Analysis of the Hash Function – Modified Grøstl

Gurpreet Kaur
Defence Institute of Advance Technology (DU), Pune, INDIA

Vidyavati S Nayak
Defence Institute of Advance Technology (DU), Pune, INDIA

Dhananjoy Dey
Scientific Analysis Group, DRDO, Metcalfe House, Delhi, INDIA

SK Pal
Scientific Analysis Group, DRDO, Metcalfe House, Delhi, INDIA

ABSTRACT
Modified Grøstl - mGrøstl hash function was recently proposed by the authors as an alternative of one of the five finalist of SHA-3 competition namely Grøstl-256. This research paper presents the detailed analysis of algorithm along with performance evaluation of mGrøstl in eBASH project. According to the analysis, paper points out that different performance can be gained by adapting different platform.

Keywords
eBASH, Entropy Test, Grøstl, Hash Function, mGrøstl, SHA3 Competition.

1. INTRODUCTION
Hash functions play vital role in cryptology. A hash function, *h*, is an algorithm which processes an arbitrary length message into a fixed length hash code. Hash functions are used in different cryptographic applications like digital signatures, password protection schemes, and e-government. A hash function must provide different security properties depending on the security requirements of the application. It takes input of variable size and produces an output of fixed size, e.g., 160 bits for the most commonly used hash function SHA-1.

Hash functions should be very efficient and the security requirements for hash functions are usually described as follows [1].

• *Preimage Resistance*: For any given output *y*, it is computationally *infeasible* to construct an input *x* which hashes to *y*.

• *Second Preimage Resistance*: This property, also called *weak collision resistance*, specifies that for any given input *x* and output $h(x) = y$, it is computationally *infeasible* to construct a second input $x' \neq x$ which hashes to the same hash value *y*.

• *(Strong) Collision Resistance*: It is computationally *infeasible* to construct two distinct inputs which hash to the same value.

These three properties make the cryptographic one-way hash function suitable for achieving many security goals including authenticity, digital signatures, digital time stamping and entity authentication. Nowadays, the most commonly used dedicated cryptographic hash functions are SHA-1 and SHA-2. Since collisions on standard hash functions were reported in 2004 [2], improvements to design hash algorithms as well as the methods of attacking hash functions have progressed at a similar, rapid pace.

For this reason, NIST announced the SHA-3 competition [3] in Nov 2007. In SHA-3 hash function competition the five finalists selected in final round on 09 Dec 2010 are Blake [4], Grøstl [5], JH [6], Keccak [7] and Skein [8].

1.1 eBASH
Several ongoing projects are evaluating the efficiency of the SHA-3 candidates. Recently, D. J. Bernstein and T. Lange have launched the project eBASH (ECRYPT Benchmarking of All Submitted Hashes) [9] which is a project to measure the performance of hash functions. Anyone can submit his/her design to evaluate the efficiency by an independent third party. Time refers to time on real computers: time on an Intel Core 2 Quad, time on an AMD Athlon 64 X2, time on an IBM PowerPC G5 970, etc.

eBASH measures efficiency of each hash function on a wide variety of computers, ensuring direct comparability of all systems on whichever computers are of interest to the users.

1.2 Grøstl overview
Grøstl [5] is one of the five SHA-3 finalists and is an iterated hash function with a compression function built from two fixed, large, distinct permutations. The design of Grøstl is based on principles very different from those used in the SHA-family. The two permutations are constructed using the wide pipe design strategy, which makes it possible to give strong statements about the resistance of Grøstl against large classes of cryptanalytic attacks.

This paper presents the analysis of Modified Grøstl algorithm in terms of efficiency, entropy test, avalanche effect along with performance evaluation in eBASH project. This made to the conclusion, which shows that Modified Grøstl is faster when compared with Grøstl[1].

The organization of the paper is as follows. Section 2 presents the brief description of Modified Grøstl hash function, In Section 3 we discuss the security analysis using various tests conducted on the Modified Grøstl hash function followed by Section 4, presents the performance analysis along with the comparison with original Grøstl algorithm

2. Modified Grøstl
mGrøstl[2] - Modified Grøstl-256 hash function was recently proposed and developed by the authors and is under publication in [10] as an alternative to one of the five finalists SHA3 candidate, viz., Grøstl . For the completeness of this paper the brief description of the mGrøstl is presented below.

mGrøstl can take arbitrary length (< 2641024-bit block) of input and gives 256 bits output. We have modified the hash function construction, the padding procedure and the

[1] From now onwards we will mention Grostl-256 as Grøstl.
[2] We named Modified Grostl-256 as mGrøstl.

