Cryptanalysis of the 10-Round Hash and Full Compression Function of SHAvite-3-512*

Praveen Gauravaram[1], Gaëten Leurent[2], Florian Mendel[3],
María Naya-Plasencia[4], Thomas Peyrin[5],
Christian Rechberger[6], and Martin Schläffer[3]

[1] Department of Mathematics, DTU, Denmark
[2] ENS, France
[3] IAIK, TU Graz, Austria
[4] FHNW Windisch, Switzerland
[5] Ingenico, France
[6] ESAT/COSIC, K.U.Leuven and IBBT, Belgium
`martin.schlaeffer@iaik.tugraz.at`

Abstract. In this paper, we analyze the SHAvite-3-512 hash function, as proposed and tweaked for round 2 of the SHA-3 competition. We present cryptanalytic results on 10 out of 14 rounds of the hash function SHAvite-3-512, and on the full 14 round compression function of SHAvite-3-512. We show a second preimage attack on the hash function reduced to 10 rounds with a complexity of 2^{497} compression function evaluations and 2^{16} memory. For the full 14-round compression function, we give a chosen counter, chosen salt preimage attack with 2^{384} compression function evaluations and 2^{128} memory (or complexity 2^{448} without memory), and a collision attack with 2^{192} compression function evaluations and 2^{128} memory.

Keywords: hash function, cryptanalysis, collision, (second) preimage

1 Introduction

With the advent of new cryptanalysis [6, 20] of the FIPS 180-2 standard hash function SHA-1 [14], NIST has initiated an open hash function competition [15]. SHAvite-3 [3, 4], a hash function designed by Biham designed by Biham and Dunkelman, is a second round candidate in the SHA-3 hash function competition [16]. It is an iterated hash function based on the HAIFA hash function framework [2]. In this framework, the compression functions also accepts a salt

* This work was supported 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). Parts of this work were carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme, and while authors were participating in the ECRYPT2 workshop "Hash^3: Proofs, Analysis and Implementation" in November 2009. The first author is supported by the Danish Council for Independent Research (FTP and FNU) grant 274-09-0096.

and counter input in addition to the chaining value and message block. The mixing of the salt with the message aims to increase the security of hash-then-sign digital signatures against offline collision attacks [9,10], whereas the counter aims to thwart any attempts to mount some generic attacks on the iterated hash functions [1,8,11].

The SHAvite-3 compression function consists of a generalized Feistel structure whose round function is based on the AES. SHAvite-3 proposes two different instances called SHAvite-3-256 and SHAvite-3-512 with 12 and 14 rounds respectively. The first round version of SHAvite-3 has been tweaked for the second round of the SHA-3 competition due to a chosen salt and chosen counter collision attack [18] on the compression function. Recently, Bouillaguet et al. [5] have proposed the cancellation property to analyse hash functions based on a generalized Feistel structure. They have applied their method to find second preimages for 9 rounds of the SHAvite-3-512 hash function.

In this paper, we further analyze SHAvite-3-512 by improving the previous analysis of Bouillageut et al. [5]. We first present a chosen counter, chosen salt collision and preimage attack for the full compression function of SHAvite-3-512. The complexity for the preimage attack is 2^{384} compression function evaluations and 2^{128} memory (or complexity 2^{448} without memory), and for the collision attack we get 2^{192} compression function evaluations and 2^{128} memory. We then propose a second preimage attack on the hash function reduced to 10 rounds with a complexity of 2^{497} compression function evaluations and 2^{16} memory.

The paper is organised as follows: In Section 2, we briefly describe the SHAvite-3-512 hash function and in Section 3, we provide the fundamental ideas used in our attacks. In Section 4, we provide a preimage and collision attacks for the full 14 round compression function. In Section 5, we present a second preimage attack on the 10 round hash function and we conclude in Section 6.

2 The SHAvite-3-512 Hash Function

SHAvite-3-512 is used for the hash sizes of $n = 257, \ldots, 512$ bits. First, the message M is padded and split into ℓ 1024-bit message blocks $M_1 \| M_2 \| \ldots \| M_\ell$. Then, each message block is iteratively processed using the 512-bit compression function C_{512} and finally, the output is truncated to the desired hash size as follows:

$$\begin{aligned} h_0 &= IV \\ h_i &= C_{512}(h_{i-1}, M_i, salt, cnt) \\ hash &= trunc_n(h_i) \end{aligned}$$

The 512-bit compression function C_{512} of SHAvite-3-512 consists of a 512-bit block cipher E^{512} used in Davies-Meyer mode. The input of the compression function C_{512} consists of a 512-bit chaining value h_{i-1}, a 1024-bit message block M_i, a 512-bit *salt* and a 128-bit counter (*cnt*) to denote the number of bits processed by the end of the iteration. The output of the compression function C_{512} is given by (+ denotes an XOR addition):

$$h_i = C_{512}(h_{i-1}, M_i, salt, cnt) = h_{i-1} + E^{512}(M_i \| salt \| cnt, h_{i-1})$$

2.1 State Update

The state update of the compression function consists of a 14-round generalized Feistel structure. The input h_{i-1} is divided into four 128-bit chaining values (A_0, B_0, C_0, D_0). In each round $i = 0, \dots, 13$, these chaining values are updated using the non-linear round functions F_i and F'_i by the Feistel structure given as follows (also see Figure 1):

$$(A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}) = (D_i, A_i + F_i(B_i), B_i, C_i + F'_i(D_i))$$

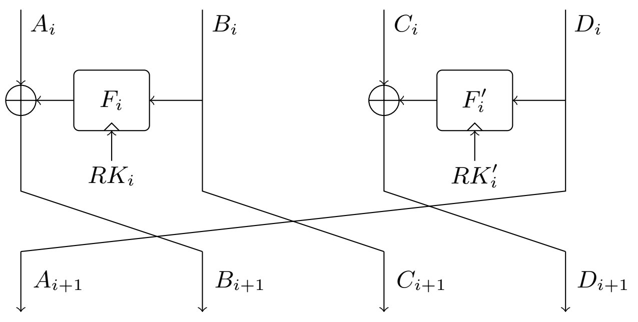

Fig. 1. Round i of the state update of SHAvite-512.

