PERFORMANCE ANALYSIS OF SHA-2 AND SHA-3 FINALISTS

Ram Krishna Dahal, Jagdish Bhatta, Tanka Nath Dhamala[1]

Central Department of Computer Science & IT, Tribhuvan University, Kathmandu, Nepal
[1]Presently, Alexander von Humboldt Research Fellow at the University of Kaiserslautern, Germany.

ABSTRACT

National Institute of Science and Technology (NIST) published the first Secure Hash Standard SHA-0 in 1993 as Federal Information Processing Standard publication (FIPS PUBS) which two years later was replaced by SHA-1 to improve the original design and added SHA-2 family by subsequent revisions of the FIPS. Most of the widely used cryptographic hash functions are under attack today. With the need to maintain a certain level of security, NIST had selected new cryptographic hash function through public competition. The winning algorithm, Keccak will not only have to establish a strong security, but also has to exhibit good performance and capability to run. In this context, we have analysed SHA-3 finalists along with the used standard SHA-2. The performances of respective algorithms are evaluated by computing cycles per byte. The empirical analysis shows that two SHA-3 finalists viz. Skein and BLAKE perform better which are nearly same as the performance of SHA-2.

KEYWORDS

Hash Functions, SHA-2, SHA-3 Finalists, BLAKE, JH, Skein, Keccak, Grøstl

1. INTRODUCTION

Security is omnipresent. The major goal of security can be classified into three parts: confidentiality, integrity, and availability. Confidentiality refers to the secrecy of the message that can be achieved by means of encryption/decryption technique. Integrity refers to the correctness of the message in which the changes have to be done only through authorized mechanism, detected otherwise. And availability is the challenge to make available the data to authorized entity at any time [5, 10]. One way of achieving integrity is creating message digest using hash functions. Nowadays, cryptographic hash functions are considered as workhorses of cryptography. They are originally created for improving the efficiency of digital signature, and now used to secure the very fundamentals of our information infrastructure [4]. A series of related hash functions have been developed, such as MD4, MD5, SHA-0, SHA-family.

NIST does not currently plan to withdraw SHA-2 or remove it from the revised Secure Hash Standard; however, it is intended that SHA-3 can be directly substituted for SHA-2 in current applications, and will significantly improve the robustness of NISTs overall hash algorithm toolkit. It is therefore, worthwhile to compare the performance between the currently used SHA-2 along with the SHA-3 finalists. The submitted algorithms for SHA-3 are supposed to provide message digests of 224, 256, 384 and 512 bits to allow substitution for the SHA-2 family. Since SHA-3 is expected to provide a simple substitute for the SHA-2 family of hash functions, certain

DOI:10.5121/ijcis.2013.3301

properties of the SHA-2 hash functions must be preserved, including the input parameters; the output sizes; the collision resistance; pre-image resistance and second pre-image resistance properties along with the one- pass streaming mode of execution [17].

2. RELATED WORKS

After the successful attack on SHA-1 [12], NIST feel the necessity to develop a new cryptographic hash algorithm through public competition. The winner of the competition i.e. Keccak is named as SHA-3, and will complement the SHA-2 hash algorithms currently specified in Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard.

NIST proposed three categories of evaluation criteria viz. security, cost & performance and implementation characteristics for the five finalists [17]. Even though security of the algorithm was identified as the most important factor when evaluating the candidates, it is of great concern to analyse the SHA-3 finalists with the used SHA-2. In [17] NIST identifies a number of well-defined security properties that was expected for the winning candidate. Moreover, as specified in [17], cost and performance were identified as the second-most important criterion upon evaluating the various candidates. Within the context of this competition, cost includes computational efficiency and memory requirements. Computational efficiency refers to the speed of an algorithm. And as NIST states in [20] that, NIST expects SHA-3 to offer improved performance over the SHA-2 family of hash algorithms at a given security strength.