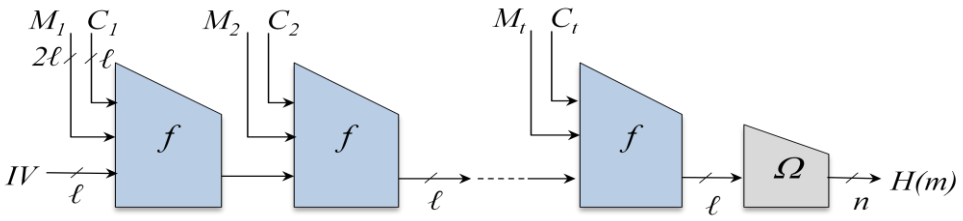

Fig 1: The Modified Grøstl Construction

compression function in mGrøstl. The three modifications details have discussed below.

2.1 Construction of Hash Function

In mGrøstl, we have designed the construction for 2ℓ bit message block. So the message M is first padded with the following method and split into 2ℓ-bit (1024 bit) message blocks $M_1 \ldots M_t$, and each message block is processed sequentially. The mGrøstl hash function iterates the compression function f as follows:

$$H_i \leftarrow f(H_{i-1}, M_i, C_i) \text{ for } 1 \le i \le t.$$

It maps the ℓ-bit previous Hash value $H_{i-1}$where $H_0=IV$ (Initial Vector), 2ℓ- bit message block M_i, and ℓ-bit counter value C_i where ℓ is defined to be 512 for mGrøstl-256 bit output as shown in Figure 1. The padding procedure is discussed below.

2.1.1 Padding

The hash value of a message M of length L=1024 × (t-1) + 8r bits can be computed in the following manner:

The padding procedure is shown in Figure 2. First append 1 to the end of the message M. Let k be the number of zeros added for padding. 7-bit representation of r bytes is appended to the end of k zeros and at the last the 64-bit representation of total number of blocks t is placed. k is the smallest non-negative integer satisfying the following condition:

$$8r + 1 + k + 7 + 64 \equiv 0 \ mod\ 1024$$

$$\text{i.e., } k + 8r \equiv 952 \ mod\ 1024$$

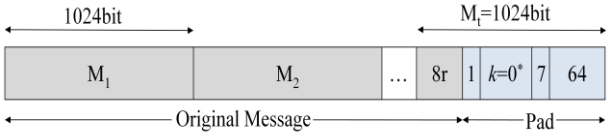

Fig 2: Padding Procedure

2.2 The Compression Function

The compression function f is based on two underlying ℓ-bit permutations P and Q [5] which are same as original Grøstl. The function f for mGrøstl is defined as follows:

For every message block M_i, is divided into two equal part, i.e., $M_i = L_i \| R_i$. Transform the message block, Previous Hash function H_{i-1} and the counter value C_i in the following way:

$$Ptmp \leftarrow \mathrm{P}(L_i \oplus C_i) .$$

$$Qtmp \leftarrow \mathrm{Q}(R_i \oplus H_{i-1}) .$$

Where counter $C_i = i \bmod 2^{64}$ for $1 \le i \le t$ which increments in next iteration. *Ptmp*, *Qtmp* are temporary 512-bit representation used for storing the intermediate values. Further the temporary values are transformed to get the next chaining variable H_i as follows.

$$Ptmp \leftarrow \mathrm{P}(Ptmp \oplus Qtmp) .$$

$$H_i \leftarrow Ptmp \oplus Qtmp \oplus H_{i-1}.$$

The construction of compression function f for mGrøstl is as shown in Figure 3.