The non-linear functions F_i and F'_i are keyed by the 512-bit round keys $RK_i = (k^3_{0,i}, k^2_{0,i}, k^1_{0,i}, k^0_{0,i})$ and $RK'_i = (k^3_{1,i}, k^2_{1,i}, k^1_{1,i}, k^0_{1,i})$ respectively. Each round function is composed of four AES rounds with subkeys $k^0_{0,i}$ and $k^0_{1,i}$ used as a key whitening before the first AES round and an all zero-key 0^{128} for the last internal round. Hence, the round functions F_i and F'_i are defined as:

$$F_i(x) = AES(0^{128}, AES(k^3_{0,i}, AES(k^2_{0,i}, AES(k^1_{0,i}, k^0_{0,i} + x)))) \quad (1)$$

$$F'_i(x) = AES(0^{128}, AES(k^3_{1,i}, AES(k^2_{1,i}, AES(k^1_{1,i}, k^0_{1,i} + x)))) \quad (2)$$

2.2 Message Expansion

The message expansion of C_{512} (the key schedule of E^{512}) takes as input a 1024-bit message block, a 512-bit salt and 128-bit counter. The 1024-bit message block M_i is represented as an array of 8 128-bit words $(m_0, m_1, \dots, m_7)$, the 512-bit

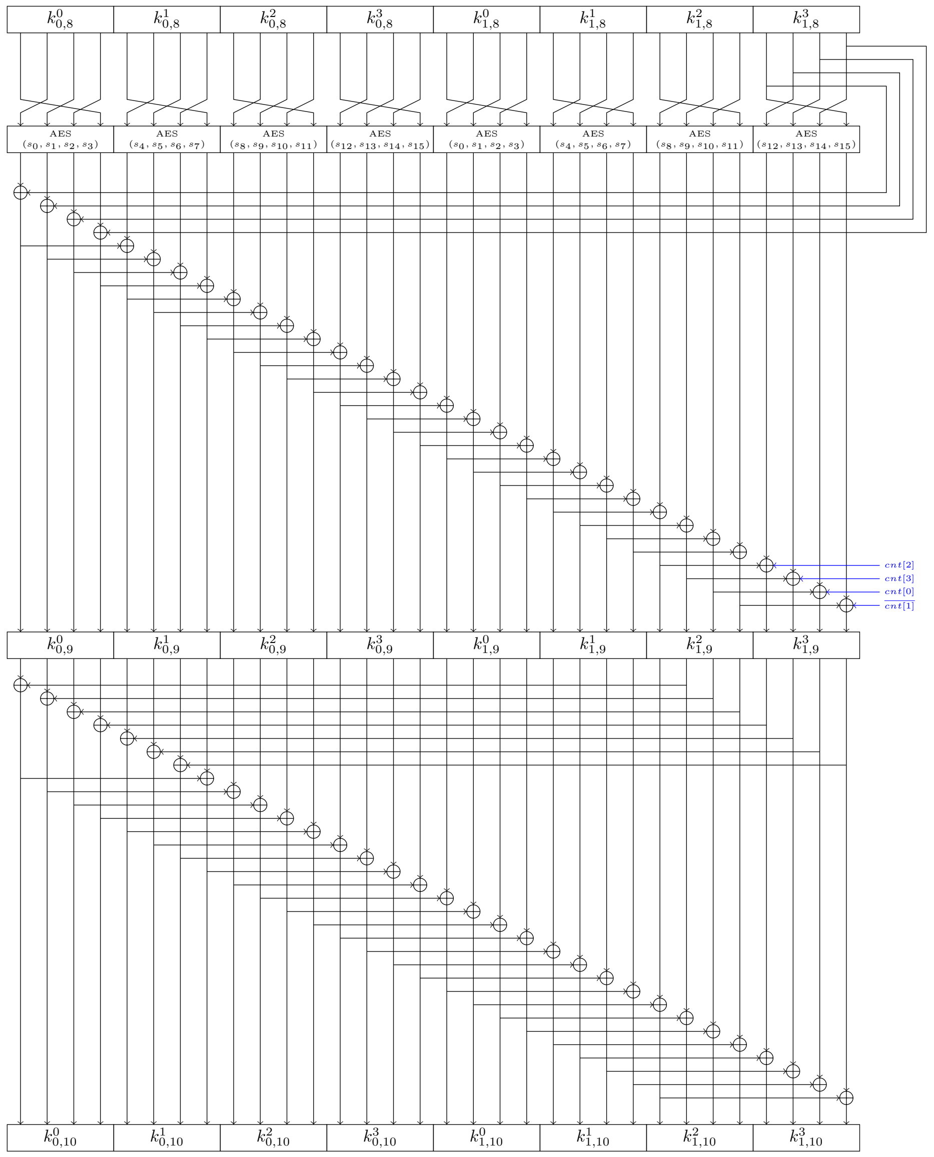

Fig. 2. One round ($r = 2$) of the message expansion (key schedule) of SHAvite-512 with the counter values ($cnt_2 \| cnt_3 \| cnt_0 \| \overline{cnt_1}$) added prior to subkey $k^3_{1,9}$.

salt as an array of 16 32-bit words $(s_0, s_1, \ldots, s_{15})$ and the counter as an array of 4 32-bit words $(cnt_0, cnt_1, cnt_2, cnt_3)$.

The subkeys for the odd rounds of the state update are generated using parallel AES rounds and a subsequent linear expansion step. The AES rounds are keyed by the salt words. The subkeys for the even rounds are computed using only a linear layer. One out of $r = 0, \ldots, 7$ rounds of the message expansion is shown in Figure 2. The first subkeys of round $r = 0$ are initialized with the message block:

$$(k^0_{0,0}, k^1_{0,0}, k^2_{0,0}, k^3_{0,0}, k^0_{1,0}, k^1_{1,0}, k^2_{1,0}, k^3_{1,0}) = (m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7)$$

Note that in rounds $r = 0, 2, 4, 6$ of the message expansion, the counter value (with one word inverted) is added to the subkeys:

$$\begin{aligned} k^0_{0,1} &= k^0_{0,1} + (cnt_0 \| cnt_1 \| cnt_2 \| \overline{cnt_3}) \\ k^1_{0,5} &= k^1_{0,5} + (cnt_3 \| cnt_2 \| cnt_1 \| \overline{cnt_0}) \\ k^3_{1,9} &= k^3_{1,9} + (cnt_2 \| cnt_3 \| cnt_0 \| \overline{cnt_1}) \\ k^2_{1,13} &= k^2_{1,13} + (cnt_1 \| cnt_0 \| cnt_3 \| \overline{cnt_2}) \end{aligned}$$

We refer to [4] for additional details on the message expansion.

