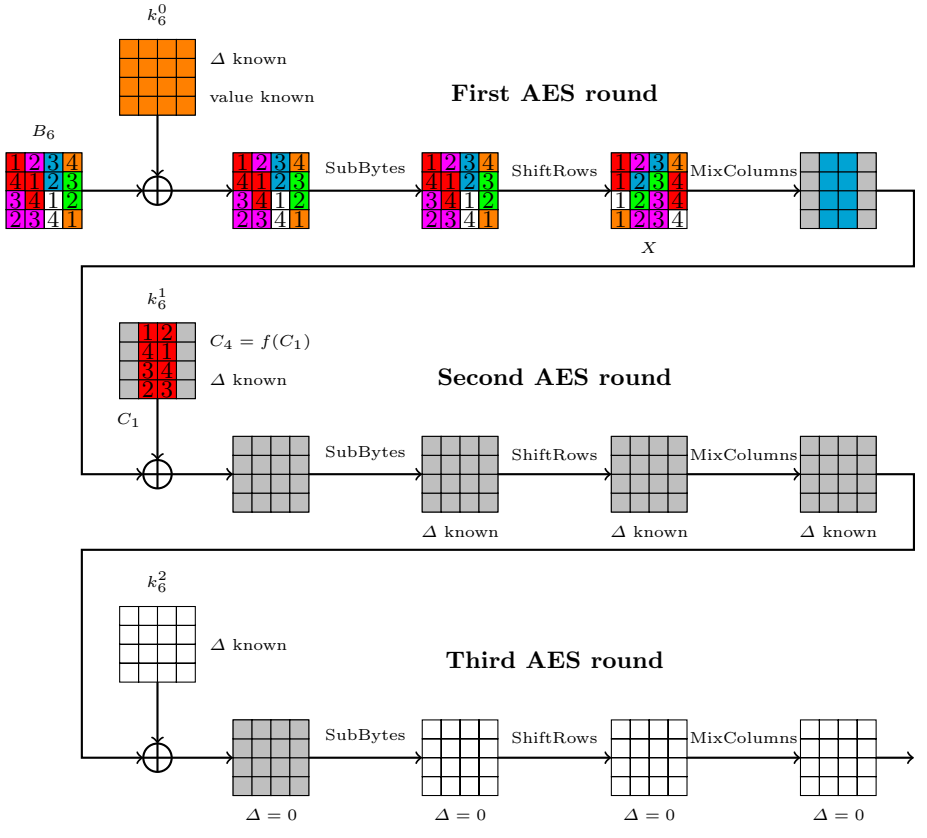

Fig. 7. The 3 AES rounds of the seventh round: the red bytes will determine the middle columns of k_6^1 (because of the message expansion and of previous conditions); the blue bytes will determine the differences in the 2 middle columns on the input of the second round; the orange bytes are fixed (due to round 6); the white and light gray bytes are free. The green bytes are the results of an XOR between blue bytes and orange bytes whereas pink bytes are the results of an XOR between blue bytes and red bytes.

Fixing the differences in the seventh round: We introduce the following notations: let S be a 128-bit AES state, we denote $(S)^i$ the i-th column of S and $(S*)^i$ the i-th column of $ShR(S)$ (i.e., the i-th diagonal of S), for $i \in [0..3]$. We give in Figure 7 a complete illustration of our attack.

Let us analyze the relations that link together the values already fixed and the values to be fixed. On the one hand, we have:

$$(B_6)^i \implies (A_5)^i = (B_4)^i \implies (k_4^0)^i \tag{2}$$

that one can read as "setting the value of $(B_6)^i$ will fix the value of $(A_5)^i$ (because the Feistel structure imposes $(A_5)^i = (B_4)^i$) which will in turn deduce the value of $(k_4^0)^i$". It is important to remark that the value of $(k_4^0)^3$ has already been fixed

in round 6, as shown in Table 2. Therefore, because of Relation (2), $(B_6)^3$ will also be known. Then, from the message expansion, we can derive the following relations:

$$(k_4^0)^i \implies (k_5^0)^{i+1} \implies (k_6^1)^{i+1} \text{ for } i \in \{0,1\}, \tag{3}$$

$$(k_4^0)^2 \implies (k_5^0)^3 \implies (k_6^1)^3 = (k_5^0)^3 \oplus (k_6^1)^0. \tag{4}$$

One can check that the values of column 0 and column 3 of k_6^1 are associated by a linear relation. We recall that for the time being the difference of k_6^1 is already known, but not its value.