A comprehensive comparison of all Round 3 SHA-3 candidates and the current standard SHA-2 has presented in [6] for hardware performance in modern FPGAs, and suggested that Keccak is the only candidate that consistently outperforms SHA-2 for all considered FPGA families and two hash function variants, while BLAKE is the most flexible for hardware implementation. In high-performance implementations [21], Keccak is the only candidate that is better than SHA-2 the clear throughput/area winner. Skein and BLAKE have the lowest maximum throughput to area ratio. All of the algorithms seem reasonably efficient, but BLAKE and Grøstl seem the most flexible. Performance analysis of SHA-3 finalists dealing with Java code optimization techniques [8] show that two SHA-3 finalists, Skein and Keccak, feature exceptional performance in Java and the third and fourth place - depending on the digest length and the word size BLAKE and Grøstl which are followed by JH that lags far behind.

3. THE SHA-3 FINALISTS

3.1. BLAKE

BLAKE has not recreated wheel; BLAKE is built on previously studied components, chosen for their complementarity [1]. The inheritance of BLAKE is treble; BLAKE's iteration mode is HAIFA, an improved version of the MerkleDamgard paradigm that provides resistance to long-message second pre-image attacks, and explicitly handles hashing with a salt. Next, BLAKE's internal structure is the local wide-pipe. It makes local collisions impossible in the BLAKE hash functions, a result that doesn't rely on any intractability assumption. And the third one is that the BLAKE's compression algorithm is a modified version of Bernstein's stream cipher ChaCha, whose security has been intensively analysed and performance is excellent, and which is strongly parallelizable. As in [1], the iteration mode HAIFA would significantly benefit to the new hash standard, for it provides randomized hashing and structural resistance to second pre-image attacks. The BLAKE local wide-pipe structure is a straightforward way to give strong security guarantees against collision attacks. The choice of borrowing from the stream cipher ChaCha comes from the fact behind the cryptanalysis of Salsa20 and ChaCha [2].

3.2. Grøstl

Grøstl is a collection of hash functions, capable of returning message digests of any number of bytes from 1 to 64, i.e., from 8 to 512 bits in 8-bit steps. The variant returning n bits is called Grøstl-n. In [7], the submitter has explicitly stated the message digest sizes 224, 256, 384, and 512 bits. Grøstl is a wide-pipe Merkle-Damgard (MD) hash algorithm with an output transformation. The compression function is a fresh construction, using two fixed $2n$-bit permutations - denoted by P and Q - in parallel to produce a $2n$-bit compression function. Intermediate chaining values are 512 bits wide for Grøstl-256 and 1024 bits for Grøstl-512. The output transformation processes the final chaining state and discards half the bits of the result to yield an n-bit message digest. Grøstl-256 has ten rounds, while Grøstl-512 has fourteen rounds. The permutations, P and Q, are based on the structure of AES, reusing the AES S-box, but expanding the size of the block to 512 bits for Grøstl-224 and Grøstl-256, and to 1024 bits for Grøstl-384 and Grøstl-512 in a straightforward way [21].

3.3. JH

JH family specifies four hash algorithms - JH-224, JH-256, JH-384, and JH-512. In the design of JH, a new compression function structure is proposed to construct a compression function from a large block cipher with constant key. The AES design methodology is generalized to high dimensions so that a large block cipher can be constructed from small components easily. Hash function JH consists of five steps: padding the message M, parsing the padded message into message blocks, setting the initial hash value $H^{(0)}$, computing the final hash value $H^{(N)}$, and generating the message digest by truncating $H^{(N)}$ [13]. The JH family of hash algorithms for different hash sizes is based on a single compression function, $F8$, which uses a fixed 1024-bit permutation, $E8$. In $F8$, a message block of 512 bits is XORed with the first 512 bits of the input of $E8$, and the last 512 bits of the output of $E8$. The permutation $E8$ uses two 4×4-bit S-boxes, S_0 and S_1, an eight-bit linear permutation L, and a permutation $P8$. Different members of the JH family are distinguished by the different IVs used, and the message digests are obtained by truncating the final output to the desired hash sizes [21].