3 Basic Attack Strategy

We first show how we can keep one chaining value of the state update constant using the cancellation property. Then, we interleave the cancellation property to simplify the resulting conditions and to keep the chaining value constant for a larger number of rounds. These properties can be used to construct partial preimages of the compression function. Note that we define a partial preimage attack as the task to find a preimage for only parts of the target hash value. Then, we extend this partial preimage attack to a collision or preimage attack of the compression function, or a (second) preimage of the hash function.

3.1 The Cancellation Property

The cancellation property was published by Bouillaguet et al. in the analysis of the SHA-3 candidates Lesamta and SHAvite-3 [5]. Using the cancellation property, a disturbance introduced in one state variable cancels itself 2 rounds later again. In our attacks on SHAvite-3-512, the cancellation property is used to ensure that $B_i = B_{i+4}$ in the state update. This is the case if and only if $F_{i+3}(B_{i+3}) = F'_{i+1}(D_{i+1})$ (see Table 1). Hence, a disturbance $F'_{i+1}(D_{i+1})$ introduced in round $i + 1$ cancels itself in round $i + 3$ (also see Figure 3).

Using $B_{i+3} = D_{i+1} + F_{i+2}(B_{i+2})$ and Equations (1) and (2) we get:

$$\begin{aligned} &AES(0^{128}, AES(k^3_{0,i+3}, AES(k^2_{0,i+3}, AES(k^1_{0,i+3}, k^0_{0,i+3} + D_{i+1} + F_{i+2}(B_{i+2})))) = \\ &AES(0^{128}, AES(k^3_{1,i+1}, AES(k^2_{1,i+1}, AES(k^1_{1,i+1}, k^0_{1,i+1} + D_{i+1})))). \end{aligned}$$

This equation and thus, the characteristic of Table 1, is fulfilled under the following conditions:

$$(k^3_{0,i+3}, k^2_{0,i+3}, k^1_{0,i+3}) = (k^3_{1,i+1}, k^2_{1,i+1}, k^1_{1,i+1}) \tag{3}$$

Table 1. We use the cancellation property to ensure that $B_i = B_{i+4}$. This is the case if $F_{i+3}(B_{i+3})$ and $F'_{i+1}(D_{i+1})$ are equal.

i	A_i	B_i	C_i	D_i	*condition*
i	?	B_i	?	?	
$i+1$	?	?	B_i	D_{i+1}	
$i+2$	D_{i+1}	B_{i+2}	?	$B_i + F'_{i+1}(D_{i+1})$	
$i+3$	$B_i + F'_{i+1}(D_{i+1})$	B_{i+3}	B_{i+2}	?	$F_{i+3}(B_{i+3}) = F'_{i+1}(D_{i+1})$
$i+4$	?	B_i	?	?	

and

$$k^0_{0,i+3} + F_{i+2}(B_{i+2}) = k^0_{1,i+1}. \tag{4}$$

Hence, using the cancellation property we always get $B_i = B_{i+4}$, no matter which value D_{i+1} has. Also differences in D_{i+1} are cancelled in round $i+4$ again. Of course, we can repeat the cancellation property for other rounds as long as we can fulfill the resulting conditions. The cancellation property is used twice in the 9-round hash attack on SHAvite-3-512 by Bouillaguet et al. [5] and in the 10-round hash attack in Section 5, and 4 times in the 13 and 14-round attack of the compression function in Section 4.

3.2 Interleaving

In this work, we extend the 9 round attack on SHAvite-3-512 by several rounds. This is possible by interleaving the cancellation property such that the conditions can be fulfilled more easily. Note that equation (4) depends on the chaining value B_{i+2}. In our attack, we use the following sufficient conditions, which allow us to fulfill the conditions on the keys and chaining values independently:

$$F_{i+2}(B_{i+2}) = 0 \tag{5}$$

$$RK_{i+3} = RK'_{i+1}. \tag{6}$$

Figure 3 shows that if the output of $F_{i+2}(B_{i+2})$ is zero, the same input D_{i+1} enters the round functions F_{i+3} and F'_{i+1}. Hence, if the keys of these non-linear functions are equal, any disturbance introduced by D_{i+1} cancels itself two rounds later. Note that we get equivalent conditions if we continue with $B_{i+4} = B_{i+8}$ and interleave with $B_{i+2} = B_{i+6}$. Hence, we get

$$B_i = B_{i+4} = B_{i+8} = \dots$$
$$B_{i+2} = B_{i+6} = B_{i+10} = \dots$$

if the following conditions on the chaining values are fulfilled:

$$F_{i+2}(B_{i+2}) = 0, \;\; F_{i+6}(B_{i+2}) = 0, \;\; F_{i+10}(B_{i+2}) = 0, \dots$$
$$F_{i+4}(B_i) = 0, \;\; F_{i+8}(B_i) = 0, \;\; F_{i+12}(B_i) = 0, \dots$$

