Updates on SHAvite-3

Eli Biham[1] and Orr Dunkelman[2]

[1] Computer Science Department, Technion.
Haifa 32000, Israel
biham@cs.technion.ac.il
[2] Faculty of Mathematics and Computer Science
Weizmann Institute of Science
P.O. Box 26, Rehovot 76100, Israel
orr.dunkelman@weizmann.ac.il

Abstract. In this paper we present the current state of the SHA-3 submission SHAvite-3. We address the performance of SHAvite-3, and gather some implementation data, available at the moment. We then discuss some of the issues and results concerning the security of SHAvite-3. While we prove that the SHAvite-3 family offers secure hash functions, we also suggest a slightly tweaked version of SHAvite-3_{512} to offer a larger security margins.

1 Introduction

SHAvite-3 is hash function proposal in the SHA-3 competition [4,5]. It is based on combining the components of the AES as well as the HAIFA [3] framework to offer a secure and efficient hash function. The SHAvite-3 hash function is composed of two hash functions: SHAvite-3_{256} for digests of up to 256 bits, and SHAvite-3_{512} for digests of lengths between 257 and 512 bits. Each of these two functions uses a different compression function, but both of them share many similarities.

Both compression functions are a Davies-Meyer transformation of a block cipher (E^{256} and E^{512}, respectively). The first compression function, C_{256}, used in SHAvite-3_{256} accepts a chaining value of 256 bits, a salt of 256 bits, a bit counter of 64 bits, and a message block of 512 bits. The second compression, C_{512}, while accepts a chaining value of 512 bits, a salt of 512 bits, a bit counter of 128 bits, and a message block of 1024 bits.

The use of the AES round function as a base for computation allows reaching speeds of about 5.5 cycles per byte [2], which was measured for the original version of SHAvite-3. Following some minor security issues in the compression functions of SHAvite-3 in [15,16], SHAvite-3 was tweaked to offer a more resilient compression function [5].

In this note we discuss the current status of the SHAvite-3, and address its performance and security. We start by addressing a small mistake in the implementation of AES' round function. Fixing this mistake is expected to improve the performance of SHAvite-3 on platforms which contain support for AES' round function, e.g., machines with the AES-NI instruction set.

We continue the discussion with security analysis. We gather the recent results on SHAvite-3 [6,9,14] and discuss their implications on the security of the SHAvite-3 hash function.

This note is organized as follows: Section 2 discusses the performance issues. Section 3 addresses the security analysis. We conclude this note with Section 4.

2 The Performance of SHAvite-3

In this section we discuss the competitive performance of SHAvite-3 on various platforms. We start our discussion with a discussion about a mistake in the implementation of SHAvite-3, and its effects on the performance of SHAvite-3. We follow with a discussion on the software performance of SHAvite-3, and the effects of the new AES-NI instruction set on it. We conclude with the hardware performance of SHAvite-3.

2.1 Mistake in the Implementation of the AES Round Function

Our implementations of SHAvite-3 (including the reference code) relies on implementing the AES round function using the common approach of accessing tables which combine the MixColumns operation into the SubBytes operation. This allows an efficient implementation of SHAvite-3 on 32-bit platforms.

However, all these round functions contain a mistake, where the state is accidently *transposed* during the call for the tables. Namely, the current version of the relevant code is:

```
output[0] = Table0[input[0] >> 24] ^ Table1[(input[1] >> 16) & 0xff] ^ \
            Table2[(input[2] >> 8) & 0xff] ^ Table3[input[3] & 0xff];
output[1] = Table0[input[1] >> 24] ^ Table1[(input[2] >> 16) & 0xff] ^ \
            Table2[(input[3] >> 8) & 0xff] ^ Table3[input[0] & 0xff];
output[2] = Table0[input[2] >> 24] ^ Table1[(input[3] >> 16) & 0xff] ^ \
            Table2[(input[0] >> 8) & 0xff] ^ Table3[input[1] & 0xff];
output[3] = Table0[input[3] >> 24] ^ Table1[(input[0] >> 16) & 0xff] ^ \
            Table2[(input[1] >> 8) & 0xff] ^ Table3[input[2] & 0xff];
```

