Changes in the SHAvite-3 Submission Package

Eli Biham[1,⋆] and Orr Dunkelman[2,⋆⋆]

[1] Computer Science Department, Technion
Haifa 32000, Israel
biham@cs.technion.ac.il
[2] École Normale Supérieure
Département d'Informatique,
CNRS, INRIA
45 rue d'Ulm, 75230 Paris, France
orr.dunkelman@ens.fr

Abstract. This document lists the different modifications done in the SHAvite-3 submission package, including the tweak suggested for SHAVite-3.

The changes summarized in this report were divided according to their cause, to facilitate an easy evaluation of the changes. All the locations are given with respect to the version submitted to NIST on January 15th, 2009.

1 Tweak Related Changes

As mentioned in the submission, we have suggested a small tweak for SHAvite-3, to increase the security of the compression function.

⋆ The first author was supported in part by the Israel MOD Research and Technology Unit.
⋆⋆ The second author was supported by the France Telecom Chaire.

1.1 The Tweak

Location	Original Text	New Text
Sect. 4: Specifications of SHAvite-3		
Pp. 10, first para.	Eight of the produced words are XORed with the counter (four with $cnt[0]$ and four with $cnt[1]$), thus preventing any slide properties of the cipher: $rk[16], rk[54], rk[91]$, and $rk[124]$ are XORed with $cnt[0]$ during their update, and $rk[17], rk[53], rk[90]$, and $rk[127]$ are XORed with $cnt[1]$.	Eight of the produced words are XORed with the counter (four with $cnt[0]$ and four with $cnt[1]$), where in four of these positions, the counter is complemented,[1] thus preventing any slide properties of the cipher: $rk[16], rk[58]^\star, rk[87]^\star$, and $rk[124]$ are XORed with $cnt[0]$ during their update, and $rk[17]^\star, rk[53], rk[90]$, and $rk[127]^\star$, are XORed with $cnt[1]$. All the locations marked by * are actually XORed with the complemented value of the respective counter.
– " –	Also added footnote 1:	We note that this is the only difference between the original submitted version of SHAvite-3_{256} and the tweaked version.
Pp. 10, bullet 1(c)	If $i = 16$ then $rk[16]\oplus = cnt[0]$ and $rk[17]\oplus = cnt[1]$.	If $i = 16$ then $rk[16]\oplus = cnt[0]$ and $rk[17]\oplus = \overline{cnt[1]}$.
Pp. 10, bullet 1(d)	If $i = 84$ then $rk[86]\oplus = cnt[1]$ and $rk[87]\oplus = cnt[0]$.	If $i = 84$ then $rk[86]\oplus = cnt[1]$ and $rk[87]\oplus = \overline{cnt[0]}$.
Pp. 10, bullet 1(h)	If $i = 56$ then $rk[57]\oplus = cnt[1]$ and $rk[58]\oplus = cnt[0]$.	If $i = 56$ then $rk[57]\oplus = cnt[1]$ and $rk[58]\oplus = \overline{cnt[0]}$.
Pp. 10, bullet 1(i)	If $i = 124$ then $rk[124]\oplus = cnt[0]$ and $rk[127]\oplus = cnt[1]$.	If $i = 124$ then $rk[124]\oplus = cnt[0]$ and $rk[127]\oplus = \overline{cnt[1]}$
Pp. 13, Sect. 4.2.2, second para.	Sixteen of the produced words are XORed with the counter (four with each $cnt[i]$).	Sixteen of the produced words are XORed with the counter (four with each $cnt[i]$), where four times the XORed counter word is complemented.[2]
– " –	Also added footnote 2:	We note that this is the only difference between the original submitted version of SHAvite-3_{512} and the tweaked version.
Pp. 14, bullet 1(c)	If $i = 32$ then $rk[32]\oplus = cnt[0]$, $rk[33]\oplus = cnt[1]$, $rk[34]\oplus = cnt[2]$, and $rk[35]\oplus = cnt[3]$.	If $i = 32$ then $rk[32]\oplus = cnt[0]$, $rk[33]\oplus = \underline{cnt[1]}$, $rk[34]\oplus = cnt[2]$, and $rk[35]\oplus = \overline{cnt[3]}$.
Pp. 14, bullet 1(g)	If $i = 164$ then $rk[164]\oplus = cnt[3]$, $rk[165]\oplus = cnt[2]$, $rk[166]\oplus = cnt[1]$, and $rk[167]\oplus = cnt[0]$.	If $i = 164$ then $rk[164]\oplus = cnt[3]$, $rk[165]\oplus = \underline{cnt[2]}$, $\underline{rk[166]}\oplus = cnt[1]$, and $rk[167]\oplus = \overline{cnt[0]}$.