3.4. Keccak

Keccak is obtained from sponge function with parameters capacity c, bitrate r and diversifier d with the application of specific padding to input. To ensure the message can be evenly divided into r-bit blocks, it is padded with the bit pattern 10*1. The basic block permutation function consists of $12+2l$ iterations of five sub-rounds. For absorbing r bits of data, the data is XORed into the leading bits of the state, and the block permutation is applied. To compress, the first r bits of the state are produced as output, and the block permutation is applied if additional output is desired. Hash is computed by initializing the state to 0, padding the input, and breaking it into r-bit parts. Input is drawn into the state; i.e., for each piece, it is XORed into the state and then the block permutation is applied. After the final block permutation, the leading n bits of the state are the desired hash. Though some tweaks have made latter [3]. The message block size varies according to the output size: Keccak-512 has a block size of 576 bits, Keccak-384 has 832 bits, Keccak-256 has 1088 bits, and Keccak-224 has 1152 bits. In keeping with the standard terminology for a sponge construction, the message block size is referred to as r, for rate, and the difference between the permutation size and the message block size is referred to as c, for capacity [3].

3.5. Skein

Skein is a family of hash functions with three different internal state sizes: 256, 512, and 1024 bits. Skein-512 is introduced as primary proposal by its designer. Skein is an iterative hash algorithm that is rested on a tweakable block cipher -Threefish. Threefish is used to build the compression function of Skein using a modified Matyas-MeyerOseas construction, which is then iterated using a domain extender similar to the HAIFA domain extender used by BLAKE. The designers refer to the whole construction as a Unique Block Iteration (UBI). The internal structure of Threefish is a 72-round substitution-permutation network using a 128-bit MIX function consisting of 64-bit addition, rotate and XOR operations, each used exactly once per MIX function. [21]

3.6. SHA-2

The SHA-2 algorithms use a conventional Davies-Meyer construction with a chaining variable of eight words and a message block input of sixteen words. SHA-256 uses 32-bit words, while SHA-512 uses 64-bit words in its design. Except for the word size and the number of rounds that is called for, the two compression functions are similar and rely on bitwise AND operations in the round function and modular addition, in the message expansion and the round function, for nonlinearity. It requires to recomputed only two words of the chaining variable during each round, and part of round $i + 1$ can be pre-computed during round i [21].

4. IMPLEMENTATION AND ANALYSIS

All of the algorithms are implemented in JAVA. The Java implementations of the candidates are based on the reference implementation representing the candidate in the third round. As the goal of this work is to measure the performance of SHA-3 finalists along with the currently used SHA-2, the implementations are done for two variants of each family that are 256 and 512. The extra functionality like salting and keyed hashing are also implemented but set as zero. As a part of performance evaluation, an empirical analysis has been done measuring the execution time for the candidate algorithms implemented in Java. The System configuration during the implementation and testing is Intel Core 64-bit machine with i5 processor of 2.30 Ghz in Ubuntu and Windows 7 environment. The execution time are measured using the standard JavaTM function System.nanoTime() for all of the algorithms. Though, various processes running in the background may affect the absolute execution time of the particular function that is maintained same for all algorithms.

4.1. Measuring Performance

When timing cryptographic primitives is concerned, the subject is how many cycles it takes to process a byte, on average. Measuring bytes per second is a useful thing when comparing the performance of multiple algorithms on a single box, but it gives no real indication of performance on other machines. Therefore, cryptographers prefer to measure how many processor clock cycles it takes to process each byte, because doing so allows for comparisons that are more widely applicable. Such comparisons will generally hold fast on the same line of processors running at different speeds. For evaluating the cycle per byte of cryptographic hash function, the number of bytes processed is divided by the number of cycles it takes to process. One important thing to note about timing cryptographic code is that some types of algorithms have different performance characteristics as they process more data. That is, they can be dominated by per-message overhead costs for small message sizes.

