A Short Note on AES-Inspired Hashes

Tor E. Bjørstad

University of Bergen, Norway
Email : tor.bjorstad@ii.uib.no

1 Introduction

This paper gives a rough classification of various AES-based hash algorithms that have been accepted for Round 1 of the NIST Hash Function Competition. We compare the different algorithms based on the number of "AES-like" rounds performed by a single invocation of the compression function, versus the size of the input block. This is not particularly rigorous or scientific way of comparing hash algorithms, but as we shall see it may still yield somewhat interesting results.

With the exception of Sarmal and Whirlpool, all the listed algorithms use the AES S-box as their only source of nonlinearity. How they use it is a different matter. What all the algorithms do have in common, is that they use the 8 bit S-box together with a MDS matrix, in a construction where `SubBytes()` and `MixColumns()` are applied after one another on some internal data structure. The actual MDS matrices used are frequently different from the one in AES; for instance, Fugue uses a 16×16 matrix.

For the sake of comparison, the number of of rounds has been normalised to correspond to those of AES-128. This is not entirely fair, but it would be hard to make any comparison between the internals of such wildly different hash algorithms without making some (over)simplifications.

The reader is invited to take note of 3 main observations. First, there is a very wide spread between the number of "rounds" used by the different algorithms – but a strong tendency of clustering. Secondly, different strategies are used in moving from 256 to 512 bit digests, with the ratio between the number of equivalent rounds per byte ranging between 1 and 2. Finally, there is a fair amount of correlation between this parameter and the reported software speeds on the NIST reference platform.

2 The Algorithms

We consider 12 different algorithms which were accepted for the first round of the NIST Hash Function Competition: ARIRANG [4], Cheetah [10], ECHO [2], Fugue [7], Grøstl [6], LANE [9], Lesamnta [8], LUX [11], Sarmal [13], SHAMATA [1], SHAvite-3 [3], and Twister [5]. Whirlpool-512 [12] and AES itself are used as familiar reference points. Any mistakes in this list are due to my own misreading of the specs; please email corrections as needed.

– AES has a 16 byte message block, and runs for 10/14 rounds.
– ARIRANG has a 64/128 byte message block, and consists of a 40-round unbalanced Feistel network. In each round, the the non-linear function G is called twice. The function performs `SubBytes()` and `MixColumns()` on a single word of 4/8 bytes.
– Cheetah has a 128 byte message block. The key schedule uses 3/5 AES-like rounds to expand the input, viewing it as an 8×16 byte array. The 32/64 byte chaining value is then treated as a 4×8 byte array and transformed for 16/12 AES-like rounds.
– ECHO has a 192/128 byte message block. The compression function runs for 8/10 rounds on the full 256 byte state. In each "big" round, 2 AES rounds are performed on each 16-byte substate.
– Fugue has a 4 byte message block. The nonlinear "Super Mix"-function takes 16 bytes from the state, and applies `SubBytes()` followed by a very large (16×16) MDS matrix. This is done 2/4 times.
– Grøstl has a 64/128 byte message block. The compression function consists of two AES-like permutations on 64/128 bytes, which use 10/14 rounds.
– LANE has a 64/128 byte message block. Message and IV is expanded into six lanes of 32/64 bytes. Each lane is processed using for 6/8 rounds, using 2/4 AES rounds at each step. The expanded state is then merged to 2 lanes, which are transformed for another 3/4 rounds.
– Lesamnta has a 32/64 byte message block. The compression function is an unbalanced Feistel network that runs for 32 steps. In each step, the `SubState()` function applies 4 AES-like rounds to an 8/16 byte block. The key schedule also uses 32 steps, where each step uses a similar round transformation.
– LUX has a 4/8 byte message block. The size of the "core" is 32/64 bytes. For every message block, an AES-like round is performed once on the entire core.
– Sarmal has a 128 byte message block which is processed by two parallel lanes. The number of rounds in each lane is 16/20. In each round, two nonlinear g-functions are evaluated in each lane. The g-function is similar to ARIRANG-512, applying `SubBytes()` and an 8×8 MDS matrix to a single 8 byte column. The S-box is different from AES, and Sarmal also contains additions and subtractions in $\mathrm{GF}(2^{64})$.
– SHAMATA has a 16 byte input block. The registers are clocked twice for each input, and every clocking uses 1/2 AES rounds.
– SHAvite-3 has a 64/128 byte input block. The underlying Feistel cipher runs for 12/14 steps, and in each step the non-linear function uses 3/8 AES rounds. The key schedule uses 16/24 additional AES rounds.
– Twister has a 64 byte input block. The compression function uses 9/10 "minirounds". Each miniround consists of an AES-like transformation of the 64 byte state.
– Whirlpool has a 64 byte input block. It uses 10 AES-like rounds on a 64 byte state. The same round function is also used by the key schedule. The S-box of Whirlpool is different from AES.