Location	Original Text	New Text
Pp. 15, bullet 1(k)	If $i = 440$ then $rk[440] \oplus = cnt[1]$, $rk[441] \oplus = cnt[0]$, $rk[442] \oplus = cnt[3]$, and $rk[443] \oplus = cnt[2]$.	If $i = 440$ then $rk[440] \oplus = cnt[1]$, $rk[441] \oplus = cnt[\underline{0}]$, $\underline{rk}[442] \oplus = cnt[3]$, and $rk[443] \oplus = cnt[2]$.
Pp. 15, bullet 1(o)	If $i = 316$ then $rk[316] \oplus = cnt[2]$, $rk[317] \oplus = cnt[3]$, $rk[318] \oplus = cnt[0]$, and $rk[319] \oplus = cnt[1]$.	If $i = 316$ then $rk[316] \oplus = cnt[2]$, $rk[317] \oplus = cnt[\underline{3}]$, $\underline{rk}[318] \oplus = cnt[0]$, and $rk[319] \oplus = cnt[1]$.

1.2 Related Changes

Location	Original Text	New Text
Sect. 4: Specifications of SHAvite-3		
Pp. 12, Sect. 4.1.4		The value of MIV_{256} has changed.
Pp. 13, Table 2		All the values of IV_m presented in the table have changed.
Pp. 16, Table 3		All the values of IV_m presented in the table have changed.
Pp. 16, Sect. 4.2.4		The value of MIV_{512} has changed.
Sect. 4.3	Became Sect. 4.4, and the following text was added as Sect. 4.3:	**Changes from the Original Submission** As noted earlier, the only difference between the new C_{256} and its previous version is the fact that in four out of the eight positions where counter words are XORed, the counter word is complemented (specifically, twice $cnt[0]$ and twice $cnt[1]$). This insures that at least four of the eight counter words which enter the computation are non-zero. The only difference between the new C_{512} and its previous version is the fact that four out of sixteen positions where the counter words are XORed, the counter word is complemented (each of the four words is complemented once). This insures that at least four of the sixteen counter words which enter the computation are non-zero.
Sect. 5: Design Criteria and Rationale		
Pp. 19, Sect. 5.4, first para.	and the locations of where the counter and salt are mixed.	and the locations of where the counter and salt are mixed, as well as when the counter is complemented.

Location	Original Text	New Text
Pp. 19, Sect. 5.4, fifth para.	Added a new paragraph after:	Following the concerns raised in [57, 61] (which had no impact on the security of SHAvite-3 as a hash function), we tweaked the compression functions of SHAvite-3. The tweak was chosen to be the complementation of counter words in four locations (out of the eight or 16 locations where the counter is XORed into the expanded message). The complementation locations were chosen to ensure that no slide properties would exist even for specially chosen message and salt values. For ease of implementation, each time the counter is XORed into the state, the last word of the counter is complemented. This allows software implementations to precompute the complementation, thus reducing the effect this may have on the running time.
Sect. 6: The Security of SHAvite-3		
Pp. 22, Sect. 6.1.1, the "slide attacks" bullet	The only problematic case is when $\#bits = 0$, which happens only during initializations (where the adversary has almost no control over the inputs), and during the processing of a full padding block (again, where the adversary has no control over the inputs).	As observed in [57, 61], without the complementation of the counter, the case $\#bits = 0$ (which happens only during initializations and the processing of a full padding block, i.e., where the adversary has almost no control over the inputs) the block ciphers posses some slid properties and fix points (for message block $m = 0$ and salt $salt = 5252...52_x$). In [22] several other special relationships for $\#bits = 0$, $m = 0$, and $salt = 5252...52_x$ are discussed. Again, this issue is solved by the tweak.
Appendix A: Test Vectors		
		The test vectors were updated.