The reason behind using Cycles/byte for measuring performance is that, if just absolute timings are measured, then obviously the better performance will be in the higher frequency of the processor. During this study, Cycles/byte calculation is performed with the parameters viz. Time in seconds spent performing hash (*Ts*), Frequency of the CPU in Hz (*F*) and Message input length in bytes (*L*). Hence Cycles/byte is computed as [9, 22]:

$$\text{Cycles/byte} = \frac{Ts \times F}{L}$$

4.2. Analysis

Following tables and corresponding charts show the overall performance in the two different architectures. The graphs illustrated in figures 1, 2 and 3 depict the performance of candidates for 1KB, 1MB and 256MB on Ubuntu system. Similarly, figures 4, 5 and 6 show the performance of candidates for 1KB, 1MB and 64MB input sizes on Windows system respectively.

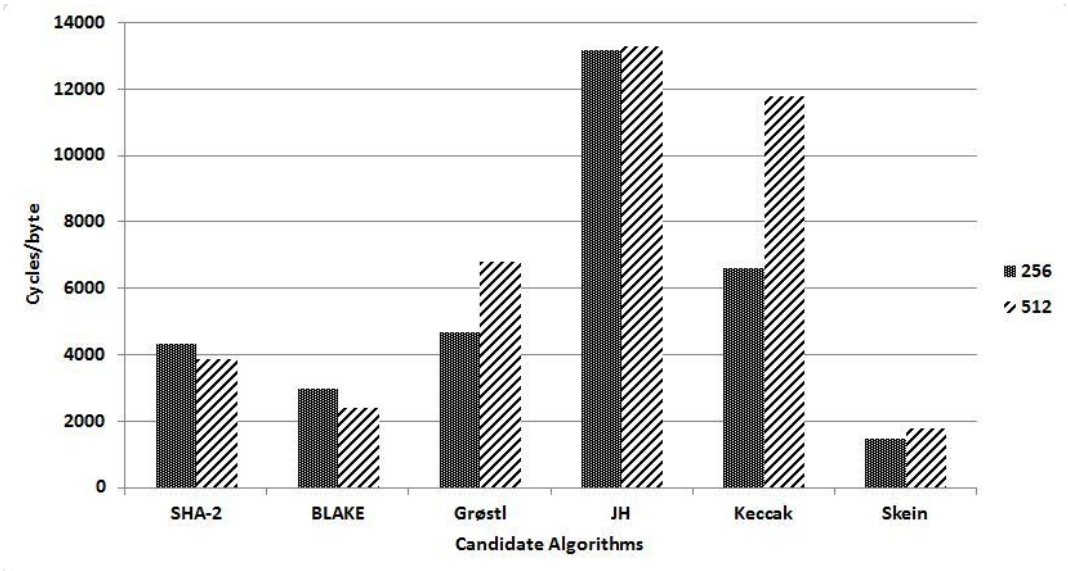

Figure 1: Performance of SHA-3 finalists and SHA-2 on Java/64-Bit Server (Ubuntu 11.10) for 1KB input size

From above graph it can be deduced that Skein is best in each categories of 256 and 512 bit size, for 1 KB input size of message. The greater variation is seen on the performance measured in Cycles/byte as the Skein is approximately 8 times faster than the worst performing JH, while the performance of currently used SHA-2 is almost 3 times faster than the worst performing JH.