while is pointed out by Olivier Billet and Thomas Pornin, the correct round function should be:

```
output[0] = Table0[input[0] & 0xff ] ^ Table1[(input[1] >> 8) & 0xff] ^ \
            Table2[(input[2] >> 16) & 0xff] ^ Table3[(input[3] >>24)];
output[1] = Table0[input[1] & 0xff ] ^ Table1[(input[2] >> 8) & 0xff] ^ \
            Table2[(input[3] >> 16) & 0xff] ^ Table3[(input[0] >>24)];
output[2] = Table0[input[2] & 0xff ] ^ Table1[(input[3] >> 8) & 0xff] ^ \
            Table2[(input[0] >> 16) & 0xff] ^ Table3[(input[1] >>24)];
output[3] = Table0[input[3] & 0xff ] ^ Table1[(input[0] >> 8) & 0xff] ^ \
            Table2[(input[1] >> 16) & 0xff] ^ Table3[(input[2] >>24)];
```

We note that this issue is not expected to affect the speed performance of any implementation which is based on tables. However, code which is based on the AES-NI instruction set like the one used in [2], is expected to enjoy a speedup, as there will be no need to transpose the state each call for the AES round function. Hence, the number of instructions needed for the implementation of SHAvite-3 is expected to decrease, which is expected to improve the speed of such implementations.

2.2 Software Performance

Table 1 taken from [5] offers the speed figures obtained in software for SHAvite-3 by our measurements and ones made by eBASH [8] and by Benadjila et al. [2].

Hash Function	32-Bit Platform	64-Bit Platform
MD5	5.67	6.96
SHA-1	9.38	7.34
SHA-256	27.29	19.08
SHA-512	78.38	14.71
SHAvite-3_{256} (measured)	32.83	25.13
SHAvite-3_{256} (eBASH, Intel/AMD)	28.73–84.42	22.79–61.24
SHAvite-3_{256} (eBASH, PowerPC)	20.62–43.99	24.71–39.64
SHAvite-3_{256} (conjectured)	26.6	18.6
SHAvite-3_{256} (with AES inst.*)	5.6 [2]	
SHAvite-3_{256} (with AES inst.†)	4.8 [1]	
SHAvite-3_{256} (with AES inst.)	7.7 [7]	
SHAvite-3_{512} (measured)	55.90	35.86
SHAvite-3_{512} (eBASH, Intel/AMD)	55.30–242.09	40.28–255.10
SHAvite-3_{512} (eBASH, PowerPC)	32.00–184.78	38.41–64.39
SHAvite-3_{512} (conjectured)	35.3	28.4
SHAvite-3_{512} (with AES inst.*)	5.5 [2]	
SHAvite-3_{512} (with AES inst.†)	4.3 [1]	
SHAvite-3_{512} (with AES inst.)	8.7 [7]	
SHAvite-3_{512} (with AES inst.)	5	

* — the figures presented in [2] are estimates for untweaked SHAvite-3.
† — these figures are measured values for the tweaked SHAvite-3.

Table 1. Speed Comparison of Hash Functions (in cycles/byte)

We note that the results of [2] were estimated as the actual chip was not available at the time. We supplied Benadjila a tweaked version of the original code for measurements, and he provided us with the results of [1]. We note that recently, a new set of results is reported in [7].

At the same time, we supplied an implementation using AES-NI to the eBASH project. At the moment, the reported speeds are of 8 cycles/byte for SHAvite-3_{256} and of 8.9 cycles/byte for SHAvite-3_{512}, but with very high variability.

Our own measurements on an i5 machine show 5 cycles per byte, after the first few hundred calls to the compression function. The first (slower) calls as well as the discrepancy between our results and other measurements, seems to be related to the Intel's architecture, and we are in the process of finding a way to ensure the faster speed for all compression function calls.

Despite these issues, it is evident that the performance of SHAvite-3 are comparable with the speeds of MD5 and SHA1, and outperform the SHA2 family by far.

