SHA-3 Blake Finalist on Hardware Architecture of ARM Cortex A8 Processor

Gurpreet Singh
Electronics and Communication Engineering
Ludhiana College of Engineering and Technology,
Ludhiana (INDIA)

Rajeev Sobti
Computer Science Engineering
Lovely Professional University,
Phagwara, (INDIA)

ABSTRACT

Information is an asset in today's life. Internet plays major role for sharing the information between two parties. To protect the information from attacks there exist several algorithms. Cryptographic hash functions are the one that is used for the purpose of modern security. In mobile computing, portables devices are used to share information. Most of portable devices are based on ARM processors. In this work, a BLAKE algorithm from SHA-3 finalists is selected for analysis on ARM Cortex A8 Processor. BLAKE is a hash function selected by NIST in SHA-3 competition. Many factors need to be considered such as utilization of memories ROM or RAM, power consumption and cycles required for particular algorithm. In this paper, the objective is to compare the performance of all variants of BLAKE in terms of cycles required on ARM Cortex A8.

General Terms

SHA-3 competition, Kernel Loadable module, Minicom, Embedded Linux, AM335x, *scp* command, *insmod* command, NIST.

Keywords

Cryptography, Hash functions, SHA-3, BLAKE, ARM11, ARM Cortex A8.

1. INTRODUCTION

Security must require when information is sent and received over communication channel to prevent from different types of attacks. Information needs to be protected from unauthorized access, unauthorized change and available only for authorized person when it is needed, which are three fundamental requirement of security [1].

Cryptography is defined as the science and study of secret writing and used to provide secure communication for transferring information over network [2].

1.1 Cryptographic Hash Functions

A Cryptographic Hash functions are those functions which compress an input of variable-length block of data length to a fixed length hash output value i.e. MD (Message digest) [3].

According to need, Cryptographic Hash functions are restricted by four security requirements.
Properties of Cryptographic hash functions h(x) [2] [3]:
1. A hash function should be easy to calculate.
2. Preimage Resistance: For each and every output of a hash function, it is 'computationally impracticable' to find any input hashing to that output.

3. Second Preimage Resistance: For a given input, It is 'computationally impracticable' to find a second input hashing to the same output.

4. Collision Resistance: It is 'computationally impracticable' to find two colliding inputs, i.e., x and x' such that x'! = x with h(x) = h(x').

There are different types of hash functions like MD4, MD5, RIPEMD, RIPEMD-160, HAVAL, Tiger, Whirlpool, Radio Gatun, SHA-0, SHA-1 and SHA-2 [1]. Some attacks were observed on SHA – 0 and SHA – 1[2]. SHA – 2 has same structure as SHA –1. In 2007, NIST (National Institute of Standards Technology) announced call for developing new hash algorithms which improves weaknesses of MD5, SHA1 and SHA-2, more reliable and given a name as SHA-3[4] [9].

SHA-3 has five algorithms BLAKE, Grøstl, JH, KECCAK and Skein which is selected as SHA-3 finalists. [4][5][6][9].

2. BLAKE HASH FUNCTION

BLAKE is one of five hash algorithms which is selected for the final round of NIST SHA-3 competition [9]. It is one of the simplest design to implement, and relies on previously analyzed components: the HAIFA structure and the ChaCha core function [7][8].

The two main types of BLAKE are BLAKE-256 and BLAKE-512. They are work with 32 and 64-bit words, and produce 256 and 512-bit digests. BLAKE is a family of four hash functions: BLAKE-224, BLAKE-256, BLAKE-384, and BLAKE-512.

BLAKE has a 32-bit version (BLAKE-256) and a 64-bit one (BLAKE-512), from which other instances are derived using different initial values, different padding, and truncated output [8].

The final BLAKE consists is an increased number of rounds: 14 instead of 10 for BLAKE-224 and BLAKE-256, and 16 instead of 14 for BLAKE-384 and BLAKE-512. The table listed below shows properties of BLAKE Hash Function:

Table 1. Properties of the BLAKE hash functions (in bits).