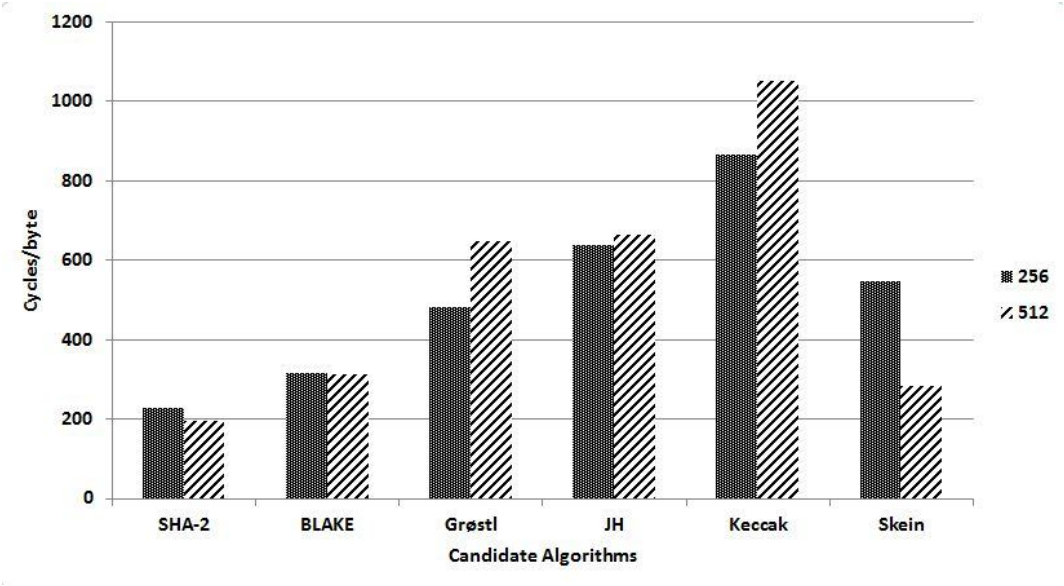

Figure 2: Performance of SHA-3 finalists and SHA-2 on Java/64-Bit Server (Ubuntu 11.10) for 1MB input size

As the size of input message is increased to 1MB, the SHA-2 is seen as best performer. The performance of Skein is affected largely by the block size of the hashing algorithm. In this case, the running time of Keccak, worst performer, takes nearly 4 times greater time than that of SHA-2. BLAKE, Skein, Grøstl and JH come in order of best to worst in between SHA-2 and Keccak.

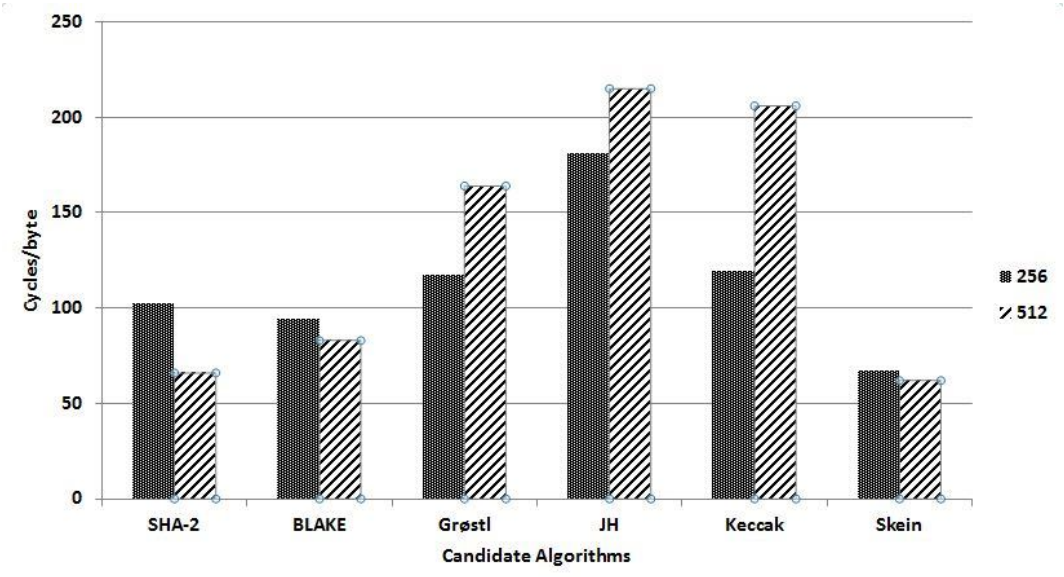

Figure 3: Performance of SHA-3 finalists and SHA-2 on Java/64-Bit Server (Ubuntu 11.10) for 256MB input size

Here in input size of 256 MB, the performance of Skein is consistent for different hash size, 256 and 512, and seems best. The running time for different candidate algorithms seems to be contracted as the size of the input become larger regardless of the environment.