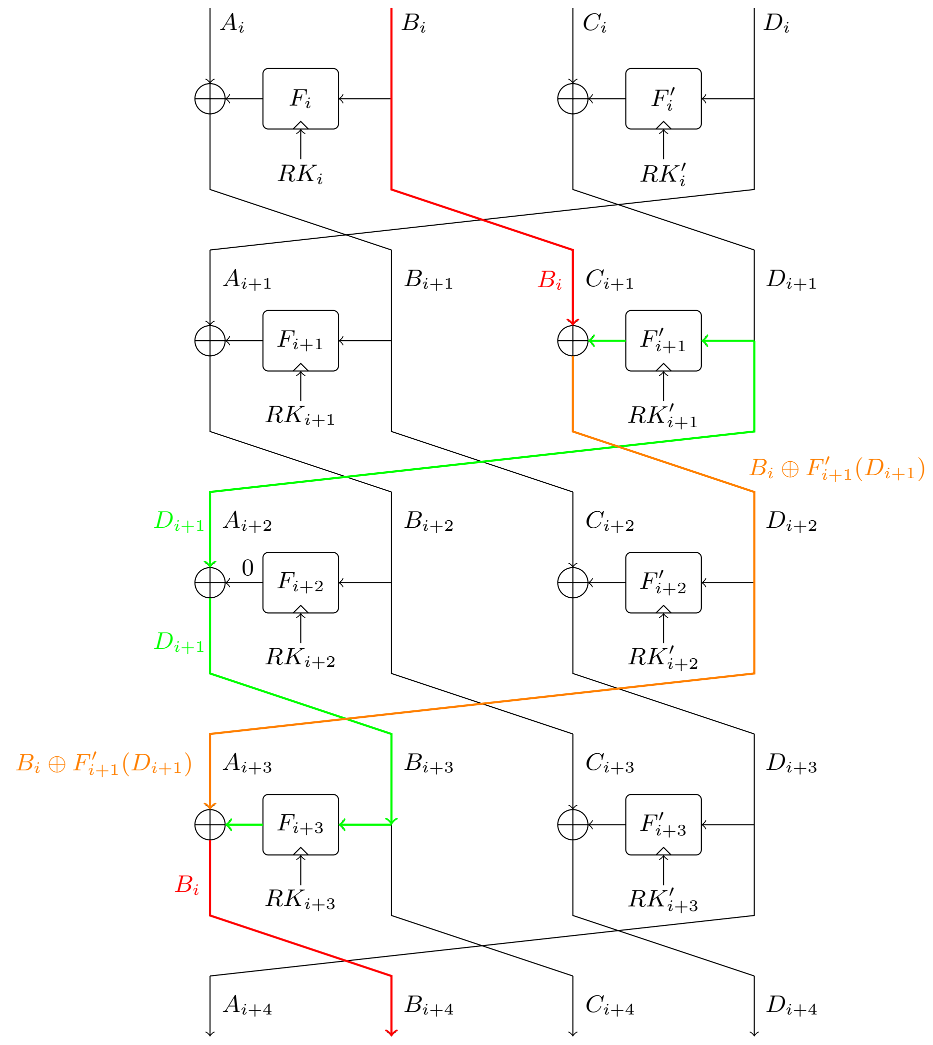

Fig. 3. We need to ensure that $B_i = B_{i+4}$ for a number of times. This is the case if $F_{i+3}(B_{i+3}) = F'_{i+1}(D_{i+1})$. However, we use the following sufficient conditions which can be fulfilled more easily using interleaving: $F_{i+2}(B_{i+2}) = 0$ and $RK_{i+3} = RK'_{i+1}$.

Since F is an invertible function, all these conditions are fulfilled if we choose $B_i = F^{-1}_{i+4}(0)$ and $B_{i+2} = F^{-1}_{i+2}(0)$, and the keys of the respective functions are the same. Hence, we get the following conditions only on the keys:

$$RK_{i+2} = RK_{i+6} = RK_{i+10} = \dots \tag{7}$$

$$RK_{i+4} = RK_{i+8} = RK_{i+12} = \dots \tag{8}$$

$$RK_{i+3} = RK'_{i+1},\ \ RK_{i+5} = RK'_{i+3},\ \ RK_{i+7} = RK'_{i+5}, \dots \tag{9}$$

An example starting with B_3 is given in Table 2.

Table 2. Interleaving.

i	A_i	B_i	C_i	D_i	*conditions*
3	?	B_3	?	?	
4	?	?	B_3	D_4	
5	D_4	B_5	?	$B_3 + F'_4(D_4)$	$F_5(B_5) = 0$
6	$B_3 + F'_4(D_4)$	D_4	B_5	D_6	$RK_6 = RK'_4$
7	D_6	B_3	D_4	$B_5 + F'_6(D_6)$	$F_7(B_3) = 0$
8	$B_5 + F'_6(D_6)$	D_6	B_3	D_8	$RK_8 = RK'_6$
9	D_8	B_5	D_6	$B_3 + F'_8(D_8)$	$RK_9 = RK_5$
10	$B_3 + F'_8(D_8)$	D_8	B_5	D_{10}	$RK_{10} = RK'_8$
11	D_{10}	B_3	D_8	$B_5 + F'_{10}(D_{10})$	$RK_{11} = RK_7$
...	...	...	...	...	...

3.3 From Partial Preimages to Preimages and Collisions

In the following attacks on SHAvite-3-512, we show how to fix one 128-bit output word H_i of the (reduced) compression function C_{512} to some predefined value:

$$H_0 \| H_1 \| H_2 \| H_3 = C_{512}(A_0 \| B_0 \| C_0 \| D_0, msg, salt, cnt)$$

Let's assume, we are able to construct a partial preimage on H_0 with a complexity of $2^x < 2^{128}$. Then, this partial preimage can be extended to construct a preimage or collision for the compression function below the generic complexity.

In a preimage attack on the compression function, we have to find some input values $A_0 \| B_0 \| C_0 \| D_0, msg, salt, cnt$ to the compression function for a given output $H_0 \| H_1 \| H_2 \| H_3$. For example, by repeating a partial preimage attack on H_0 about 2^{384} times, we expect to find a chaining value where also H_1, H_2 and H_3 are correct. In other words, we can find a preimage for the (reduced) compression function of SHAvite-512 with a complexity of about 2^{384+x}.

Similarly, a collision for the compression function can be constructed. If we can find 2^{192} inputs $A_0 \| B_0 \| C_0 \| D_0, msg, salt, cnt$ such that all produce the same output value H_0, two of the inputs also lead to the same values H_1, H_2, and H_3 due to the birthday paradox. Hence, we can construct a collision for the compression function of SHAvite-512 with a complexity of 2^{192+x}.

Further, by using an arbitrary first message block and a standard meet-in-the-middle attack we can turn the preimage attack on the compression function into a (second) preimage attack on the hash function. Note that in this case the *salt* and *cnt* values need to be the same for all partial preimages.

4 Attacks on the Compression Function

In this section, we present preimage and collision attacks for the SHAvite-3-512 compression function reduced to 13 and 14 rounds. We first give an outline of the attack and describe the characteristic used to find partial preimages. Then, we show how to find a message, salt and counter value according to the conditions of this characteristic. Finally, we extend the attack to find many message, salt and counter values such that the partial preimage attack can be extended to find a collision and preimage for 13 and 14 rounds. Note that the given preimage attack is an s-Pre (enhanced preimage) attack, as defined by Reyhanitabar et al. [19]. However, the attacks on the compression fucntion can not be extended to the hash function, since in the attack we have to choose the salt and counter value.

4.1 Outline of the Attack

In the attack on 13 and 14 rounds of the compression function, we use the same idea as in the 9-round attack in [5], but we extend it to more rounds by interleaving more cancellations. In detail, we use the cancellation property four times at rounds 6, 8, 10 and 12. This requires several 128-bit equalities on the key-schedule. We can satisfy the equalities using the degrees of freedom an attacker has in the choice of the message, salt, counter and chaining value in an attack on the compression function. Both, the 13 and 14 round attacks use the characteristic given in Table 3. For the 13 round attack we omit the last round.