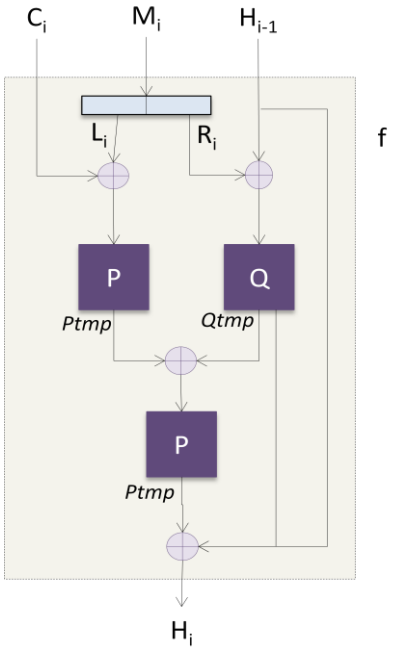

Fig 3: The compression function *f* for mGrøstl

3. EXPERIMENTAL RESULTS

3.1 Evaluation for Hash Function Speed

The metric used for evaluating the hash function speed was trivially the execution time of a single hash computation, averaged over ten numbers of repetitions. A performance comparison of the both hash functions considered for message digest generation is reported in Table 1 done on a Sony notebook with a Intel Core i3 with chipset M370 @ 2.40Gz processor with 3 GB RAM running Linux OS is used.

Table 1: Average execution time comparisons

File Size (in MB)	Grøstl (in ms.)	Modified Grøstl (in ms.)
10	894.44	671.06
12.6	1131.61	855.42
16.5	1447.14	1094.79
19.1	1605.09	1243.31
28.8	2318.69	1839.46

Here we observed that mGrøstl is 1.27 times faster than Grøstl. The performance chart comparison between original Grøstl and mGrøstl as shown in Figure 4.

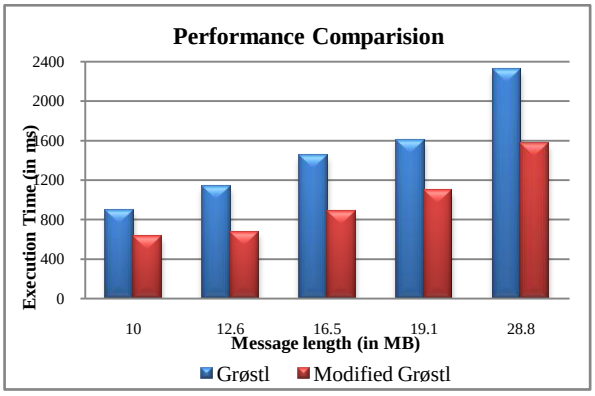

Fig 4: Performance comparison

3.2 Evaluation of the mGrøstl as a Hashing Algorithm

To evaluate the quality of a hash algorithm; the original Grøstl and the mGrøstl are evaluated against the following metrics [11].

1. Entropy test,
2. Bit Variances test

3.2.1 Entropy Test:

Entropy measures the average amount of information content of a message and gives maximum result when it equals the total number of bits in the message. Since, it is infeasible to calculate the entropy of the message digest, an approximate method [12] is used.

Approximate Entropy Assessment Method: Let the message digest divided into block size of 1 byte. By taking all possible combinations of byte pairs, we generated a set of 16 bit numbers (0-65535) for each message digest. For a various message digests if the frequencies of occurrences of these numbers (0-65535) are equal, then the approximate entropy for the 16 bit sub-blocks of the message digest is 16.

For the *entropy test*, all possible combinations of 8-bit numbers from each 256-bit message digest, (=32*(32-1)) are taken to form 16 bit numbers. The test is carried out for 200000 messages. Thus, there are 200000*32*(32-1) 16-bit number occurrences in the digest pool.

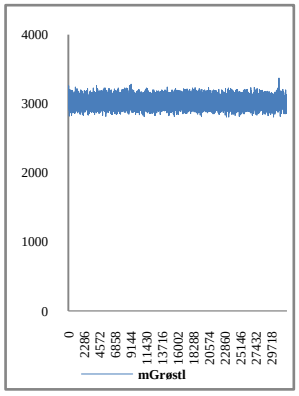

Fig 5: Result for mGrøstl

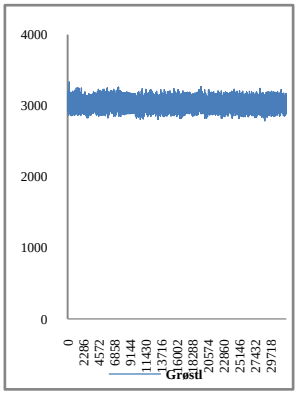

Fig 6: Result for Grøstl

The Figure 5 and Figure 6 show the entropy results for both Algorithms that describe the average occurrence frequencies of all the numbers (0-65535) are almost equal i.e. 3027. The approximate entropy for 16-bit sub block:

The original Grøstl = 15.99945 and

The Modified Grøstl = 15.99948.