2.3 Hardware Performance

In [17] a comparative implementation of all second round SHA-3 candidates in 0.18μ ASIC-technology is given (even though mainly the 256-bit digest sizes were compared). Four versions of SHAvite-3_{256} were implemented (each with a different number of instances of the AES round function). The fastest result obtained used an area of 58,826 GE, with a clock frequency of 88.57 MHz, and latency of 19 cycles. This results in a throughput of 2.387 Gbit/sec.

FPGA implementations of 256-bit variants of the SHA-3 candidates were compared in [12] on Virtex-5 (XC5VLX30-3FF324). At a maximal clocking frequency of 251 MHz, and a core that takes 38 cycles. The throughput for long messages was 959 Mbps, which was the the third (after Luffa with 1,172 Mbps, and Keccak with 967 Mbps). The latency measured for short

	SHAvite-3_{256}			SHA-256		
Platform	Throughput (Mbps)	Area (CLBs/ALUTs)	Clock Freq. (MHz)	Throughput (Mbps)	Area (CLBs/ALUTs)	Clock Freq. (MHz)
Xilinx Spartan 3	1173.52	4114	84.60	715.56	838	90.84
Xilinx Virtex 4	2104.74	4114	152.23	1441.60	838	183.02
Xilinx Virtex 5	2885.89	1130	208.55	1630.49	433	207.00
Altera Cyclone II	1320.72	9400	95.40	874.65	1655	111.04
Altera Cyclone III	1420.85	9323	114.40	899.27	1653	126.86
Altera Stratix II	2353.39	2501	170.00	1245.18	973	158.08
Altera Stratix III	3528.65	2497	255.00	1676.29	963	212.81
	SHAvite-3_{512}			SHA-512		
Platform	Throughput (Mbps)	Area (CLBs/ALUTs)	Clock Freq. (MHz)	Throughput (Mbps)	Area (CLBs/ALUTs)	Clock Freq. (MHz)
Xilinx Spartan 3	1354.83	7997	75.31	1138.51	1367	90.06
Xilinx Virtex 4	2901.30	8544	161.97	2133.31	1403	168.75
Xilinx Virtex 5	3847.44	1951	213.45	2728.68	646	215.84
Altera Cyclone II	1560.94	20441	86.71	1182.53	2916	93.54
Altera Cyclone III	1859.57	20434	103.73	1430.44	2915	113.15
Altera Stratix II	2533.38	5507	140.53	2241.93	1639	177.34
Altera Stratix III	3858.84	5605	215.38	2968.34	1620	234.80

Table 2. Various SHAvite-3 FPGA Implementations

messages (of 1024 bits) was the fifth at 0.454 μs (to be compared with Luffa's 0.207 μs). The power consumption was 0.24 Watt, to be compared with the best result of 0.23 Watt (obtained for three SHA-3 candidates).

Another FPGA analysis of the SHA-3 candidates was performed in [10]. In this report, the implementations of SHAvite-3 on various FPGAs were considered. It was found that a throughput of 2,886 Mbps was achievable using 1,130 CLBs at 208 MHz for SHAvite-3_{256}, and 3,835 Mbps could be reached using 1,954 CLBs at 213 MHz. These results were obtained on the Xilinx Virtex 5 family of FPGAs. Using the Altera Stratix III, throughputs of 3,529 and 3,869 Mbps could be reached for SHAvite-3_{256} and SHAvite-3_{512}, respectively. We outline in Table 2 a summary of all implementations of SHAvite-3 reported in [10].

Two comments: We note that all reported implementations have implemented the AES round function from scratch. In the case the platform already supports AES, it seems that some of the AES implementation will be deemed redundant, and thus it is expected that SHAvite-3's implementation would actually need to introduce a much smaller circuit.

Moreover, it seems that some implementations took into consideration the salt. We note that when the salt is fixed, one could reduce the required resources for implementing SHAvite-3.