Table 3. Characteristic for the attack on 13 and 14 rounds of the compression function. We can keep the value Z constant as long as the conditions are fulfilled.

i	A_i	B_i	C_i	D_i	*conditions*
0	?	?	?	?	
1	?	?	?	?	
2	?	X	?	?	
3	?	Z	X	?	
4	?	Y	Z	D_4	
5	D_4	Z	Y	$Z + F'_4(D_4)$	$F_5(Z) = 0$
6	$Z + F'_4(D_4)$	D_4	Z	D_6	$RK_6 = RK'_4$
7	D_6	Z	D_4	$Z + F'_6(D_6)$	$RK_7 = RK_5$
8	$Z + F'_6(D_6)$	D_6	Z	D_8	$RK_8 = RK'_6$
9	D_8	Z	D_6	$Z + F'_8(D_8)$	$RK_9 = RK_7$
10	$Z + F'_8(D_8)$	D_8	Z	D_{10}	$RK_{10} = RK'_8$
11	D_{10}	Z	D_8	$Z + F'_{10}(D_{10})$	$RK_{11} = RK_9$
12	$Z + F'_{10}(D_{10})$	D_{10}	Z	?	$RK_{12} = RK'_{10}$
13	?	Z	D_{10}	?	$RK_{13} = RK_{11}$
14	?	?	Z	?	

We start the attack using X, Z, Y, Z for B_2, B_3, B_4, B_5. Choosing $B_3 = B_5 = Z$ gives slightly simpler and more uniform conditions for the key schedule which can be fulfilled easier (see Section 4.2 and 4.3). The characteristic requires that the outputs of the functions F_5, F_7, F_9 and F_{11} are zero and we get:

$$F_5(Z) = F_7(Z) = F_9(Z) = F_{11}(Z) = 0 \ .$$

As already mentioned in the previous section, the best way to guarantee that the above conditions hold is to ensure that the four subkeys $RK_5, RK_7, RK_9, RK_{11}$ and hence, the functions F_5, F_7, F_9, F_{11} are equal. In this case Z can be easily computed by $Z = F_5^{-1}(0)$ and we get the following conditions for the key-schedule:

$$RK_5 = RK_7 = RK_9 = RK_{11} \ .$$

Next, we need to ensure that the output of $F'_4(D_4)$ and $F_6(D_4)$ is equal to one another such that $B_7 = Z + F'_4(D_4) + F_6(D_4) = Z$ after round 7. In total, the characteristic specifies that:

- $F'_4(D_4) = F_6(D_4)$ such that $B_7 = Z$
- $F'_6(D_6) = F_8(D_6)$ such that $B_9 = Z$
- $F'_8(D_8) = F_{10}(D_8)$ such that $B_{11} = Z$
- $F'_{10}(D_{10}) = F_{12}(D_{10})$ such that $B_{13} = Z$

and we get the following conditions on the key schedule for $i = 4, 6, 8, 10$ (also see Table 3):

$$RK'_i = RK_{i+2} \ .$$

If we can find a message, salt and counter value, such that all conditions on the state values and key-schedule are fulfilled, the characteristic of Table 3 is followed from round $i = 5$ to the end. Then, we simply compute backwards to get the inputs (A_0, B_0, C_0, D_0) of the compression function as a function of X, Y and Z (see also Table 4). Note that we can choose only values for X and Y since Z is fixed by the attack:

$$A_0 = Z + F_4(Y) + F'_2(Y + F_3(Z)) + F_0(Y + F_3(Z) + F'_1(Z + F_2(X))) \quad (10)$$

$$B_0 = Y + F_3(Z) + F'_1(Z + F_2(X)) \quad (11)$$

$$C_0 = Z + F_2(X) + F'_0(X + F_1(Z + F_4(Y) + F'_2(Y + F_3(Z)))) \quad (12)$$

$$D_0 = X + F_1(Z + F_4(Y) + F'_2(Y + F_3(Z))) \quad (13)$$

4.2 Finding the Message

An easy solution to fulfill all the conditions on the subkeys is to ensure that all keys are equal. This was possible for the round 1 version of SHAvite-512. By setting the message to the all zero value, each byte of the salt to `0x52` and the counter to 0, all subkeys are equal to zero [18]. In this case the required conditions are trivially fulfilled.

Table 4. We get the input of the 14-round characteristic as a function of X,Y and Z by computing backwards from round $i = 4$. We only show the updated chaining values A_i and C_i since $B_i = C_{i+1}$ and $D_i = A_{i+1}$.

i	A_i	C_i
0	?	$Z + F_2(X) + F'_0(X + F_1(Z + \dots$ $+F_4(Y) + F'_2(Y + F_3(Z))))$
1	$X + F_1(Z + F_4(Y) + F'_2(Y + F_3(Z)))$	$Y + F_3(Z) + F'_1(Z + F_3(X))$
2	$Z + F_2(X)$	$Z + F_4(Y) + F'_2(Y + F_3(Z))$
3	$Y + F_3(Z)$	X
4	$Z + F_4(Y)$	Z
5	D_4	Y
6	$Z + F'_4(D_4)$	Z
7	D_6	D_4
8	$Z + F'_6(D_6)$	Z
9	D_8	D_6
0	$Z + F'_8(D_8)$	Z
11	D_{10}	D_8
12	$Z + F'_{10}(D_{10})$	Z
13	?	D_{10}
14	?	Z

For the second round of the SHA-3 competition, SHAvite-512 has been tweaked and some counter words are inverted to prevent all zero keys. However, by choosing the counter to be

$$(cnt_3, cnt_2, cnt_1, cnt_0) = (0, 0, \overline{0}, 0),$$

the value $(cnt_2, cnt_3, cnt_0, \overline{cnt_1})$ added in round 5 of the key schedule is zero (see Figure 2). In contrast to the attack on the round 1 version, this results in a valid counter value. If we choose each byte of the salt to be `0x52` and the subkeys of round 5 to be zero, a large part of the subkeys will remain zero until non-zero values are added by the counter again. This happens in round 3 and round 7 of the key schedule. We only require that subkeys with round index $i = 4, \dots, 13$ are equal. Table 5 shows that indeed all required subkeys can be forced to zero. By computing backwards we get the message which is given in Appendix A. Since the key $RK_5 = 0$, we further get for $Z = F_5^{-1}(0) =$ `0x1919...19`.