So the approximate entropy for the 16 bit sub blocks of the message digest is almost 16.

3.2.2 The Bit-Variance Test

The bit variance test actually measures the uniformity of each bit of the digest. Since it is computationally difficult to consider all input message bit changes, we have evaluated the results for only up to 1024 files, viz. M, M_1, M_2, ..., M_{1024} which we have generated for conducting avalanche effect (discussed in next section), Finally from all the digests produced, the probability (P_i) for each digest bit to take on the value of 1 and 0 is measured. If $P_i(1) = P_i(0) = 1/2$ for all digest bits i ($i \leq 1 \leq n$) where n is the digest length, then the one way hash function under consideration has attained maximum performance in terms of the bit variance test.

We have performed the test and evaluated the results for 1024 files [11] and found the following results:

Mean frequency of 1's (expected) = 512.50

Mean frequency of 1's (calculated) = 510.87

Plotting the probability (Figure 7) of each of the bits (256-bit), we see that the average probability is approximately 0.50.

Thus, mGrøstl passes the bit variance test.

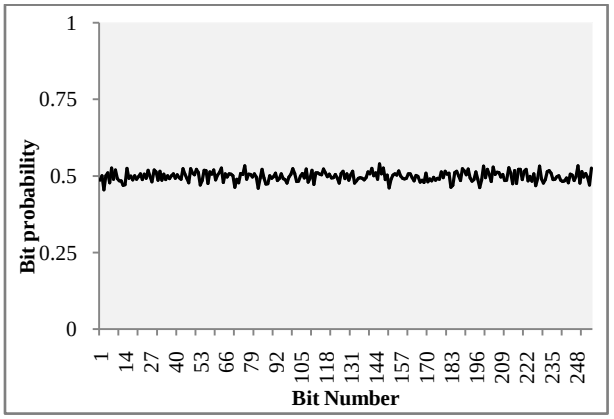

Fig 7: The probability of a bit position for mGrøstl

3.3 Avalanche Effect

The avalanche effect is evident if, when an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g., half the output bits flip).

Tool for the test: We have taken an input file M consisting of 1024 bits and computed $H(M)$. By changing the i^{th} bit of M, the files M_i have been generated, for $1 \leq i \leq 1024$. Thus hamming distance of each M_i from M is exactly one for $1 \leq i \leq 1024$. We then computed $H(M_i)$ for $1 \leq i \leq 1024$, computed the Hamming distances d_i between $H(M)$ and $H(M_i)$, for $1 \leq i \leq 1024$, i.e., number of ones for $H(M) \oplus H(M_i)$. The Table 2 shows the max, min, mode and the mean values of the above distances. The normal distribution of Hamming distances for mGrøstl is shown in Figure 8.

To satisfy strict avalanche criterion, each d_i should be 128 for $1 \leq i \leq 1024$. But we have found (Table 2) that d_i's were lying between 102 and 153 for the above files and in most of the cases $d_i = 128$. The observed deviation is acceptable so as to resist collision search using differential attack.

Table 2. Hamming Distances

	No. of ones for $d_i = H(M) \oplus H(M_i)$	
	Grøstl	**mGrøstl**
Maximum	152	153
Minimum	101	102
Mode	126	128
Mean	128.82	127.73

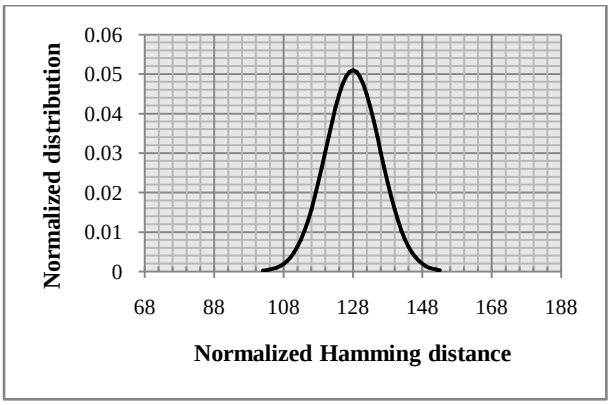

Fig 8: The normalized Hamming distances for mGrøstl

4. eBASH EXPERIMENTAL RESULTS

The three implementations of mgrostl256[3] were evaluated in eBASH project. The results in the form of graph are automatically posted at [13] whenever the eBASH performance results are updated. The snapshot of this graph is presented in the following subsection. The interested readers may check the eBASH web page [9] for the latest updates.