3 The Security of SHAvite-3

3.1 The Security of SHAvite-3_{256}

In [14] several attacks on reduced-round SHAvite-3_{256} are described. A 6-round free-start collision with time complexity 2^{120} time and 2^{56} memory is reported, along with its extension to a

Rounds	Attack	Time	Memory	Adversary's control
6	free-start collision	2^{120}	2^{56}	chaining value, message
7	comp. func. distinguisher	2^{120}	2^{56}	chaining value, message
7	free-start near-collision	2^{25}	2^{14}	chaining value, message, salt
8	comp. func. distinguisher	2^{25}	2^{14}	chaining value, message, salt

Table 3. Summary of Attacks on SHAvite-3_{256}

distinguisher on 7 rounds of the compression function. We note that the 6-round attack relies on removing the swap operation of the sixth round.

An additional attack reported in [14] is a 7-round free-start near-collision attack where the adversary controls the salt values. This attack has time complexity of 2^{25} compression function calls, and memory requirements of 2^{14}. The attack can be extended by one more round for a distinguishing attack on 8-round SHAvite-3_{256}, if the adversary controls the salt as well. We summarize the results of [14] in Table 3.

As can be clearly seen from the above results, SHAvite-3_{256} is a secure hash function which offers a large security margins.

3.2 The Security of SHAvite-3_{512}

In [6] the security of SHAvite-3_{512} is studied using the cancellation property. The result allows to attack 9-round of the compression function C_{512}, by finding (for a given salt and bit counter) an input chaining value and a message whose output has 128-bit value chosen by the adversary in an amortized time complexity of one compression function evaluation. This allows a 9-round collision attack on the compression function with complexity 2^{192} (rather than 2^{256}) and a 9-round pseudo-preimage attack on the compression function in time 2^{384}. The latter can be extended to attack the 9-round variant of the hash function in time of about 2^{448} compression function calls (and 2^{64} memory).

In [9] this attack is extended by one more round, and allows finding pseudo-preimages in 10-round reduced variant of C_{512} in time 2^{384} and memory of 2^{128}. These pseudo-preimage attacks translate to second-preimage attacks on 10-round SHAvite-3_{512} in time complexity 2^{448} and memory of 2^{128}.

Along these results, a chosen-salt, chosen-counter attacks on the full compression C_{512} are suggested in [9]. These attacks include a collision attack on C_{512} with time 2^{192} and 2^{128} memory, as well as a pseudo-preimage attacks with time 2^{384} and 2^{128} memory (or 2^{448} time). As mentioned before, these attacks assume that the adversary can obtain a full control over all the inputs of C_{512}, and can be used to distinguish C_{512} from a random function from 2176 bits to 512 bits.

3.3 The Effects of the Full Compression-Function Distinguisher

We note that the attack suggested by [9] shows that in the chosen-salt settings, the compression function of SHAvite-3_{512}, C_{512} is not ideal. However, while this may imply that SHAvite-3_{512} is an insecure hash function, following the results of [13,11], one can see that the SHAvite-3_{512} **is a secure hash**.

Rounds	Attack	Time	Memory	Adversary's control
9	Second Preimage (comp. func.) [6]	2^{384}	—	message block
	Second Preimage (hash. func.) [6]	2^{448}	2^{64}	message block
10	Second Preimage (comp. func.) [9]	2^{480}	—	message block
	Second Preimage (comp. func.) [9][1]	2^{448}	—	message block
	Second Preimage (comp. func.) [9][1]	2^{416}	2^{64}	message block
	Second Preimage (comp. func.) [9][1]	2^{384}	2^{128}	message block
	Second Preimage (hash. func.) [9]	2^{496}	2^{16}	message block
	Second Preimage (hash. func.) [9][1]	2^{480}	2^{32}	message block
	Second Preimage (hash. func.) [9][1]	2^{464}	2^{64}	message block
	Second Preimage (hash. func.) [9][1]	2^{448}	2^{128}	message block
14	Collision (comp. func.) [9]	2^{192}	2^{128}	message block, counter, salt, chaining value
	Preimage (comp. func.) [9]	2^{384}	2^{128}	message block, counter, salt, chaining value
	Preimage (comp. func.) [9]	2^{448}	—	message block, counter, salt, chaining value

[1] — using the improved algorithm of [6].