2 Improved Technical Work

2.1 New Security Analysis

Following new results on hash function cryptanalysis (namely paper [2]), we re-evaluated the security of HAIFA as a mode of iteration.

Location	Original Text	New Text
Sect. 3: HAsh Iterative FrAmework		
Pp. 4, first paragraph	and chosen target preimage attacks [36].	chosen target preimage attacks [47], and Trojan message attacks [2].
Pp. 4, second paragraph	and the chosen-target preimage attacks of [1, 25, 36, 37].	the chosen-target preimage attacks, and the Trojan message attacks of [1, 2, 32, 47, 48].
Pp. 6, Sect. 3.4.3	Added at the end of the paragraph:	Hence, a HAIFA hash function can be distinguished after q queries to the compression function with probability at most $O(q^2/2^{m_c})$ (or if $m_c = m$ — with probability at most $O(q^2/2^m)$).
Pp. 8, Sect. 3.4.4	Added at the end of the considered attacks:	**Trojan Message Attacks** In this attack the adversary introduces a malicious suffix which allows efficiently finding second preimages to a restricted set of messages [2]. The two variants of the attack are using collisions in the compression function. Once the actual compression function is unknown in advance due to the salt, this attack is rendered impossible. By fixing the salt length to more than $m/2$, the attack becomes slower than second preimage attacks, even if the adversary is allowed to supply a different suffix for every possible salt.
Sect. 5: Design Criteria and Rationale		
Pp. 18, Sect. 5.2	Added a bullet concerning why Enveloped Merkle-Damgård was not selected as the mode of iteration of SHAvite-3:	Enveloped Merkle-Damgård — While the enveloped Merkle-Damgård mode offers the preservation of the pseudo random properties of the compression function, it does not offer full second preimage resistance for long messages and is not secure against the herding attack. Hence, we decided to avoid the use of this mode.
Pp. 18, Sect. 5.2, Tree hashes	hash is presented in [1].	hash is presented in [2].[3]

Location	Original Text	New Text
– ” –	Added footnote 3:	The attack of [2] is applicable whenever all compression functions used in the tree are the same. This is not true when the there is an additional input to the compression function which changes in different calls, independent of the message.
Pp. 18, Sect. 5.3, second para.	in order to protect against herding attacks	in order to protect against herding attacks and Trojan message attacks
Sect. 6: The Security of SHAvite-3		
Pp. 22, Sect. 6.1.1, the ”Algebraic Approaches” bullet	Added at the end of bullet:	We also note that this seems to render cube attacks [33] on the full cipher unsuccessful.
Sect. 8: Performance		
Pp. 27, Sect. 8, second para.	Added a new paragraph:	Finally, we note that the tweak is not expected to invalidate most of the previous work done on measuring and implementing SHAvite-3. This follows the fact that the tweak is composed of negating four words, which can be done in software in four 32-bit operations or in very few hardware gates.

2.2 New Performance Analysis

The performance figures for SHAvite were extended and improved. This follows the following reasons:

1. The use of a better compiler with more optimization flags.
2. The report on the efforts of the eBASH project.
3. The results of [10] concerning the new AES instruction (AES-NI extension).
4. The improved small footprint implementation of AES ([39] with comparison to the earlier [27]).