4.3 Using Different Salt Values

Actually, we can relax the previous condition that all required subkeys are zero. This allows us to use many different salt values. Instead of trying to get the subkeys to be zero, we try to find subkeys which are periodic (see Table 6). Due to the nature of the key schedule, we will then get $RK_{i+2} = RK'_{i+2} = RK_i = RK'_i$

Table 5. Subkeys in SHAvite-512 where '0' denotes an all-zero subkey $k_{0,i}^{j}$ or $k_{1,i}^{j}$, and '?' denotes a subkey which is not zero. The counter values are XORed prior to the subkeys marked by *.

i	RK_i				RK'_i				r
	$k_{0,i}^{0}$	$k_{0,i}^{1}$	$k_{0,i}^{2}$	$k_{0,i}^{3}$	$k_{1,i}^{0}$	$k_{1,i}^{1}$	$k_{1,i}^{2}$	$k_{1,i}^{3}$	
0	?	?	?	?	?	?	?	?	M
1	?*	?	?	?	?	?	?	0	1
2	0	?	?	?	?	0	0	0	
3	0	?	?	?	0	0	0	0	2
4	0	?	0	0	0	0	0	0	
5	0	0*	0	0	0	0	0	0	3
6	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	4
8	0	0	0	0	0	0	0	0	
9	0	0	0	0*	0	0	0	0	5
10	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	6
12	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	?*	?	7

Table 6. Subkeys in SHAvite-512 with periodic round keys RK_i or RK'_i.

i	RK_i				RK'_i			
	$k_{0,i}^{0}$	$k_{0,i}^{1}$	$k_{0,i}^{2}$	$k_{0,i}^{3}$	$k_{1,i}^{0}$	$k_{1,i}^{1}$	$k_{1,i}^{2}$	$k_{1,i}^{3}$
i	a	b	c	d	a	b	c	d
$i+1$	e	f	g	h	e	f	g	h
$i+2$	a	b	c	d	a	b	c	d

as long as the counter does not interfere. This will be enough to apply the cancellation property, and to interleave it.

We can find salts and messages giving such a periodic expanded message by solving a linear system of equations. The message expansion alternates linear and non-linear layers, where the non-linear layer is one AES round with the salt used as the key. Note that the 512-bit salt is used twice in the 1024-bit message expansion. Hence, if we look for solutions with equal left and right halves, both halfs will still be equal after the non-linear layer. To find solutions, we construct a linear system with 1024 binary variables (or 32 32-bit variables), corresponding to the inputs and outputs of the non-linear layer. Hence, we get 1024 binary equations stating that the output of the linear layer must be equal to the input of the non-linear layer, such that the key-schedule will be periodic.

Each solution to this system gives an input and output of the non-linear layer. The resulting output and input of the single non-linear AES round can easily be connected by computing the according salt value. Then, we compute the message expansion backwards to get the message that will give a good expanded message for the given salt. Surprisingly, the system has many solutions, with a kernel of dimension 9, given by the following basis:

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1
\end{bmatrix}$$

This means that we can generate $2^{9\cdot 32} = 2^{288}$ different salt values and corresponding message blocks such that the conditions on the subkeys are fulfilled. An example of a random salt and message block in this class is given below:

```
Salt:    7f  51  e2  fb  ca  a5  95  ac  04  42  40  19  30  0f  17  82
         6d  31  01  30  86  30  0d  18  05  dc  db  90  96  2f  2a  78
         32  71  59  03  e3  bb  97  17  ee  41  bc  97  a3  b2  5c  18
         ce  af  fd  90  d6  8d  bf  fd  ab  11  9d  62  6a  11  13  b6
Message: e1  c1  44  35  67  84  1b  18  ca  ad  ac  f8  13  4d  ea  c9
         b1  a0  79  d1  e7  a9  11  ca  75  eb  96  82  cc  81  50  4f
         7a  69  43  38  8f  5d  e8  e4  e0  ce  3f  6b  55  1a  27  4e
         8d  e6  2d  9a  63  76  78  73  67  10  8e  d2  38  02  45  90
         12  16  0b  cc  6f  ab  c8  1a  ca  0e  c8  b7  5b  e7  33  93
         58  87  01  5f  09  b3  64  c3  a9  a2  5a  15  c9  70  a8  cb
         a0  80  ff  a8  c7  6d  24  60  09  8e  9d  15  0b  b1  af  8d
         5c  37  11  26  17  df  d7  eb  9f  bc  f0  82  2d  ad  98  e0
```

4.4 Collision and Preimages for 13 Rounds

If all the conditions on the key-schedule of Table 3 are fulfilled, the characteristic is followed and we get $B_{13} = Z$ after 13 rounds. For each value of X and Y (Z is fixed in the attack), we can compute backward to get the input of the compression function according to Equation (11) (also see Table 4).

After applying the feed-forward to B_0, we get for the output word H_1:

$$H_1 = B_0 + B_{13} = Z + Y + F_3(Z) + F_1'(Z + F_2(X))$$

For any value of H_1 we can simply choose a value X and compute Y. Hence, we can construct a partial preimage on 128 bits (on H_1) for the compression function of SHAvite-3-512 reduced to 13 rounds. The complexity is about one compression function evaluation. In Appendix A, we give 3 examples for inputs (chaining values) of the compression function of SHAvite-512 leading to outputs with H_1 equal to zero. Note that we can repeat this attack for all choices of X and hence, 2^{128} times.

To construct a collision and preimage for the whole output of the compression function, we need to repeat the partial preimage algorithm 2^{192} and 2^{384} times, respectively. Note that we can construct up to $2^{128+288} = 2^{416}$ partial preimages using the freedom in X and the message expansion.

4.5 Collision and Preimages for 14 Rounds

The 14-round attack follows the characteristic of Table 3. In order to extend the 13-round attack to 14 rounds, we simply add one round at the end. In this case, the conditions on the message, salt and counter value are the same as for 13 rounds. Again, we compute backwards to get the input of the compression function as a function of X, Y and Z (see Equation (12) and Table 4). Notice that after applying the feed-forward we get for the output word H_2:

$$H_2 = C_0 + C_{14} = F_2(X) + F_0'(X + F_1(Z + F_4(Y) + F_2'(Y + F_3(Z))))$$

which can be rewritten as:

$$F_0'^{-1}(H_2 + F_2(X)) + X = F_1(Z + F_4(Y) + F_2'(Y + F_3(Z))) \qquad (14)$$

This equation is nicely split using X only on the left-hand side, and Y only on the right-hand side. Any solution X, Y to the equation gives a 128-bit partial preimage for any chosen value of H_2. Note that we can also shift the characteristic by one round to get a partial preimage for H_1.