Table 4. Summary of the Attacks on SHAvite-3_{512}

We first note that according to [11], one can prove that despite the collision attack on the compression function, it is not possible to leverage it to a collision attack on the hash function. This is due to the fact that the adversary needs to be able to handle two consecutive compression function calls in order to find a collision in the actual hash function. As the attack on the compression function requires a specific counter value, and cannot be applied for any other counter value, one can conclude that this is indeed the case, and SHAvite-3_{512} itself is still a collision resistant, even against a chosen-salt adversary.

Moreover, the problem of indifferentiability in the presence of compression functions impurities was discussed in [13], and it was proven there that given the set of weak "states" (i.e., inputs), or pairs of weak "states", the loss in indifferentiability is linear with the size of the set of weak states and quadratic with the number of queries. Specifically, if the number of weak inputs to the compression function is $|\mathcal{W}|$ (in the case of the attack of [9], there are no issues with pairs of values), then the indifferentiability of the hash function is

$$16 \cdot \frac{q^2}{2^n} + 4|\mathcal{W}| \cdot \frac{q}{2^n}$$

rather than

$$8 \cdot \frac{q^2}{2^n}$$

for an n-bit state and q queries.

As $|\mathcal{W}| = 2^{128}$ for the attack of [9], then the indifferentiability of SHAvite-3_{512} is $16q^2/2^{512} + 4q/2^{384}$ (to be compared with $8q^2/2^{512}$. While the increase might seem high, it means that up to $q = 2^{128}$ queries to the compression function, the indifferentiability is roughly $4q/2^{384} < 2^{-254}$, and starting from $q = 2^{128}$, is about $16q^2/2^{512}$ which means that SHAvite-3_{512} is still secure until the birthday bound.

Conclusion about the Security of SHAvite-3_{512} Given the analysis presented before, we have a complete and utter confidence in the security of SHAvite-3_{512}. As can be seen from the theoretical results of [13,11], the effects of results which require one specific counter value are very limited.

Moreover, we note that the results on the full compression function use chosen salts. The set of possible weak salts, contains 2^{288} salts, which do not contain the default salt 0, used in applications where no salt is used.

3.4 The Tweak

Despite our confidence in the security of SHAvite-3_{512}, we decided to suggest a tweaked version of SHAvite-3_{512}, which is expected to offer larger security margins.

The main changes can be summarized as adding two more rounds to SHAvite-3_{512} (increasing the number of rounds from 14 to 16), and adding the counter in four more locations (in all eight nonlinear update steps, rather than only every second one). The counters added are the negation of the counter words added in the previous nonlinear update step. This also make SHAvite-3_{512}'s nonlinear update step more consistent with SHAvite-3_{256}'s one, as now the counter is used in every nonlinear update step.

Namely, the new message expansion algorithm in the tweaked SHAvite-3_{512} is as follows (where the tweak changes are *emphasized*):