Location	Original Text	New Text
Sect. 1: Introduction		
Pp. 2, second para.	achieves for 256-bit digests a speed of 35.3 cycles per byte on a 32-bit machine and of 26.7 cycles per byte on a 64-bit machine. For 512-bit digests, SHAvite-3 achieves speeds of 58.4 cycles per byte on a 32-bit machine, and 38.2 cycles per byte on a 64-bit machine.	achieves for 256-bit digests a speed of 32.83 cycles per byte on a 32-bit machine and of 25.13 cycles per byte on a 64-bit machine. For 512-bit digests, SHAvite-3 achieves speeds of 55.90 cycles per byte on a 32-bit machine, and 35.86 cycles per byte on a 64-bit machine. As shown in [10], on future Intel CPUs which support the AES-NI instruction set, speeds of 5.6 and 5.5 cycles per byte, respectively, are attainable. These speeds suggest that on future platforms, SHAvite-3 is the fastest remaining candidate.
Sect. 7: HAIFA-MAC and SHAvite-3-MAC		
Pp. 26, Sect. 7, one before last para.	Added at the end of the paragraph:	In Table 4 we compare the number of compression function calls when using SHA-256, HMAC-SHA-256, SHAvite-3, and Shavite-3-MAC (when they are used to produce a 256-bit digest/tag).
same	Added table 4	
Sect. 8: Performance		
Pp. 26, Sect. 8, first para.	Added at the end of the paragraph:	Moreover, it seems that on Intel CPUs with the new AES-NI instruction, SHAvite-3 is going to be the fastest candidate [10].
Pp. 26, Sect. 8, second para.	ANSI-C code is 35.3 cycles per byte for 224-/256-bit digests, and 58.4 cycles per byte for 384-/512-bit digests on 32-bit Intel machines. On a 64-bit machine, the corresponding running times are 26.7 and 38.2 cycles per byte, respectively.	ANSI-C code is 32.83 cycles per byte for 224-/256-bit digests, and 55.90 cycles per byte for 384-/512-bit digests on 32-bit Intel machines. On a 64-bit machine, the corresponding running times are 25.13 and 35.86 cycles per byte, respectively.
Pp. 27, Sect. 8, second para.	Added at the end of the paragraph:	As shown in [10], the actual gains for SHAvite-3_{256} is expected to be about 77%, and for SHAvite-3_{512} is expected to be about 84%.
Pp. 28, Sect. 8.1.2, first para.	would improve our current speed of 35.3 cycles per byte	would improve our current speed of 32.83 cycles per byte
– " –	compiled with gcc 4.0.3).	compiled with gcc 4.4.1).

Location	Original Text	New Text
Pp. 28, Sect. 8.1.2, second para.	SHAvite-3_{512} has a running time of 55.0 cycles per byte	SHAvite-3_{512} has a running time of 55.90 cycles per byte
Pp. 28, Sect. 8.1.2, third paragraph	For comparison, on the same machine, which we obtained 35.3 cycles per byte for (not-well-optimized) SHAvite-3_{256}, the fastest SHA-1 implementation has a running time of 9.8 cycles per byte, SHA-256 had a running time of 28.8 cycles per byte, and SHA-512 had a running time of 77.8 cycles per byte. All measurements were done using the NESSIE test suite [44].	For comparison, on the same machine, which we obtained 32.83 cycles per byte for (not-well-optimized) SHAvite-3_{256}, the fastest SHA-1 implementation (obtained from the OpenSSL library) has a running time of 9.38 cycles per byte, SHA-256 had a running time of 27.29 cycles per byte, and SHA-512 had a running time of 78.38 cycles per byte. All measurements were done using the NESSIE test suite [58] using code from OpenSSL and internal NESSIE code.
Pp. 28, Sect. 8.1.2, third para.	Added a new paragraph:	In the eBASH project [35], the timings of various SHA-3 candidates (as well as other hash functions) on various platforms were measured. The measurements for SHAvite-3 were done on one variant of the code which we provided, and that was optimized to a specific type of a 32-bit machine without salt support (i.e., the fixed salt was used). The running times on x86 platforms (Intel and AMD) varies between 28.73 and 84.42 cycles/byte for SHAvite-3_{256} (the lower speeds are usually obtained on older machines or older compilers or both), validating our previous speed claims. For PowerPCs with 32-bit CPUs the speeds of SHAvite-3_{256} varied between 20.62 and 43.99 cycles per byte. For SHAvite-3_{512} the measured speeds on x86 platforms vary between 55.30 and 242.09 cycles per byte. For PowerPCs with 32-bit CPUs, the running times are between 32.00 and 184.78 cycles per byte.
Pp. 28, Sect. 8.1.3, third para.	has a running time of 26.7 cycles per byte	has a running time of 25.13 cycles per byte