4.1 Implementation comparison: mgrostl256

We submitted three implementations of mgrostl256; those are 'Opt64', 'Opt32' and 'Ref' implementation into eBASH Project. We got the following graph as resultant Implementation comparison as shown in Figure 9.

However, the time to hash a message depends heavily on the message length and on the CPU used for hashing. A graph showing these dependencies is naturally two-dimensional: one axis shows the CPU, and one axis shows time. [14]

Vertical axis: architecture/microarchitecture/CPU/machine.

Horizontal axis: courier time; the time axis is cycles per byte

[3] Project eBASH named mGrøstl as mgrostl256

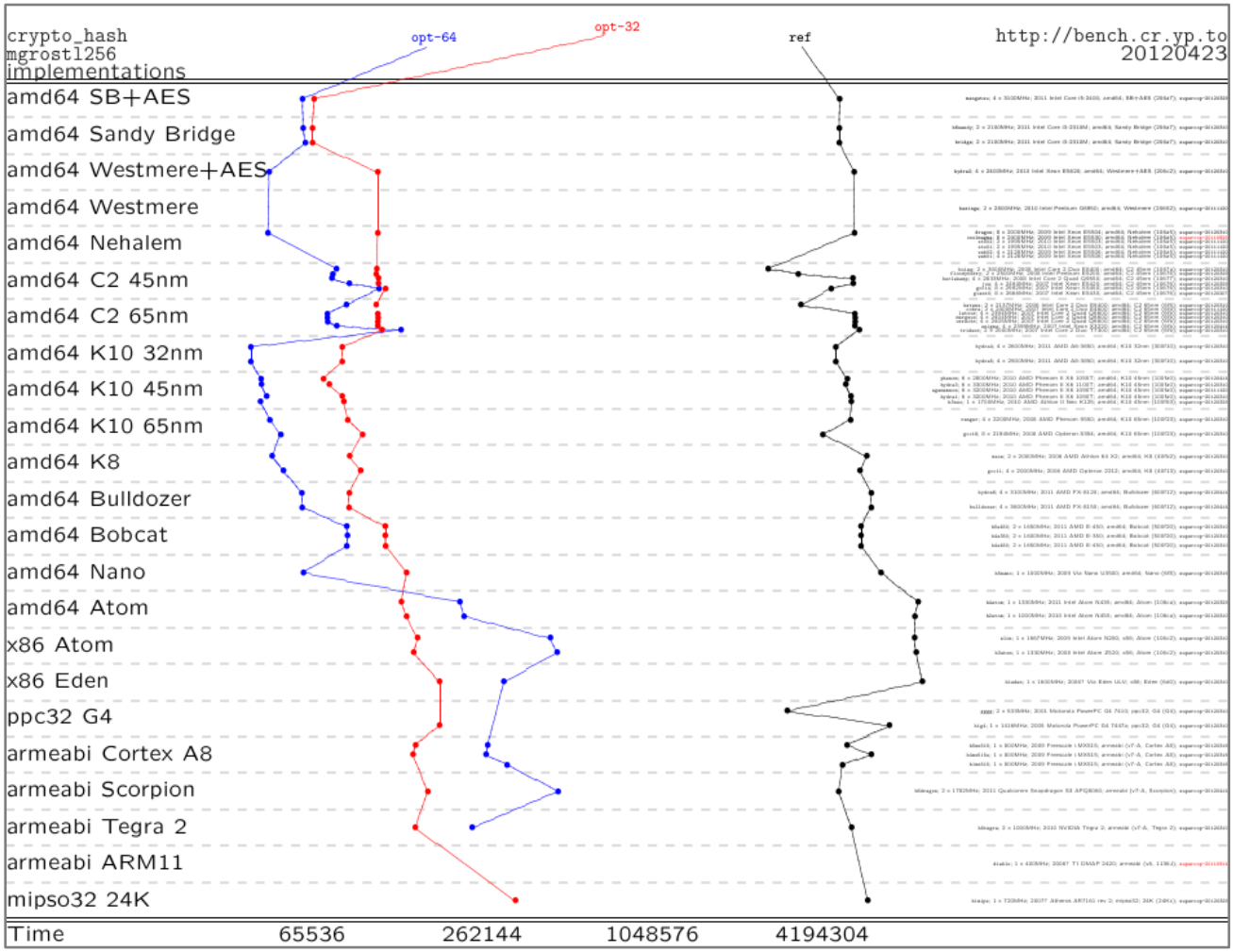

Fig 9: Implementation comparison: crypto_hash/mgrostl256

4.2 Measurements of mgrostl256, indexed by machine

The measurements of hash functions performance are given in the eBASH project. We further compiled the results as per the implementations Opt32 and Opt64 for both original Grøstl and mGrøstl. The computer name is linked to additional information about the implementations and compilers selected for benchmarking.

They have included four different architectures for compiling the results: armeabi, ppc32, x86, and amd64. As sometimes one micro-architecture includes useful additional instructions that will not work on other micro-architectures: for example, Sandy Bridge and the very new Ivy Bridge and Bulldozer all support AES instructions that are useful for Grøstl, while other amd64 micro-architectures do not support AES instructions.

The following graphs represent the Time vs. compiler chart for particular computer architecture. The graphs show that the mGrøstl is faster as compared to Grøstl.