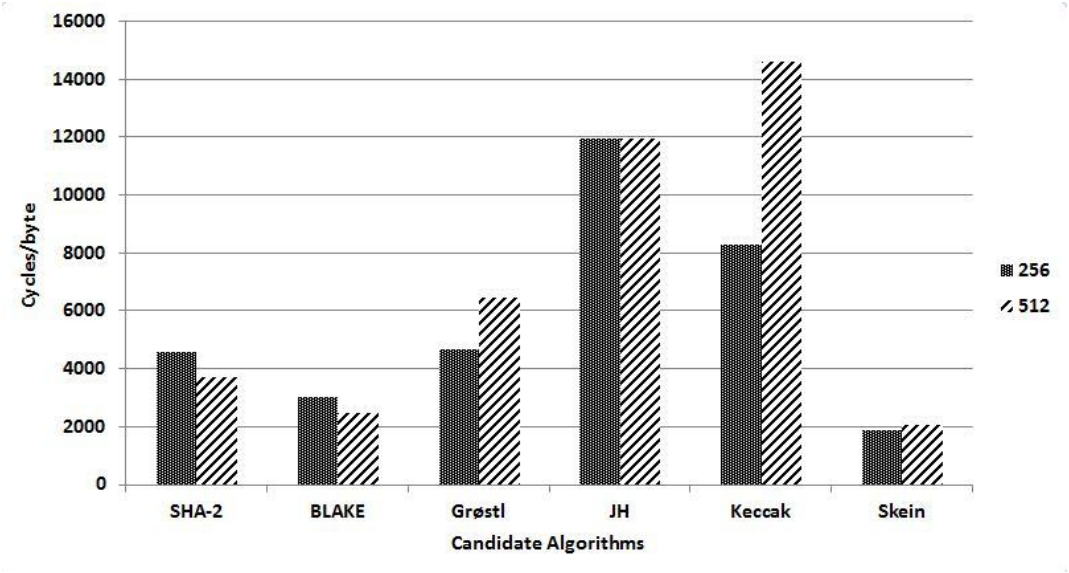

Figure 4: Performance of SHA-3 finalists and SHA-2 on Java/x86 (Windows 7) for 1KB input size

With the above graph it can be observed that JH performs nearly same regardless of the input size. There is vast difference on the performance of Keccak for hash size 256 and 512, though both of the algorithms found to be bad performer. The efficiency of Skein is nearly 6 times better than that of JH on the basis of runtime. Skein leads, then after BLAKE and SHA-2 follows successively.