Location	Original Text	New Text
– ” –	compiled with gcc 4.2.4).	compiled with gcc 4.4.1).
– ” –	The code of SHAvite-3_{512} has a running time of 38.2 cycles per byte. For comparison, on this machine, SHA-1 takes 9.5 cycles per byte, SHA-256 takes 25.3 cycles per byte, and SHA-512 takes 16.9 cycles per byte.	The code of SHAvite-3_{512} has a running time of 35.86 cycles per byte. For comparison, on this machine, SHA-1 takes 7.34 cycles per byte, SHA-256 takes 19.08 cycles per byte, and SHA-512 takes 14.71 cycles per byte.
– ” –	Added after the paragraph a new one:	The measurements of eBASH for 64-bit Intel and AMD platforms were between 22.79 and 61.24 cycles per byte for SHAvite-3_{256} and between 24.71 and 39.64 cycles per byte for PowerPC platforms. For SHAvite-3_{512} the measurements on Intel and AMD platforms were 40.28 and 255.10 cycles per byte. For PowerPCs, the corresponding range is 38.41 to 64.39 cycles per byte.
Pp. 29, Table 4	The table was updated to include the updated results as well as the independent results obtained by eBASH and [10].	
Pp. 29, Sect. 8.2, first para.	This would lead to a running time of less than 8 cycles per byte on such CPUs.	In [10] the untweaked version of SHAvite-3 was timed, and the outcome was 5.6 cycles/byte with the new AES-NI extension. We note that of all the second round candidates considered in [10], SHAvite-3 was the fastest. As the tweak is expected to have little effect on the time measures, we can safely estimate the running time of the tweaked version as having the same value, i.e., 5.6 cycles/byte.
Pp. 29, Sect. 8.2, second para.	AES round instructions themselves can be interleaved.	AES round instructions themselves can be interleaved as was shown in [10]. For the untweaked SHAvite-3_{512} the running time was measured to be 5.5 cycles/byte with the new AES-NI extension. As before, the tweak is not expected to affect the performance, and thus, we expect a similar speed for the tweaked version.

Location	Original Text	New Text
Pp. 29, Sect. 8.3.1, first para.	The smallest AES implementation in ASIC is reported in [27]. The suggested implementation uses about 3400 gates, and has a throughput of 9.9 Mbps in a 80 MHz maximal frequency (the implementation used a 0.35μ technology). Of the 3400 gates, about 60% (about 2040) are reported to store 256 bits of the internal state (a rate of about 8 gates per memory bit).	The smallest AES implementation in ASIC is reported in [39]. The suggested implementation uses about 3100 gates, and has a throughput of 121 Mbps in a 152 MHz maximal frequency (the implementation uses a 0.13μ technology). Of the 3100 gates, about 60% (i.e., 1860 gates) are reported to store 256 bits of the internal state (a rate of about 7.2 gates per memory bit).
Pp. 30, Sect. 8.3.2, second para.	and another 1360 gates for the AES core	and another 1240 gates for the AES core
– " –	a full implementation of SHAvite-3_{256} in about 10300 gates.	a full implementation of SHAvite-3_{256} in about 10100 gates.
Pp. 30, Sect. 8.3.2, third para.	The speed of this implementation is about 100 cycles for an AES round (at 80 MHz), which implies a speed of about 5200 cycles for an invocation of C_{256}, or a throughput of about 7.6 Mbps.	The speed of this implementation is about 16 cycles for an AES round (at 152 MHz), which implies a speed of about 840 cycles for an invocation of C_{256}, or a throughput of about 93.5 Mbps at 153 MHz clock rate.
Pp. 31, Sect. 8.3.3, first para.	Hence, the implementation is expected to use about 18500 gates, and achieve a speed of about 4.7 Mbps.	Hence, the implementation is expected to use about 18400 gates, and to achieve a speed of about 57.9 Mbps.
Pp. 31, Sect. 8.3.3, second para.	four AES round cores need to be used. This increases the circuit size to about 100,500 gates.	three AES round cores need to be used with some additional memory.[7] This increases the circuit size to about 81,000 gates.
– " –	Added footnote 7:	The three cores are used as follows: one in each $F^4(\cdot)$, and one for the message expansion. There is a requirement for some additional memory in the message expansion in this approach.