On the other hand, we have $(B_6*)^i \implies (X)^i$, where X is the state represented in Figure 7 just before the first MC. We can then write the following relation:

$$SB[(k_6^0*)^i \oplus (B_6*)^i] \oplus SB[(k_6^0*)^i \oplus \Delta(k_6^0*)^i \oplus (B_6*)^i] = \Delta(X)^i. \tag{5}$$

For the path to be verified, we need the difference $MC(\Delta(X)^i) \oplus \Delta(k_6^1)^i$ to be compatible with the difference fixed by k_6^2 after the second SB, and the differential transition must be possible by the values of k_6^1 (which are not fixed yet). From the previous equations (2), (3), (4) and (5) we obtain that for making the path to be verified over each column i, the following columns or diagonals intervene at the same time: for $i = 0$, $\{(B_6*)^0, (B_6)^3\}$; for $i = 1$, $\{(B_6*)^1, (B_6)^0\}$; for $i = 2$, $\{(B_6*)^2, (B_6)^1\}$; for $i = 3$, $\{(B_6*)^3, (B_6)^2\}$.

From the previous relations we can deduce that there are some bytes of B_6 that interact in more than one way with the relations for one column i, ((3,3) for $i = 0$, (3,0) for $i = 1$, (3,1) for $i = 2$, (3,2) for $i = 3$). Thus, since we have defined the main relations that must be verified, we can now describe how to find a valid pair. A conforming pair is a pair of values that verifies the differential path of the seventh round as well as the path of the previous rounds. This can be done with a complexity of 2^{25} in time and 2^{14} in memory with the following process:

• We consider 2^{24} values of the bytes of $(B_6*)^1$ (the ones at byte positions $(0,1); (1,2); (2,3); (3,0)$). As the byte $(2,3)$ has an already fixed value, because it belongs to $(B_6)^3$ which was determined in the previous steps, the 2^{24} values are all the possible ones. Thus, the second column of the state after the first MixColumns of the 7th round will be determined by the previous values (i.e. the bytes of $(B_6*)^1$). The values of $(B_6*)^1$ that give us the match we are looking for are such that the differences before and after the second SubBytes match for the second column. Those differences are influenced for the first one by the already fixed differences of k_6^1 and for the second one by the already fixed differences of k_6^2. In other words, at this step, the differences are mainly fixed but not the values. Due to the `AES` Sbox differential properties, the match between the two differences will happen with a probability equal to 2^{-4}.

We have then to consider that, when we have fixed the values for the bytes $((0,1); (1,2); (2,3); (3,0))$, the value of the byte $(3,1)$ of k_6^1 will also be determined due to equations (2) and (3).

So, when we find a match of differences, we also need that one of the two values for $(3,1)$ that makes the match possible, collides with the already fixed

value for this byte. This will happen with a probability of 2^{-7}. We obtain: $2^{24} \cdot 2^{-4} \cdot 2^{-7} = 2^{13}$ values for $(B_6*)^1$ (bytes at positions $(0,1);(1,2);(2,3);(3,0)$) that make the differential path possible for the second column of the seventh round.

• We do the same thing with the bytes of $(B_6*)^2$ (bytes at positions $(0,2);(1,3);(2,0);(3,1)$), and we also obtain 2^{13} values that make the differential path possible for the third column of the seventh round.

• We now consider the interaction between the two previous steps in order to simultaneously find the solutions for the two middle columns. The byte at position $(0,1)$ of B_6 has already been fixed during the first step, and it determines the value of the byte $(0,2)$ of k_6^1, which belong to $(k_6^1)^2$ that affects the second step. Analogously the byte of the position $(2,0)$ from B_6 that was fixed at the second step determines the value of the byte $(2,1)$ of k_6^1, that belongs to $(k_6^1)^1$ and that affects the first step. Thus if we want to compute all the valid candidate values for the two columns in the middle before the second SB we will obtain: $2^{13} \cdot 2^{13} \cdot 2^{-7} \cdot 2^{-7} = 2^{12}$ possible values for fixing $(B_6*)^1$ and $(B_6*)^2$.

• We consider one of the previously determined values among the 2^{12} possibles. We consider now the yet unfixed bytes corresponding to positions $(1,0);(2,1)$ of B_6. Because of Relations (2), (3) and (4) they will determine the values of the associated bytes of k_6^1. Thus, since the differences are already fixed, each of these bytes has only 2 possible values that fulfill the difference match in the second SB for bytes $(1,1)$ and $(2,2)$. From $(B_6*)^3$ (i.e. the values from B_6 that will influence the fourth column in the second SB), we still have one byte, $(3,2)$, that has no influence on the already fixed parts. Therefore this byte can go through 2^8 distinct values. In total, for the fourth column after the first MC there are $2^{8+2} = 2^{10}$ possible values and differences that do not interfere with the differential path of the two middle columns in the second SB. We can compute how many of these 2^{12} values and differences for $(B_6*)^3$ could satisfy the path for the fourth column: $2^2 \cdot 2^8 \cdot 2^{-4} \cdot 2^4 = 2^{10}$ values for $(B_6*)^3$ make the fourth column on the second SubBytes also verify the path. The term 2^{-4} is present because one requires a possible match of differences and the term 2^4 comes from the fact that we can associate 2^4 values of the fourth column for one fixed difference before SubBytes.