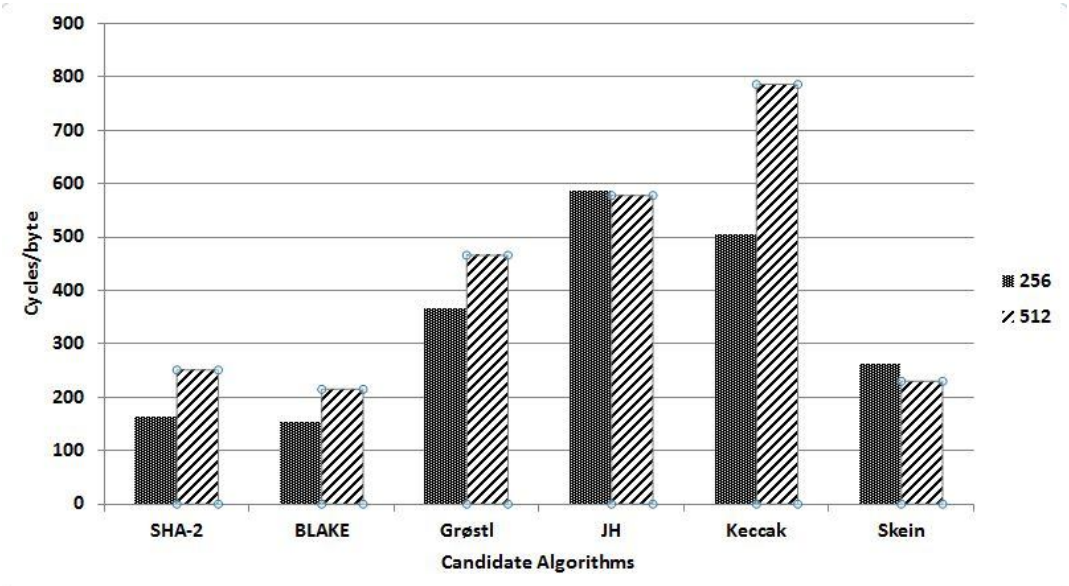

Figure 5: Performance of SHA-3 finalists and SHA-2 on Java/x86 (Windows 7) for 1MB input size

For input size 1MB on Windows environment, BLAKE seems to be a best choice which is nearly 4 times faster than Keccak and thereafter SHA-2 and Skein respectively.

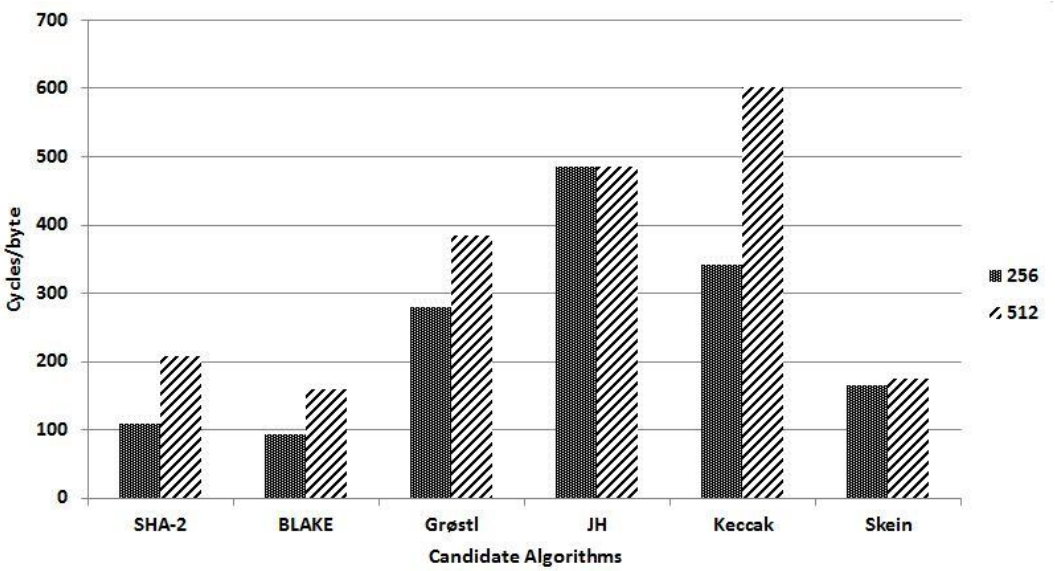

Figure 6: Performance of SHA-3 finalists and SHA-2 on Java/x86 (Windows 7) for 64MB input size

With the input size of 64MB, the Cycles/byte for Keccak 256 is dominantly worst, and then JH performs badly on both 256 and 512 hash size. BLAKE 512 is best which nearly better 1.5 times than SHA-2, 2 times than Skein, 3 times than Grøstl, 3.5 times than Keccak and 5 times than JH. However, Blake 256 being best performer but at the same time, SHA-2 comes along with it.

5. CONCLUSIONS

From the above mentioned illustrations, on Ubuntu environment, it is found that SHA-2, BLAKE, and Skein runs better in 512-bit hash size than the 256-bit. While Grøstl, JH, and Kecccak run better on hash size of 256-bit. But in case of Windows environment, none of the algorithm outweighs themselves in either of the hash sizes. The performance result shows that, the value of Cycles/byte decreases as the size of input message increases. For very short message Skein leads and is followed by Blake and Grøstl. In case of large input message size, Blake beats others, and Skein, Grøstl, Keccak, and JH come in order.