3 Editorial Changes

Location	Original Text	New Text
Abstract	which iterates a round function based on the AES round.	which iterates a round function based on the AES round function.
Sect. 2: AES and Some Mathematical Background		
Pp. 2, last para.	In order to explicitly the AES designers picked the following irreducible polynomial	The AES designers picked the following irreducible polynomial
Pp. 2, last para.	following the Federal Information Processing Standard 197	following Federal Information Processing Standard 197
Sect. 3: HAsh Iterative FrAmework		
Pp. 4, second para.	Under reasonable assumptions, it is claimed that HAIFA does preserve	Under reasonable assumptions, it is claimed that HAIFA preserves
Pp. 5, first para.	However, a careful application would ensure that the salt contains enough randomness to be unpredicted.	However, a careful application would ensure that the salt contains enough randomness to be unpredictable.
Pp. 6, Sect. 3.4.1, third para.	then the last blocks are necessarily different.	then the last compression function calls are necessarily different.
Pp. 6, Sect. 3.4.2, second para.	The reason for that is that the last block	The reason for this is that the last block
Pp. 6, Sect. 3.4.3, first para.	Among other things, this fact proves that HAIFA	Among other things, this proves that HAIFA
Pp. 7, Sect. 3.4.4, Kelsey and Schneier's attack	the cost of connecting it to the challenge message is like the cost	the cost of connecting it to the challenge message is equivalent to the cost
– " –	(and is 2^m).	(and is still 2^m).
Pp. 7, Sect. 3.4.4, the herding attack	is larger than a preimage attack and whose memory storage makes standard time-memory attacks more favorable.	is larger than a preimage attack and whose memory storage makes standard time-memory tradeoff attacks more favorable.
Pp. 8, Sect. 3.4.4, the second preimage attack based on herding	which is slightly slower than other techniques,	which is slightly slower than the one of [48],

Location	Original Text	New Text
Sect. 4: Specifications of SHAvite-3		
Pp. 9, Sect. 4.1.2, third para.	After that we repeat a process that generates	After that, we repeat a process that generates
Pp. 10, Figure 2	All the 0 values	were changed into 0^{128}.
Pp. 11, Figure 3	We note that the counters are XORed in different positions	We note that the counters are added in different positions
Pp. 13, Sect. 4.1.4, first para.	as well as 512-bit subkey $RK_{i,j} =$	as well as a 512-bit subkey $RK_{i,j} =$
Pp. 17, Sect. 4.3, first para.	(possibly in exchange for loss in security)	(possibly in exchange for a loss in security)
– " –	is expected to be slightly faster than for the general case).	is expected to be slightly faster than for the general case.
Sect. 5: Design Criteria and Rationale		
Pp. 17, Sect. 5.1, third para.	as the adversary can control the key (message).	as the adversary can control the key (through the message).
– " –	We therefore ensure that the best related-key differential would has as low	We therefore ensure that the best related-key differential would have as low
– " –	(it may be at most $2^{-m/2}$ for m-bit digest	(it may be at most $2^{-m/2}$ for an m-bit digest
Pp. 17, Sect. 5.1, fourth para.	with the "encryption" part of the block cipher,	with the "encryption" part of the block cipher (as happened in the recent attacks on AES [19–21]),
Pp. 19, Sect. 5.4, second para.	set of "common constants" (i.e., $\sqrt{2}, \varphi, \ldots$).	set of "common constants" (i.e., $\sqrt{2}, \phi, \ldots$).
Pp. 19, Sect. 5.4, fifth para.	The locations were chosen to be in the	For SHAvite-3_{256} the locations were chosen to be in the
Pp. 19, Sect. 5.4, last para.	(as the issue of possibly weak values is solved by	(as the issue of potentially weak values is solved by
Sect. 6: The Security of SHAvite-3		
Pp. 22, Sect. 6.1.1, second para.	suggest that boomerang attacks	suggests that boomerang attacks
Pp. 22, Sect. 6.1.1, impossible differential cryptanalysis bullet	is secure against impossible differential as well.	is secure against impossible differential cryptanalysis as well.