Parameter	Blake-224	Blake-256	Blake-384	Blake-512
Word	32	32	64	64
Message	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block	512	512	1024	1024
Digest	224	256	384	512
Salt	128	128	256	256

The hash function BLAKE-256 operates on 32-bit words and returns a 32-byte hash value.

Initial values and constants for Blake-256 are given as below [8]:

Initial Values

IV0 = 6A09E667	IV1 = BB67AE85
IV2 = 3C6EF372	IV3 = A54FF53A
IV4 = 510E527F	IV5 = 9B05688C
IV6 = 1F83D9AB	IV7 = 5BE0CD1

Constants

C0 =	243F6A88	C1 =	85A308D3
C2 =	13198A2E	C3 =	3707344
C4 =	A4093822	C5 =	299F31D0
C6 =	082EFA98	C7 =	EC4E6C89
C8 =	452821E6	C9 =	38D01377
C10 =	BE5466CF	C11 =	34E90C6C
C12 =	C0AC29B7	C13 =	C97C50DD
C14 =	3F84D5B5	C15 =	B5470917

The compression function of BLAKE-256 takes as input four values:
• a chain value h = h0 to h7
• a message block m = m0 to m15

• a salt s = s0 to s3, counter t = t0, t1

Table 2. Permutations of used by the BLAKE as below [8]:

σ_0	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	14	11	7	2	9	12	13	6	10
1	10	8	9	12	0	5	11	15	2
2	4	12	3	6	5	1	7	14	8
3	8	0	1	10	7	15	14	9	4
4	9	5	13	0	9	14	12	11	7
5	15	2	12	11	4	13	1	3	6
6	13	15	11	8	10	4	3	0	1
7	6	13	15	3	15	10	9	8	5
8	1	10	14	4	14	0	5	12	15
9	12	14	1	13	1	7	0	2	11
10	0	3	11	7	11	6	15	13	9
11	2	6	12	5	12	3	4	7	14
12	11	7	6	15	6	9	8	1	3
13	7	1	8	14	8	2	6	4	12
14	5	9	3	1	3	8	2	10	13
15	3	4	13	9	13	11	10	5	0

The compression function of Blake algorithm is divided in to three steps:

1. Initialization
2. Round function
3. Finalization

Although, these three steps are same in all Blake 224, 256,384,512 but with minor changes is explained later on this context.

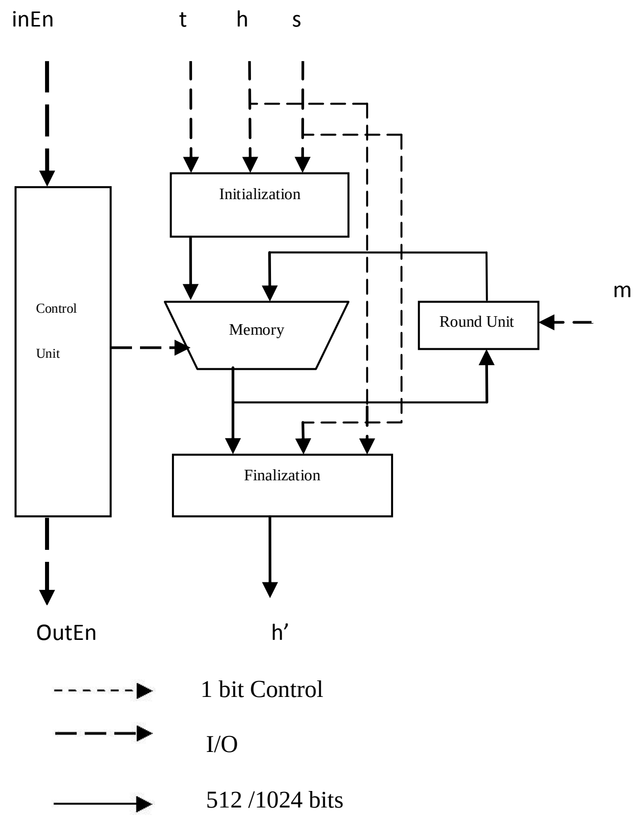

Figure 1: Block Diagram of Blake Hash Function

2.1 Initialization:

A 16-word state v0 to v15 is initialized such that different inputs produce different initial states [7].