- For $i = 0, \ldots, 31$ set $rk[i] \leftarrow msg[i]$.
- Set $i \leftarrow 32$
- Repeat *seven* times:
  1. **Nonlinear Expansion Step:** Repeat twice:
     (a) Let
     $$t[0..3] = AESRound_{0^{128}}\Big((rk[i-31]||rk[i-30]||rk[i-29]||rk[i-32]) \oplus (salt[0]||salt[1]||salt[2]||salt[3])\Big).$$
     (b) For $j = 0, \ldots, 3$: $rk[i+j] \leftarrow t[j] \oplus rk[i-4+j]$.
     (c) If $i = 32$ then $rk[32] \oplus= cnt[0]$, $rk[33] \oplus= cnt[1]$, $rk[34] \oplus= cnt[2]$, and $rk[35] \oplus= \overline{cnt[3]}$.
     (d) *If* $i = 112$ *then* $rk[112] \oplus= \overline{cnt[0]}$, $rk[113] \oplus= \overline{cnt[1]}$, $rk[114] \oplus= \overline{cnt[2]}$, *and* $rk[115] \oplus= cnt[3]$.
     (e) $i \leftarrow i + 4$.
     (f) Let
     $$t[0..3] = AESRound_{0^{128}}\Big((rk[i-31]||rk[i-30]||rk[i-29]||rk[i-32]) \oplus (salt[4]||salt[5]||salt[6]||salt[7])\Big).$$
     (g) For $j = 0, \ldots, 3$: $rk[i+j] \leftarrow t[j] \oplus rk[i-4+j]$.
     (h) If $i = 164$ then $rk[164] \oplus= cnt[3]$, $rk[165] \oplus= cnt[2]$, $rk[166] \oplus= cnt[1]$, and $rk[167] \oplus= \overline{cnt[0]}$.
     (i) *If* $i = 244$ *then* $rk[244] \oplus= \overline{cnt[3]}$, $rk[245] \oplus= \overline{cnt[2]}$, $rk[246] \oplus= \overline{cnt[1]}$, *and* $rk[247] \oplus= cnt[0]$.
     (j) $i \leftarrow i + 4$.
     (k) Let
     $$t[0..3] = AESRound_{0^{128}}\Big((rk[i-31]||rk[i-30]||rk[i-29]||rk[i-32]) \oplus (salt[8]||salt[9]||salt[10]||salt[11])\Big).$$

(l) For $j = 0, \ldots, 3$: $rk[i+j] \leftarrow t[j] \oplus rk[i-4+j]$.
(m) If $i = 440$ then $rk[440] \oplus = cnt[1]$, $rk[441] \oplus = cnt[0]$, $rk[442] \oplus = cnt[3]$, and $rk[443] \oplus = \overline{cnt[2]}$.
(n) $i \leftarrow i + 4$.
(o) Let

$$t[0..3] = AESRound_{0^{128}}\Big((rk[i-31]||rk[i-30]||rk[i-29]||rk[i-32]) \oplus (salt[12]||salt[13]||salt[14]||salt[15])\Big).$$

(p) For $j = 0, \ldots, 3$: $rk[i+j] \leftarrow t[j] \oplus rk[i-4+j]$.
(q) If $i = 316$ then $rk[316] \oplus = cnt[2]$, $rk[317] \oplus = cnt[3]$, $rk[318] \oplus = cnt[0]$, and $rk[319] \oplus = \overline{cnt[1]}$.
(r) *If* $i = 348$ *then* $rk[348] \oplus = \overline{cnt[2]}$, $rk[349] \oplus = \overline{cnt[3]}$, $rk[350] \oplus = \overline{cnt[0]}$, *and* $rk[351] \oplus = cnt[1]$.
(s) $i \leftarrow i + 4$.

2. **Linear Expansion Step:** Repeat 32 times:
   (a) $rk[i] \leftarrow rk[i-32] \oplus rk[i-7]$.
   (b) $i \leftarrow i + 1$.

– Repeat the **Nonlinear Expansion Step** an additional time *with the additional if condition: if* $i = 504$ *then* $rk[504] \oplus = \overline{cnt[1]}$, $rk[505] \oplus = \overline{cnt[0]}$, $rk[506] \oplus = \overline{cnt[3]}$, *and* $rk[507] \oplus = cnt[2]$.

We note that the new counter additions are done with the negated value of the previous nonlinear step. Moreover, if we consider the 32 new words generated in each nonlinear update step as two 16 words parts, then the location of the newly introduced counter words is similar to the previous nonlinear update step locations, up to swapping the half. This ensures a much faster diffusion of the counter values into the expanded message.

3.4.1 Effects on Security While it is obvious that the additional rounds offer a larger security margin, and thus add to the security of SHAvite-3_{512}, we continue to analyze the inherent problem that is the underlying cause for the results of [9]. That is, the ability to generate many expanded message words which satisfy equalities.

Consider the example given in [9] of a counter set to $(0, FFFF\ FFFF_x, 0, 0)$ and a salt which is set to be 52_x in all bytes. By picking the state of the message expansion to be all zeros in round 5, i.e., $rk[160, \ldots, 191] = 0$. Such a choice results in many of the $rk[\cdot]$ values being set to 0, and most importantly $rk[160, \ldots, 440]$ are all zero, as shown in Figure 5.