• By applying the same method to the first column, we will obtain 2^{10} values for $(B_6*)^0$. We know from Relation (4) that $(k_6^1)^0$ and $(k_6^1)^3$ must satisfy a linear relation. This will occur for a fixed $(k_6^1)^0$ and a fixed $(k_6^1)^3$ with a probability of 2^{-32}. As we have 2^{10} possible values of $(k_6^1)^0$ from step 5 and 2^{10} possible values for $(k_6^1)^3$ from step 4, we obtain a valid couple $((k_6^1)^0, (k_6^1)^3)$ with probability 2^{-12}.

• We repeat step 5 about 2^{12} times with the different solutions of step 3, and we get a valid couple: we obtain all the valid values that verify the differential path, i.e. that have no differences after the 7th round.

Once those steps performed, all the desired values and differences that verify the path are determined except the ones in the first round. The values of s_4, s_5, s_6 and s_7 are not fixed yet and we can choose them as explained in Table 2.

Thus, Δ_1 will determine Δ_2 and we just compute backwards until we obtain the initial state that verifies the path (and consequently, also the first round). From this input, we get a corresponding output that contains no difference after seven rounds. If we apply the Davies-Meyer transformation, the right part of the state will not contain any difference. Thus, we have exhibited a free-start near-collision attack using chosen and related salts on a 7-round reduced version of the SHAvite-3-256 compression function used in the Davies-Meyer mode with 2^{25} computations and 2^{14} memory. The computation cost and memory requirements are quite far from the complexity corresponding to the ideal case (2^{64} operations).

Adding one more round at the end of the seventh round leads to a particular input/output structure (see Figure 4). More precisely, the subkeys differences of the eighth round are no more controllable, thus the right part A_8 of the state will contain differences but not the left side B_8 which is the seventh round's right part due to the Feistel structure. Therefore, after the Davies-Meyer transform, we will have the same difference in the left side of the input state and in the left side of the output state and we obtain a chosen-related-salt distinguisher on an 8-round reduced version of the SHAvite-3-256 compression function used in the Davies-Meyer mode with the same complexity as for the previous chosen-related-salt free-start near-collisions: 2^{25} computations and 2^{14} memory. Finding such a structure in the ideal case should require at least 2^{64} computations.

This 8-round distinguisher has been verified experimentally. We provide in the extended version an example of such a distinguisher.

5 Conclusion

In this paper, we have presented the first analysis of the (2^{nd}-round tweaked) 256-bit version of the SHA-3 competition candidate SHAvite-3. As it is the case for many candidates based on the AES round function, we showed that the Super-Sbox cryptanalysis and the rebound attacks are very efficient when analyzing reduced-round versions of SHAvite-3-256. Namely, without using the salt or the counter inputs, one can attack up to seven rounds of the twelve rounds composing the SHAvite-3-256 compression function. We were even able to reach eight rounds when the attacker is assumed to be able to control the salt input. Despite the attacks being quite involved, all our practical complexity results were verified experimentally.

References

1. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
2. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007), http://eprint.iacr.org/2007/278 (accessed on Janaury 10, 2010)

3. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (Round 2) (2009), http://www.cs.technion.ac.il/~orrd/SHAvite-3/Spec.15.09.09.pdf
4. Bouillaguet, C., Dunkelman, O., Leurent, G., Fouque, P.-A.: Attacks on Hash Functions based on Generalized Feistel - Application to Reduced-Round Lesamnta and SHAvite-3-512. Cryptology ePrint Archive, Report 2009/634 (2009), http://eprint.iacr.org/2009/634.pdf
5. Daemen, J., Rijmen, V.: The Design of Rijndael. In: Information Security and Cryptography. Springer, Heidelberg (2002) ISBN 3-540-42580-2
6. Gauravaram, P., Leurent, G., Mendel, F., Naya-Plasencia, M., Peyrin, T., Rechberger, C., Schläffer, M.: Cryptanalysis of the 10-round hash and full compression function of shavite-3-512. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 419–436. Springer, Heidelberg (2010)
7. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-like Permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010), http://eprint.iacr.org/2009/531
8. Khovratovich, D.: Cryptanalysis of Hash Functions with Structures. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 108–125. Springer, Heidelberg (2009)
9. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
10. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
11. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full LANE Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 106–125. Springer, Heidelberg (2009)
12. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
13. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
14. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard (April 1995), http://csrc.nist.gov
15. National Institute of Standards and Technology. FIPS PUB 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce (November 2001)
16. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a NewCryptographic Hash Algorithm (SHA-3) Family. Federal Register 27(212), 62212–62220 (2007), http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf (October 17, 2008)
17. Peyrin, T.: Cryptanalysis of GRINDAHL. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
18. Peyrin, T.: Chosen-salt, chosen-counter, pseudo-collision on SHAvite-3 compression function (2009), http://ehash.iaik.tugraz.at/uploads/e/ea/Peyrin-SHAvite-3.txt

19. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer, Heidelberg (2010), http://eprint.iacr.org/2010/223.pdf
20. Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992), http://www.ietf.org/rfc/rfc1321.txt
21. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (2002)
22. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
23. Wang, X., Yu, H.: How to break md5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)