4.2.1 Architecture: x86, Computer: hydra6

Implementation: Opt-32,

SUPERCOP version: 20120310

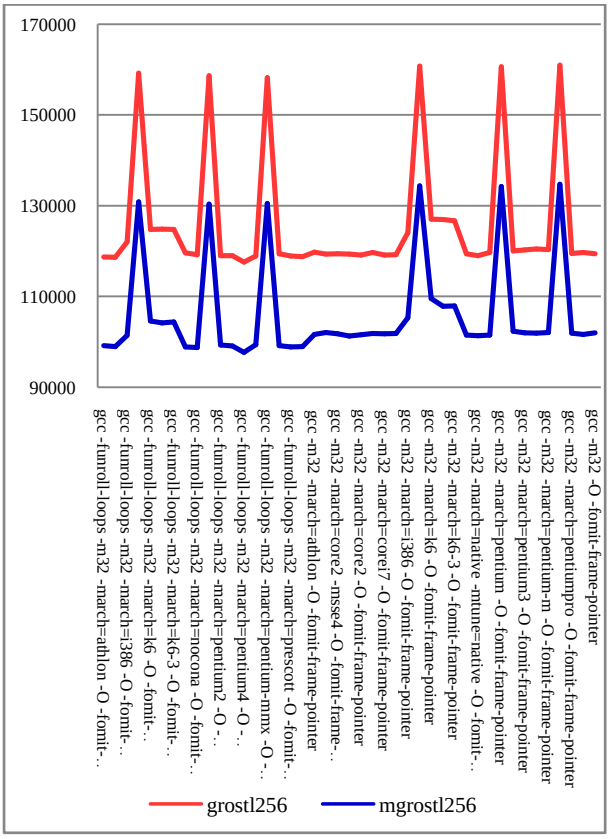

Fig. 10. Time vs. compiler chart for Architecture: x86

4.2.2 Architecture: amd64, Computer: h6sandy

Implementation: Opt 32,

SUPERCOP version: 20120310

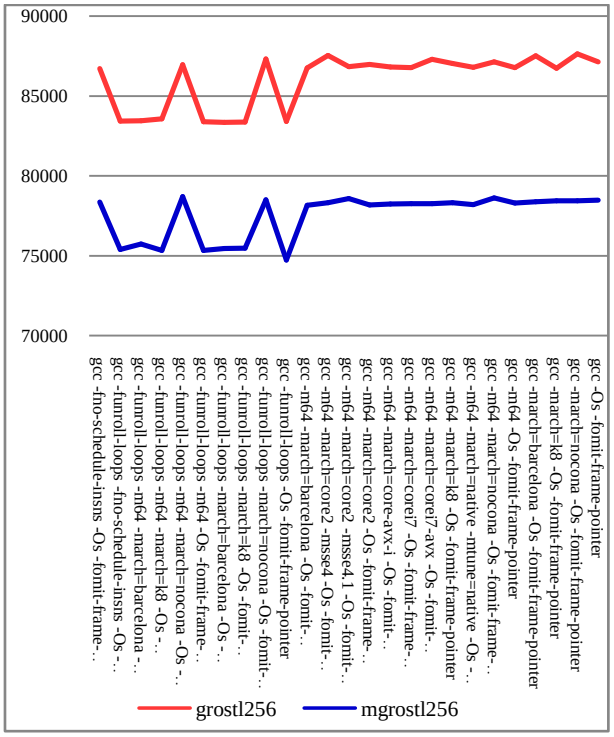

Fig 11: Time vs. compiler chart for Architecture: AMD64 Implementation: Opt 32

4.2.3 Architecture: amd64, Computer: bulldozer

Implementation: Opt 64 *;*

SUPERCOP version: 20120310

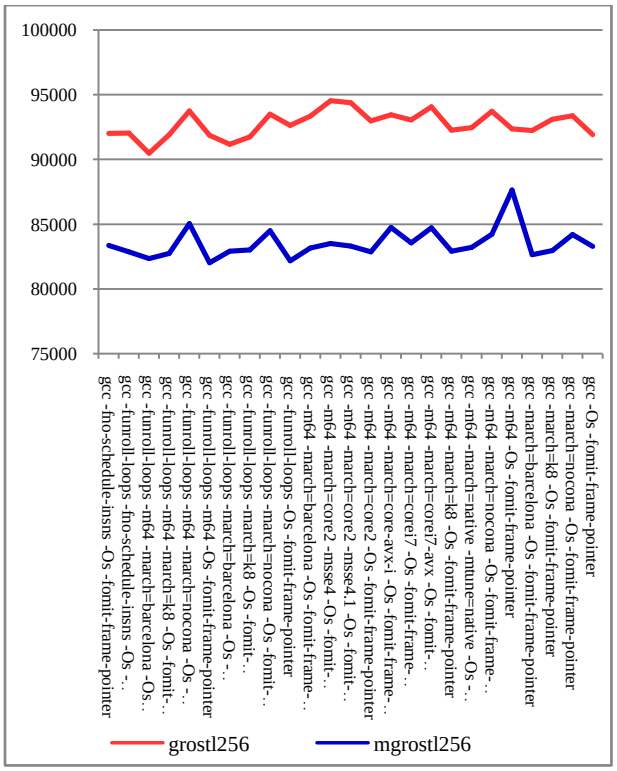

Fig 12: Time vs. compiler chart for Architecture: AMD64 Implementation: Opt 64

4.2.4 Architecture: armeabi, Computer:h4mx515e

Implementation: Opt 32,

SUPERCOP version: 20120310

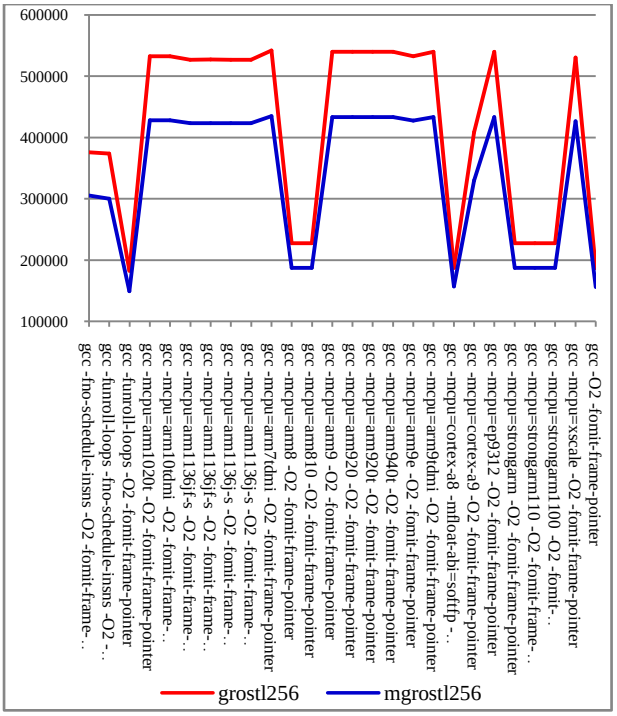

Fig 13: Time vs. compiler chart for Architecture: armeabi

4.2.5 Architecture: ppc32, Computer: stan

Implementation: Opt 32,

SUPERCOP version: 20120310

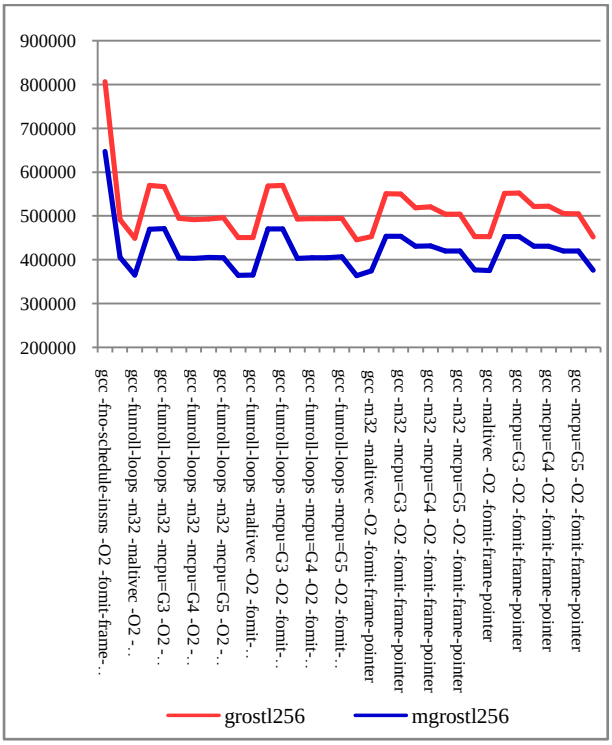

Fig 15: Time vs. compiler chart for Architecture: PPC32

5. CONCLUSION