We analyzed the tweaked SHAvite-3_{512}, and the largest number of consecutive rounds with 0 subkey was found to be 4, i.e., the adversary can maintain up to 4 consecutive pairs of subkeys $(RK_{0,i}, RK_{1,i})$, for which all subkeys are 0. Moreover, we analyzed all the possible combination, and we found out that the maximal number of words that can be fixed to 0 is 254 out of the 512 expanded message words, and that in that case 13 additional words have a "structured" difference (namely, $FFFF\ FFFF_x$). We outline the best possible expansion in Table 6.

We note that similar analysis confirms that it is impossible to achieve 2-round cyclic subkeys for more than 4 rounds (due to the different counter values). Hence, the attacks of [9] can no longer be applied (we note that the updated message expansion is also likely to affect some of the results presented in [6]).

3.4.2 Effects on Performance These tweaks to the message expansion are expected to have a negligible effect on the performance. Hence, we expect that the only performance overhead

Round	$RK_{0,i}$				$RK_{1,i}$			
i	$k^0_{0,i}$	$k^1_{0,i}$	$k^2_{0,i}$	$k^3_{0,i}$	$k^0_{1,i}$	$k^1_{1,i}$	$k^2_{1,i}$	$k^3_{1,i}$
0	?	?	?	?	?	?	?	?
1	?*	?	?	?	?	?	?	0
2	0	?	?	?	?	0	0	0
3	0	?	?	?	0	0	0	0
4	0	?	0	0	0	0	0	0
5	0	0*	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0*
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	?*	?

0 stands for a zero value, ? for an unknown value,
* marks where the counter is mixed.

Table 5. The Message Expansion of SHAvite-3_{512} for $rk[160, \ldots, 191] = 0$, $salt = 52_x \ldots 52_x$ and $counter = (0, FFFF\ FFFF_x, 0, 0)$

offered by the tweak is expected to be a reduction of 16/14 in the speed, i.e., it would take about 15% to hash the same amount of data.

Given SHAvite-3_{512}'s competitive performance, we feel that this performance overhead is well worth the increased security and confidence in SHAvite-3_{512}'s.

4 Summary and Conclusions

In this update, we have discussed the security and performance of SHAvite-3. We summarized some performance figures concerning SHAvite-3, and showed that SHAvite-3 offers a competitive performance with the SHA-2 family. Moreover, we addressed the recent security concerns regarding SHAvite-3, proving that SHAvite-3 is a secure hash function (despite some results on the full C_{512}). Finally, we offered a tweak to SHAvite-3_{512} which will improve its security margins significantly, without affecting the performance too much.

Hence, we conclude that either SHAvite-3, as submitted to the second round, or the tweaked version described in this update, are secure and efficient hash functions, which can be implemented on various platforms with various constraints. Hence, we strongly believe that the tweaked SHAvite-3 is suitable to be selected as SHA-3.

We will soon release the fully updated code of the tweaked and fixed SHAvite-3 hash function family (the code of SHAvite-3_{256} would contain a fix of the round function transposition, and the one of SHAvite-3_{512} would incorporate this fix with the tweak).

Acknowledgments

We would like to thank Olivier Billet and Thomas Pornin for detecting the mistake in the implementation of SHAvite-3. We would also like to thank them along with Ryad Benadjila and Dan J. Bernstein for discussions concerning the implementation of SHAvite-3.