The state is represented as a 4×4 matrix, and filled v_0 to v_{15} as follows:

$$\begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{array}$$

which is obtained from below:

$$\begin{array}{cccc} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \text{ xor } c_0 & s_1 \text{ xor } c_1 & s_2 \text{ xor } c_2 & s_3 \text{ xor } c_3 \\ t_0 \text{ xor } c_4 & t_0 \text{ xor } c_5 & t_1 \text{ xor } c_6 & t_1 \text{ xor } c_7 \end{array}$$

Here
h = chain value,
s =salt,
c = constant, t = time period

2.2 Round function

When the state v is initialized, the compression function iterates a series of 14 rounds. The notation of round is r, Gi (a, b, c, d) sets Gi function is performed in to three important function [7][8].

1) + addition modulo 232 or (modulo 264).
2) ⊕ Boolean exclusive OR (XOR).
3) ≫ k rotation of k bits towards less significant bits.
4) ≪ k rotation of k bits towards more significant bit

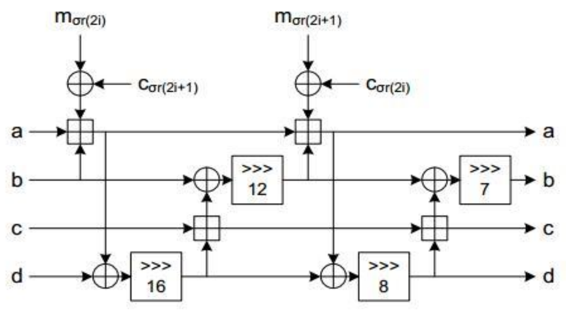

Figure 2: The Gi function.

The sequence G0 to G3 is called a column step and similarly, the last four calls G4 to G7 are called diagonal step.

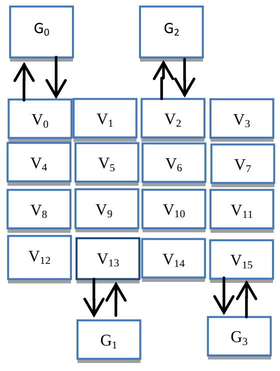

Figure 3: G functions in column step

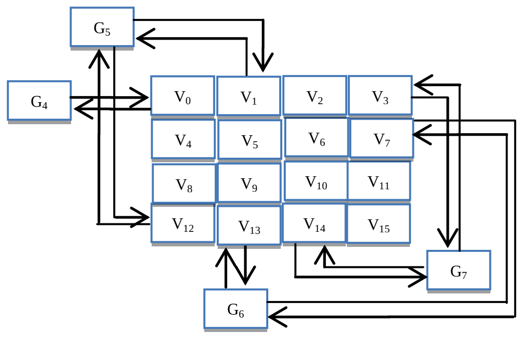

Figure 4: G functions in diagonal step

Round Function iterated 14 times in Blake-256 and the G - function repeated 112 times in Blake 256 (32-bits word)

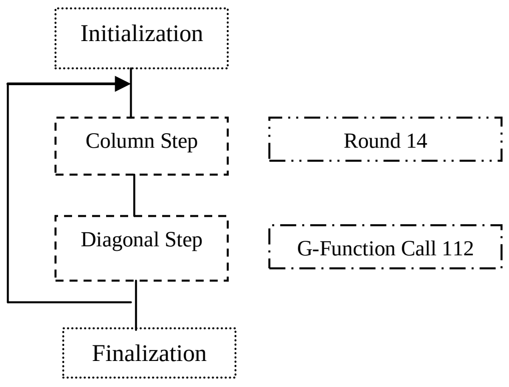

Figure 5: Round Function Iteration

2.3 Finalization:

When the round function is done, the new chain value h′0, to h′7 is take out from the state v0 to v15 with input of the initial chain value h0 toh7 and the salt s0 to s3:

h’0	=	h0 xor s0	xor v0 xor v8
h’1	=	h1 xor s1	xor v1 xor v9
h’2	=	h2 xor s2	xor v2 xor v10
h’3	=	h3 xor s3	xor v3 xor v11
h’4	=	h4 xor s0	xor v4 xor v12
h’5	=	h4 xor s1	xor v5 xor v13
h’6	=	h6 xor s2	xor v6 xor v14
h’7	=	h7 xor s3	xor v7 xor v15