The detailed analysis of algorithm is carried out along with performance evaluation in eBASH project of mGrøstl. According to the analysis from eBASH project, we can say that different performances can be gained by adapting different platform. This also shows that mGrøstl is as secure as Grøstl and is more efficient than Grøstl.

6. ACKNOWLEDGMENT

The authors would like to thank eBASH Project. They would also like to thank the anonymous reviewers for helpful suggestions and comments to improve the paper.

7. REFERENCES

[1] A. Menezes, P. Oorschot & S. Vanstone, 1997, Handbook of Applied Cryptography. CRC Press http://www.cacr.math.uwaterloo.ca/hac/

[2] E. Biham and R. Chen, "Near-collisions of SHA-0," in Advances in Cryptology—CRYPTO 2004, Lecture Notes in Computer Science, M.K. Franklin, Ed. Berlin, Germany: Springer-Verlag, 2004, vol. 3152, pp. 290–305.

[3] National Institute of Standards and Technology, 2011, Cryptographic Hash Project SHA-3 contest, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[4] J.P. Aumasson, L. Henzen, W. Meier, R. Phan, 2011, SHA-3 Proposal Blake. Candidate to the NIST Hash Competition.

[5] P. Gauravaram, L. Knudsen, K. Matusiewicz, F. Mendel, 2011, C. Rechberger, M. Schlaffer, and S. Thomsen: Grøstl - a SHA-3 candidate. Submission to NIST http://groestl.info.