Round	$RK_{0,i}$				$RK_{1,i}$			
i	$k^0_{0,i}$	$k^1_{0,i}$	$k^2_{0,i}$	$k^3_{0,i}$	$k^0_{1,i}$	$k^1_{1,i}$	$k^2_{1,i}$	$k^3_{1,i}$
0	?	?	?	?	?	?	?	?
1	$?^{\dagger\star}$	?	?	?	?	?	$?^{\dagger}$	0
2	0	?	?	?	?	0	0	0
3	0	?	$?^{\dagger}$	$?^{\dagger}$	$0^{\star}$	0	0	0
4	0	?	0	0	0	0	0	0
5	0	$0^{\star}$	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	$0^{\star}$	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	$?^{\dagger}$	$0^{\star}$
10	$?^{\dagger}$	$?^{\dagger}$	$?^{\dagger}$	0	$?^{\dagger}$	0	$?^{\dagger}$	$?^{\dagger}$
11	?	?	?	$?^{\star}$	?	?	?	?
12	?	?	?	?	?	?	?	?
13	?	?	?	?	?	?	$?^{\star}$	?
14	?	?	?	?	?	?	?	?
15	?	?	$?^{\star}$	?	?	?	?	?

0 stands for a zero value, ? for an unknown value,
* marks where the counter is mixed,
† marks that the difference in some subwords is 0 or $FFFF\ FFFF_x$.

Table 6. The New Message Expansion of SHAvite-3_{512} for $rk[160,\ldots,191] = 0$, $salt = 52_x \ldots 52_x$ and $counter = (FFFF\ FFFF_x, 0, FFFF\ FFFF_x, FFFF\ FFFF_x)$

We would like to express our gratitude to Charles Bouillaguet, Pierre-Alain Fouque, Gaëtan Leurent, Praveen Gauravaram, Florian Mendel, María Naya-Plasencia, Thomas Peyrin, Christian Rechberger, Martin Schläffer, and Marina Minier for taking the effort and analyzing the strength of SHAvite-3.

References

1. Benadjila, R.: Measurements of shavite-3. private communications (July 2010)
2. Benadjila, R., Billet, O., Gueron, S., Robshaw, M.J.B.: The intel aes instructions set and the sha-3 candidates. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 162–178
3. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions — HAIFA. Presented at the second NIST hash workshop (August 24–25 2006)
4. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (2008)
5. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function (Tweaked Version). Submission to NIST (2009)
6. 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}. Accepted to SAC 2010, full version at IACR ePrint Archive, Report 2009/634 (2009) `http://eprint.iacr.org/`.
7. ECHO Team: ECHO hash function – Intel's AES-NI Instructions Set (August 2010) `http://crypto.rd.francetelecom.com/ECHO/sha3/AES/`.
8. ECRYPT: Ecrypt benchmarking of all submitted hashes (2010)

9. 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. Volume 6055 of Lecture Notes in Computer Science., Springer (2010) 419–436
10. Homsirikamol, E., Rogawski, M., Gaj, K.: Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. Cryptology ePrint Archive, Report 2010/445 (2010) `http://eprint.iacr.org/`.
11. Indesteege, S.: Analysis and Design of Cryptographic Hash Functions. PhD thesis, Katholieke Universiteit Leuven (2010)
12. Kobayashi, K., Ikegami, J., Knežević, M., Guo, E.X., Matsuo, S., Huang, S., Nazhandali, L., Ünal Kocabas, Fan, J., Satoh, A., Verbauwhede, I., Sakiyama, ., Ohta, K.: Prototyping Platform for Performance Evaluation of SHA-3 Candidates. 3rd IEEE International Workshop on Hardware-Oriented Security and Trust (HOST 2010) (2010) Available online at `https://www.cosic.esat.kuleuven.be/publications/article-1455.pdf`.
13. Leurent, C.B.G., Fouque, P.A.: Security Analysis of SIMD. Accepted to SAC 2010, full version at IACR ePrint Archive, Report 2010/323 (2010) `http://eprint.iacr.org/`.
14. Minier, M., Naya-Plasencia, M., Peyrin, T.: Distinguishers on the Reduced-round SHAvite-3-256 Compression Function. private communications (May 2010)
15. Nandi, M., Paul, S.: OFFICIAL COMMENT: SHAvite-3. Available online (2009)
16. Peyrin, T.: Chosen-salt, chosen-counter, pseudo-collision on SHAvite-3 compression function. Available online (2009)
17. Tillich, S., Feldhofer, M., Kirschbaum, M., Plos, T., Schmidt, J.M., Szekely, A.: High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein. IACR ePrint Report 2009/510 (2009)