Location	Original Text	New Text
Pp. 22, Sect. 6.1.1, the "Algebraic Approaches" bullet	is still open.	is still open (see [18,42]).
Pp. 23, Sect. 6.1.2, third para.	at random assures that the attacker cannot select	at random assures that the adversary cannot select
Pp. 25, Sect. 6.3.1, second para.	we suggest new signature schemes to allow for a mechanism	we suggest that new signature schemes allow
Sect. 7: HAIFA-MAC and SHAvite-3-MAC		
Pp. 26, Sect. 7, second para.	Moreover, the different IV_m for different digest sizes	Moreover, the different IV_m's for different digest sizes
Pp. 26, Sect. 7, third para.	(and preferably related-key pseudorandom function).	(and preferably a related-key pseudorandom function).
Pp. 26, Sect. 7, fifth para.	Added at the end of the paragraph:	Of course, in this case the key used as salt is to be kept secret and never used publicly.
Sect. 8: Performance		
Pp. 28, Sect. 8.1.3, first para.	There are several approaches to implement AES	There are several approaches for implementing AES
– " –	larger number of registers and commands	larger number of registers and instructions
– " –	which were applied in [40, 41] to implement AES efficiently.	which were applied in [45, 53, 54] to implement AES efficiently.
Pp. 28, Sect. 8.1.3, second para.	it seems that the speed of AES on 64-bit machine	it seems that the speed of AES on 64-bit machines
Pp. 28, Sect. 8.2, first para.	to allow multiple calls for the command	to allow multiple calls for the instruction
Pp. 29, Sect. 8.2, first para.	With such a command,	With such an instruction,
Pp. 29, Sect. 8.2, second para.	where we expect even better use of the command,	where we expect even better use of the instruction,
Pp. 29, Sect. 8.3.1, second para.	using many pipelined application of AES.	using many pipelined applications of AES.

Location	Original Text	New Text
Pp. 30, Sect. 8.3.2, second para.	an implementation based on [27] would need another 128-bit of internal state	an implementation based on [39] would need another 128 bits of internal state
Pp. 30, Sect. 8.3.2, fourth para.	(each takes 22,850 gates)	(22,850 gates each)

4 Code Changes

Location	Original Text	New Text
SHAvite-3.c	local_bitcount+=BlockSizeB; The change follows a bit counter vs. byte counter issues	local_bitcount+=8*BlockSizeB;

SHAvite-3.c	Compress256(block,result, state->bitcount, state->salt);
	
	Dealing with messages whose length is divisable by the block size
		if ((state->bitcount % state->BlockSize)==0)
		Compress256(block,result, 0x0ULL, state->salt);
		else
		Compress256(block,result,state->bitcount, state->salt);

SHAvite-3.c	Compress512(block,result, state->bitcount, state->salt);
	
	Dealing with messages whose length is divisable by the block size
		if ((state->bitcount % state->BlockSize)==0)
		Compress512(block,result, 0x0ULL, state->salt);
		else
		Compress512(block,result,state->bitcount, state->salt);

SHAvite3-256.h
SHAvite3-256.h
SHAvite3-256.h
SHAvite3-256.h
	rk[17] ^ = counter[1];
	rk[58] ^ = counter[0];
	rk[87] ^ = counter[0];
	rk[127] ^ = counter[1];
	This is the tweak for SHAvite-3_{256}
		rk[17] ^ = ∼counter[1];
		rk[58] ^ = ∼counter[0];
		rk[87] ^ = ∼counter[0];
		rk[127] ^ = ∼counter[1];

SHAvite3-512.h
SHAvite3-512.h
SHAvite3-512.h
SHAvite3-512.h
	rk[35] ^ = counter[3];
	rk[167] ^ = counter[0];
	rk[443] ^ = counter[2];
	rk[319] ^ = counter[1];
	This is the tweak for SHAvite-3_{512}
		rk[35] ^ = ∼counter[3];
		rk[167] ^ = ∼counter[0];
		rk[443] ^ = ∼counter[2];
		rk[319] ^ = ∼counter[1];

SHAvite-3.c	Added comment (twice):	/* The following line is due to the required API of 64-bit message */ /* length, while the hash function deals with 128-bit lengths */

Location	Original Text	New Text
SHAvite-3	Added code (with no effect) (improves readability)	memset(block+BlockSizeB-10,0,8);
SHAvite-3	Compress512(block,result,0x0UL, state->salt);	Compress512(block,result,0x0ULL, state->salt);
all files	Added to header	/* Tweaked version (21.Sep.2009) */

We note that also the optimized code has changed (and is no longer equal to the reference code).