One easy way to find solutions for Equation (14) is to use a collision-finding algorithm: if we compute the left-hand side with 2^{64} random values for X, and the right-hand side with 2^{64} random values for Y, we expect to find one solution due to the birthday paradox with complexity 2^{64}. Note that this can be done with a memoryless collision-finding algorithm. The complexity can be reduced at the cost of higher memory requirements. By first saving 2^{128} candidates for X and then computing 2^{128} candidates for Y we get 2^{128} solutions for Equation (14) with a complexity of 2^{128}. Hence, we get an amortized cost of 1 computation per solution.

More generally, we can make a trade-off between time and memory using a distinguished point based collision-finding algorithm, as given in [17, Section 4.2]. Using 2^k bits of memory ($k \leq 128$) and 2^l processors, we can generate 2^{128} solutions choosing 128 bits of the output of the compression with complexity $2^{192-k/2-l}$. If we repeat this with several salts, we obtain the following attacks on the compression function:

- a collision attack in time $2^{256-k/2-l}$
- a preimage attack in time $2^{448-k/2-l}$

5 Attack on 10 Rounds of the Hash Function

The 10-round attack on the SHAvite-3-512 hash function is an extension of the 9-round attack from [5]. The extension is the same as the extension from 13 to 14 rounds of the compression function. Since the attack does not require freedom from the salt or counter values, it is a real attack on the 10-round SHAvite-3-512.

5.1 Extending the 9 Round Attack

We extend the 9-round attack by adding one round at the beginning according to the characteristic of Table 7. In order to satisfy the conditions in round 6 and 8, it is enough to have:

- $(k^1_{0,4}, k^2_{0,4}, k^3_{0,4}) = (k^1_{1,6}, k^2_{1,6}, k^3_{1,6})$ and $k^0_{0,4} + k^0_{1,6} = F_5(Z_5)$
- $(k^1_{0,6}, k^2_{0,6}, k^3_{0,6}) = (k^1_{1,8}, k^2_{1,8}, k^3_{1,8})$ and $k^0_{0,6} + k^0_{1,8} = F_7(Z_7)$

The condition on the keys $(k^1_{0,4}, k^2_{0,4}, k^3_{0,4}) = (k^1_{1,6}, k^2_{1,6}, k^3_{1,6})$ and $(k^1_{0,6}, k^2_{0,6}, k^3_{0,6}) = (k^1_{1,8}, k^2_{1,8}, k^3_{1,8})$ will be satisfied by carefully building a suitable message, following the algorithm given in [5]. Then, Z_5 and Z_7 are computed to satisfy the remaining conditions as follows: $Z_5 = F_5^{-1}(k^0_{0,4} + k^0_{1,6})$ and $Z_7 = F_7^{-1}(k^0_{0,6} + k^0_{1,8})$.

Table 7. Characteristic for the attack on 10 of the hash function. We can fix the output $C_i = Z_5$ as long as the conditions are fulfilled.

i	A_i	B_i	C_i	D_i	*condition*
0	?	?	?	?	
1	?	?	?	?	
2	?	X	?	?	
3	?	Z_7	X	?	
4	?	Y	Z_7	D_4	
5	D_4	Z_5	Y	$Z_7 + F'_4(D_4)$	
6	$Z_7 + F'_4(D_4)$	$D_4 + F_5(Z_5)$	Z_5	D_6	$F_6(D_4 + F_5(Z_5)) = F'_4(D_4)$
7	D_6	Z_7	?	$Z_5 + F'_6(D_6)$	
8	$Z_5 + F'_6(D_6)$	$D_6 + F_7(Z_7)$	Z_7	?	$F_8(D_6 + F_7(Z_7)) = F'_6(D_6)$
9	?	Z_5	?	?	
10	?	?	Z_5	?	

Then, we start with the state $B_2 = X$, $B_3 = Z_7$, $B_4 = Y$, and $B_5 = Z_5$ and compute backward to the input of the compression function, similar to Equation (12) and Table 4 of Section 4.1. In particular, we get for the input C_0:

$$C_0 = Z_7 + F_2(X) + F'_0(X + F_1(Z_5 + F_4(Y) + F'_2(Y + F_3(Z_7))))$$

and after applying the feed-forward, we get for the output word H_2:

$$H_2 = C_0 + C_{10} = Z_5 + Z_7 + F_2(X) + F'_0(X + F_1(Z_5 + F_4(Y) + F'_2(Y + F_3(Z_7))))$$

Just like in the 14-round attack, we can rewrite this equation such that one side depends only on X, and one side depends only on Y:

$$F_0'^{-1}(H_2 + Z_5 + Z_7 + F_2(X)) + X = F_1(Z_5 + F_4(Y) + F'_2(Y + F_3(Z_7)))$$

To extend this partial preimage to a full preimage attack on the hash function we repeat the following steps:

– find a suitable message (cost: 2^{224});
– find about 2^{128} solutions X and Y using a memoryless collision-search algorithm (cost: 2^{196}).

This generates 2^{128} inputs of the compression function such that 128 bits of the output are chosen. We need to repeat these steps 2^{256} times to get a full preimage on the compression function reduced to 10 rounds with a cost of 2^{480} and negligible memory requirements.

5.2 Second Preimage Attack

Using a first message block and by applying a generic unbalanced meet-in-the-middle attack, we can extend the preimage attack on the compression function to a second preimage attack on the hash function reduced to 10 rounds. The complexity is about 2^{497} compression function evaluations and 2^{16} memory. By using a tree based approach [7,12,13], the complexity of the attack can be reduced at the cost of higher memory requirements. Note that a preimage attack is not possible as we cannot ensure a correctly padded message block.

6 Conclusion

SHAvite-3-512 as considered during round 1 of the SHA-3 competition was shown to be subject to a chosen salt, chosen counter (pseudo) collision attack on the compression function. As a result, the compression function was tweaked by the designers. The tweaked SHAvite-3-512, as considered during round 2 of the SHA-3 competition, is here shown to still be succeptible to attacks in the same model, albeit at a higher cost. Although these attacks on the compression function do not imply an attack on the full hash function, they violate the collision resistance reduction proof of HAIFA. This illustrates that great care most be taken when salt and counter inputs are added to a compression function design. Furthermore, our analysis also illustrates that more than 70% (10 out of 14) of the rounds are not enough to defend the hash function SHAvite-3-512 against second preimage attacks.