[6] H. Wu, 2011, JH. Candidate to the NIST Hash Competition.

[7] G. Bertoni, J. Daemen, M. Peeters , G. Assche, 2011, Keccak. Candidate to the NIST Hash Competition.

[8] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, J. Walker, 2008, Skein. Candidate to the NIST Hash Competition.

[9] D. J. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems, http://bench.cr.yp.to, accessed 1 Feb 2012.

[10] Gurpreet Kaur, Vidyavati S Nayak, Dhananjoy Dey, S. K. Pal, "Modified Grøstl: An Efficient Hash Function", *(Accepted In proceeding)* CSIA2012, Advances in Intelligent and Soft Computing book Series, Springer.

[11] D. Karras & V. Zorkadis, "A Novel Suite of Tests for Evaluating One-Way Hash Functions for Electronic Commerce Application". IEEE 2000

[12] R. Chatterjee, M.A. Saifee, and D. RoyChowdhury, "Modifications of SHA-0 to Prevent Attacks," in S. Jajodia and C. Mazumdar (Eds.) ICISS 2005, LNCS 3803, Springer-Verlag Berlin Heidelberg 2005, pp. 277–289, 2005.

[13] D. J. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems, http://bench.cr.yp.to/impl-hash/mgrostl256.html accessed 3 Mar 2012.

[14] D. J. Bernstein and T. Lange, 2012, The new SHA-3 software shootout. http://eprint.iacr.org/2012/004.pdf