5 Improved Bibliography

Added the following papers to the bibliography:

2. Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, John Kelsey, *Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgård*, presented at Selected Areas in Cryptography 2009.
10. Ryad Benadjila, Olivier Billet, Shay Gueron, Matthew J.B. Robshaw, *The Intel AES Instructions Set and the SHA-3 Candidates*, accepted to ASIACRYPT 2009, available online at `http://crypto.rd.francetelecom.com/sha3/AES/paper/`.
19. Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, Adi Shamir, *Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds*, IACR ePrint report 2009/374.
20. Alex Biryukov, Dmitry Khovratovich, Ivica Nikolic, *Distinguisher and Related-Key Attack on the Full AES-256*, Advances in Cryptology, proceedings of CRYPTO 2009, Lecture Notes in Computer Science 5677, pp. 231–249, Springer-Verlag, 2009.
21. Alex Biryukov, Dmitry Khovratovich, *Related-key Cryptanalysis of the Full AES-192 and AES-256*, accepted to ASIACRYPT 2009, available online at `http://eprint.iacr.org/2009/317.pdf`.
22. Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Gaëtan Leurent, *New Self-Similarity Attack*, preprint, September 2009.
24. Carlos Cid, Gaëten Leurent, *An Analysis of the XSL Algorithm*, Advances in Cryptology, proceedings of ASIACRYPT 2005, Lecture Notes in Computer Science 3788, pp. 333–352, Springer-Verlag, 2005.
33. Itai Dinur, Adi Shamir, *Cube Attacks on Tweakable Black Box Polynomials*, Advances in Cryptology, proceedings of EUROCRYPT 2009, Lecture Notes in Computer Science 5479, pp. 278–299, Springer-Verlag, 2009.
35. ECRYPT, *ECRYPT Benchmarking of All Submitted Hashes*, available online at `http://bench.cr.yp.to/results-hash.html`.
39. Panu Hämäläinen, Timo Alho, Marko Hännikäinen, Timo D. Hämäläinen, *Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core*, Ninth Euromicro Conference on Digital System Design: Architectures, Methods and Tools, IEEE Computer Society, 2006.
45. Emilia Käsper, Peter Schwabe, *Faster and Timing-Attack Resistant AES-GCM*, proceedings of Cryptographic Hardware and Embedded Systems — CHES 2009, Lecture Notes in Computer Science 5747, pp. 1–17, Springer-Verlag, 2009.
50. Lars R. Knudsen, Vincent Rijmen, *Known-Key Distinguishers for Some Block Ciphers*, Advances in Cryptology, proceedings of ASIACRYPT 2007, Lecture Notes in Computer Science 4833, pp. 315–324, Springer-Verlag, 2007.

51. Chu-Wee Lim, Khoongming Khoo, *An Analysis of XSL Applied to BES*, proceedings of Fast Software Encryption 2007, Lecture Notes in Computer Science 4593, pp. 242–253, Springer-Verlag, 2007.
57. Mridul Nandi, Souradyuti Paul, *OFFICIAL COMMENT: SHAvite-3*, 2009, available online at `http://ehash.iaik.tugraz.at/uploads/5/5c/NandiP-SHAvite-3.txt`.
61. Thomas Peyrin, *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`.

Also, the bibliographic data of [9] were updated (following its publication in the proceedings of INDOCRYPT 2008).

6 Miscellaneous

Location	Original Text	New Text
Sect. 8: Performance		
Pp. 27, Sect. 8, second para.	will be added to the Intel CPUs (expected in the second quarter of 2009),	will be added to the Intel CPUs (expected in the last quarter of 2009 or the first quarter of 2010) and to the AMD CPUs (expected in 2011),
Pp. 28, Sect. 8.2, first para.	It is evident that adding the set of AES commands to Intel CPUs is expected	It is evident that adding the set of AES instructions to Intel and AMD CPUs is expected
– " –	The expected latency of this command is 6,	The expected latency of the new Intel instruction is 6,
Sect. 9: Summary		
		Added acknowledgments