Finally, in general, SHA-3 finalists namely Skein and BLAKE perform better which is nearly same as the performance of SHA-2. Depending on the digest length and block size Grøstl, Keccak and JH comes in order. By assuming, all algorithms are equally secure, only performance is the major factor then the best candidate for replacement of SHA-2 is Skein. Regarding to plain function to be used for the replacement of SHA-2, BLAKE is the way to go, since it is closest in terms of performance in the tested architecture.

6. FUTURE WORKS

The priority of this work has been to analyse the five SHA-3 finalists along with the SHA-2 based on the running time for evaluating their performances and no special effort is given for analysing the security of the candidates. Further, the hardware based optimization can be performed to get better performance of the algorithm. As a future work, a possibility here can be to create three implementations of each candidate, one optimized for size, one optimized for speed and one for

providing better security that might be used as required. The work can be extended for the resource constrained devices as well.

REFERENCES

[1] J. P. Aumasson, et al., SHA-3 proposal BLAKE, version 1.3, December 16, 2010
[2] J. P. Aumasson, et al., New features of Latin dances: analysis of Salsa, ChaCha, and Rumba. In FSE, 2008.
[3] G. Bertoni, et al., The Keccak reference, Version 3.0, January 14, 2011
[4] N. Ferguson, et al., The Skein Hash Function Family, Version 1.3, 1 Oct 2010
[5] B. A. Forouzan and D. Mukhopadhyay, Cryptography and Network Security, 2nd Edition, Tata McGraw-Hill, 2010.
[6] K. Gaj, et al., Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs, George Mason University
[7] P. Gauravaram, et al., Grøstl - a SHA-3 candidate, March 2, 2011
[8] C. Hanser, Performance of the SHA-3 Candidates in Java, Institute for Applied Information Processing and Communications Graz, University of Technology, March 19, 2012
[9] M. Knutsen, K. A. Martinsen, Java Implementation and Performance Analysis of 14 SHA-3 Hash Functions on a Constrained Device, Norwegian University of Science and Technology, Department of Telematics, June 2010
[10] W. Stallings, Cryptography and Network Security Principles and Practices, Fourth Edition, Prentice Hall, 2005
[11] M. S. Turan, et al., Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition, NIST Interagency Report 7764, National Institute of Standards and Technology, U.S. Department of Commerce
[12] X. Wang, et al., Finding collisions in the full sha-1., In "In Proceedings of Crypto", Springer, 2005.
[13] H. J. Wu, The Hash Function JH, 16 January, 2011
[14] A brief history of netbeans, June 2010. http://netbeans.org/about/history.html.
[15] Cryptographic hash algorithm competition, April 2010. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
[16] Cryptographic sponge functions, January 2011 http://sponge.noekeon.org/.
[17] Federal Register / Vol. 72, No. 212 / Friday, November 2, 2007 / Notices
[18] Netbeans ide 7.1 features, May 2010. http://netbeans.org/features/index.html.
[19] http://www.saphir2.com/sphlib/files/sphlib-3.0.zip
[20] Status report on the first round of the SHA-3 cryptographic hash algorithm competition, September 2009.
[21] Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition
[22] Timing Cryptographic Primitives, http://etutorials.org

Authors

Tanka Nath Dhamala, Professor of Mathematics and Head of Central Department of Computer Science and IT, Tribhuvan University, PhD from Germany in 2002 and has Post-Doc experience from Canada in the field of discrete optimization. He has supervised a number of research students and published a number of papers in international and national journals. Professor Dhamala has been involved in organizing international and national conference/workshop/seminars and is a member of different academic organizations and societies. Former DAAD and NSERC research fellow, Dr. Dhamala is a Alexander von Humboldt Research Fellow at the University of Kaiserslautern, Germany.

Jagdish Bhatta, Lecturer Central Department of Computer Science and IT, graduated from TU and published some research papers in national and international journals and supervised a number of research students.

Ram Krishna Dahal, Student of M. Sc. Computer Science and IT, Central Department of Computer Science and IT, Tribhuvan University, published some articles on national journal and participated in many national and international workshops and seminars.