Acknowledgements

We would like to thank Hitachi, Ltd. for supporting a predecessor of this work.

References

1. Andreeva, E., Bouillaguet, C., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second preimage attacks on dithered hash functions. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT Proceedings. LNCS, vol. 4965, pp. 270–288. Springer (2008)

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 10/1/2010)
3. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (2008), available online at `http://ehash.iaik.tugraz.at/uploads/f/f5/Shavite.pdf` (Accessed on 10/1/2010).
4. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Second round SHA-3 candidate (2009), available at `http://ehash.iaik.tugraz.at/wiki/SHAvite-3` (Accessed on 10/1/2010).
5. 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), available at `http://eprint.iacr.org/2009/634` (Accessed on 10/1/2010).
6. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT. LNCS, vol. 4284, pp. 1–20. Springer (2006)
7. De Cannière, C., Rechberger, C.: Preimages for Reduced SHA-0 and SHA-1. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008, Proceedings. LNCS, vol. 5157, pp. 179–202. Springer (2008)
8. Dean, R.D.: Formal Aspects of Mobile Code Security. Ph.D. thesis, Princeton University (1999)
9. Gauravaram, P., Knudsen, L.R.: On Randomizing Hash Functions to Strengthen the Security of Digital Signatures. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT. LNCS, vol. 5479, pp. 88–105. Springer (2009)
10. Halevi, S., Krawczyk, H.: Strengthening Digital Signatures Via Randomized Hashing. In: Dwork, C. (ed.) Advances in Cryptology – CRYPTO. LNCS, vol. 4117, pp. 41–59. Springer (2006)
11. Kelsey, J., Schneier, B.: Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. In: Cramer, R. (ed.) Advances in Cryptology–EUROCRYPT-2005. LNCS, vol. 3494, pp. 474–490. Springer (2005)
12. Leurent, G.: MD4 is Not One-Way. In: Nyberg, K. (ed.) FSE. LNCS, vol. 5086, pp. 412–428. Springer (2008)
13. Mendel, F., Rijmen, V.: Weaknesses in the HAS-V Compression Function. In: Nam, K.H., Rhee, G. (eds.) ICISC. LNCS, vol. 4817, pp. 335–345. Springer (2007)
14. NIST: FIPS PUB 180-2-Secure Hash Standard (Aug 2002), available at `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`(Accessed on 10/1/2010)
15. NIST: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Docket No: 070911510-7512-01 (November 2007)
16. NIST: Second Round Candidates. Official notification from NIST (2009), available at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html` (Accessed on 8/1/2010).
17. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. J. Cryptology 12(1), 1–28 (1999)
18. Peyrin, T.: Chosen-salt, chosen-counter, pseudo-collision on SHAvite-3 compression function (2009), available online at `http://ehash.iaik.tugraz.at/uploads/e/ea/Peyrin-SHAvite-3.txt` (Accessed on 10/1/2010)
19. Reyhanitabar, M.R., Susilo, W., Mu, Y.: Enhanced Security Notions for Dedicated-Key Hash Functions: Definitions and Relationships. In: Hong, S., Iwata, T. (eds.) FSE. LNCS, Springer (2010), to appear

20. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO. LNCS, vol. 3621, pp. 17–36. Springer (2005)

A Partial Preimage for 13 Rounds of the Compression of SHAvite-512

We give 3 examples for chaining inputs that all lead to a partial (128-bit) preimage of 0 for the compression function of SHAvite-512 reduced to 13 rounds. Note that the message block, the salt and counter values are equal in these examples.

```
 counter:           00 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00
 salt:              52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52
                    52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52
                    52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52
                    52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52
 message block M:   d1 58 6a 59 5f 1e ac c3 89 02 6a 23 8b 18 3d 35
                    a3 7b a6 8d 26 62 da 9a a6 8d 25 50 da 67 1e 62
                    0d fa 2b 8f a0 08 a4 97 b2 9b 25 0a 3e c3 6d c0
                    0b f7 12 3b d5 92 dd dc cf fa 79 ec 05 83 6e 9e
                    94 97 dd 03 4e e7 c1 07 8b f4 3d 9a df da 97 72
                    cc 24 50 90 0c 0a 0a b3 7c 58 d5 5d 7c 4d f9 ed
                    41 72 19 1a 8a ce 36 db ed fa 2e 40 23 66 8b d3
                    fa 1e 72 00 7b 8a 00 23 d3 00 49 88 00 96 79 19

 chaining value 1:  73 ae 12 97 3f 8f 59 33 83 e5 b8 79 9f 39 3f d6
                    19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
                    3f e8 9e 31 8c 13 5b 51 05 f6 26 2f ab 50 d0 2f
                    7e d4 37 2c 7e b3 6f e2 a3 8c 10 c1 30 cb 43 1f
 output 1:          f5 bb 28 52 27 67 80 b5 8d 68 2d 1b 66 f2 0c 1e
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    94 49 61 0d cc ed ea 7b 89 2a 90 ee e4 cc 49 0c
                    9c 3e 2e 17 78 f2 60 44 f5 f9 95 6c c0 dd 70 4f
 chaining value 2:  4d 96 cb 1f d7 26 9b f1 b8 84 e7 37 69 20 85 ee
                    19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
                    9c 0a 66 73 3f 9d 8e 4f 7d 15 85 71 6a cd fb 07
                    14 e6 c4 31 41 26 44 15 3a f8 a6 db b7 06 9a 4f
 output 2:          2a bf 6d c0 ef f7 78 b2 29 88 60 cc 04 63 22 6d
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    8b 45 69 a9 7f 67 5e 20 e4 8d 9b 01 d6 74 a9 dd
                    d3 9c 37 d1 ae ed 12 4d 47 d1 7c 28 72 26 1e 97
chaining value 3:   b3 56 96 56 1a 43 91 1e 7b 0c 3f 99 9c f2 6b be
                    19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
                    70 04 fd 88 dd b1 28 f2 03 6a 04 9c c2 65 b4 7b
                    2c d9 e6 74 aa 0b c5 78 85 e0 0c 21 89 ba 7f 8e
output 3:           31 a4 76 86 fa 16 f4 41 7a 93 6b 68 33 2d 46 c9
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    b1 a3 15 94 bc 41 2d fc 69 83 82 13 76 76 17 92
                    af 7e d8 93 c5 06 13 8e 05 2b 31 ab 65 cd 2a 51
```