3 Results

The tables below summarise the resulting "equivalent round per byte" count, sorted from low to high. Performance data is taken from the submission documents, the algorithm home pages and slides from the NIST Conference, and may not be up to date. To the best of my knowledge, all the AES-based algorithms use table-based (and not bitsliced) implementations.

Algorithm (256 bit)	Rounds / byte	Perf. (cpb)	Comment
SHAMATA	2	8.0	Broken. Practical collisions.
Sarmal	4	9.4	Alternate S-box, $\boxplus, \boxminus$.
ARIRANG	5	15.0	
Cheetah	7	9.3	
LUX	8	10.2	SHA-3 Zoo: Orange.
Fugue	8	28.0	
Twister	9	15.8	
AES	10	10.6	Table-based impl. by Bernstein, Schwabe.
		7.8	Bitsliced impl. by Käsper, Schwabe.
SHAvite-3	13	26.7	
Grøstl	20	21.3	
LANE	21	25.7	
ECHO	21.3	28.5	
Lesamnta	40	52.7	

Fig. 1. Equiv. AES rounds for 256-bit hashes. AES included for comparison.

Algorithm (512 bit)	Rounds / byte	Perf. (cpb)	Ratio 512/256	Comment
SHAMATA	4	11.0	2	SHA-3 Zoo: Orange.
Sarmal	5	10.9	1.25	Alternate S-box, $\boxplus, \boxminus$.
ARIRANG	5	11.3	1	
LUX	8	9.5	1	SHA-3 Zoo: Orange.
Twister	10	17.5	1.111	SHA-3 Zoo: Orange.
Cheetah	11	13.6	1.571	
AES-256	14	?	1.4	
Fugue	16	56.0	2	
SHAvite-3	17	38.2	1.308	
Whirlpool	20	30?	-	Not a SHA-3 candidate (FSB).
Grøstl	28	29.8	1.4	
LANE	28	?	1.333	
ECHO	40	53.5	1.875	
Lesamnta	40	51.2	1	

Fig. 2. Equiv. AES rounds for 512-bit hashes. AES and Whirlpool for comparison.

References

1. A. Atalay, O. Kara, F. Karakoc, and C. Manap. SHAMATA hash function algorithm specifications. Submission to NIST, 2008.
2. R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, and Y. Seurin. SHA-3 proposal: ECHO. Submission to NIST, 2008. http://crypto.rd.francetelecom.com/echo/.
3. E. Biham and O. Dunkelman. The SHAvite-3 hash function. Submission to NIST, 2008. http://www.cs.technion.ac.il/~orrd/SHAvite-3/.
4. D. Chang, S. Hong, C. Kang, J. Kang, J. Kim, C. Lee, J. Lee, J. Lee, S. Lee, Y. Lee, J. Lim, and J. Sung. ARIRANG. Submission to NIST, 2008.
5. E. Fleischmann, C. Forler, and M. Gorski. The Twister hash function family. Submission to NIST, 2008. http://www.twister-hash.com/.
6. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST, 2008. http://www.groestl.info/.
7. S. Halevi, W. E. Hall, and C. S. Jutla. The hash function Fugue. Submission to NIST, 2008. http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html.
8. S. Hirose, H. Kuwakado, and H. Yoshida. SHA-3 proposal: Lesamnta. Submission to NIST, 2008.
9. S. Indesteege, E. Andreeva, C. D. Canniére, O. Dunkelman, E. Käsper, S. Nikova, B. Preneel, and E. Tischhauser. The LANE hash function. Submission to NIST, 2008. http://www.cosic.esat.kuleuven.be/lane/.
10. D. Khovratovich, A. Biryukov, and I. Nikolić. The hash function Cheetah: Specification and supporting documentation. Submission to NIST, 2008. http://cryptolux.org/Cheetah.
11. I. Nikolić, A. Biryukov, and D. Khovratovich. Hash family LUX – algorithm specifications and supporting documentation. Submission to NIST, 2008. http://cryptolux.org/LUX.
12. V. Rijmen and P. S. L. M. Barreto. The Whirlpool hash function. Standardised in ISO/IEC 10118-3:2004, 2004. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html.
13. K. Varıcı, O. Özen, and Ç. Kocair. Sarmal: SHA-3 proposal. Submission to NIST, 2008.