3. Hashing a message for Blake-256

Hashing of message divided in two parts:
Padding and Iterated hash. Hashing of message depends on *m* bit length. The message is first padded and then Iteration hash processed block by block compression function.

3.1 Hashing a message for Blake-512

The process of message padding is given as below:
Append a bit 1 and as many 0 bits until the message bit length is ≅ to 895 modulo 1024. Then append a bit 1 and a 128-bit unsigned big-endian representation of the message bit length:

$$m \leftarrow m \| 1000 \ldots 0001(\ell)_{128}$$

4. BLAKE-512

operates on 64-bit words and returns a 64-byte hash value. All lengths of variables are doubled compared to BLAKE-256:

- Chain values are 512-bit,
- Message blocks are 1024-bit,
- Salt is 256-bit,
- Counter is 128-bit and

The Round Function iterated 16 times in Blake-512 and G - function repeated 128 times in Blake 512 (64-bits word).The details of initial values and constants can be found at [8].

4.1 BLAKE-224

Is similar to BLAKE-256 and **BLAKE-384** is similar to BLAKE-512 the detail can be found at [8].

5. ARM ARCHITECTURE

ARM processor plays significant role in handheld devices. Instead of computer, the mobiles phones and tablets are used for sharing information over emails and financial transactions in banking sector, online shopping etc. In our surroundings, most of portable devices are based on ARM processors.

5.1 Important Features of ARM [10]:

- ARM processor having increment, decrement logic and barrel shifter which is independent from ALU: This feature makes architecture comparatively fast from others because no heavy burden of ALU.

- **Low cost and small size:** Low cost enables device manufactures to use ARM in their products. Small size fits in to applications even like IoT (Internet of Things).
- **Less power consumption:** ARM is also suitable for power sensitive as a basic requirement of handheld battery operated applications

5.2 ARM Cortex A8

The ARM Cortex-A has wide range of OS (Linux, Android, iOS) based applications and multiples software tasks. The ARM Cortex-A series processors are based on 32-Bit RISC processor with modified Harvard architecture having Load/Store and Thumb-2 instruction. It also has Big-endian and little-endian data access support [11].The Cortex-A8 processor was the first to implement the ARMv7-A architecture. It is available in a number of different processors such as the S5PC100 from Samsung, the AM335x from Texas Instruments and the i.MX515 from Freescale etc. A wide range of device performances are available and it can run clock speeds of more than 1GHz [11, 12].

ARM Cortex A8 is used to evaluate performance of BLAKE because it is widely used for application devices from in mobiles, tablets and smart phone etc.

6. IMPLEMENTATION SETUP

Linux is very much popular as per developer's point of view. Millions of computer users have been putting Linux to work for more nearly 20 years. Ubuntu is an operating system based on the Linux kernel; created, improved, refined, and distributed by the Ubuntu Community at www.ubuntu.com [13]. In this work, Ubuntu 12.04 LTS version is used at host side.

6.1 Embedded Linux

Embedded Linux means Linux is downloaded in microprocessor which is target for embedded applications. The same Linux kernel source code (as in desktop Linux) built for particular hardware by recompiling the kernel source code with minimizing the optional features which is not a requirement for those particular applications [13]. Embedded Linux or any OS in used for embedded application for parallel processing. Embedded Linux board having AM335x processor with preloaded Linux is chosen for analysis at target side. The Open board-AM335x has the features for supporting the phyCORE-AM335x RS-232 transceiver female DB9 connector and RJ-45 jack with 10/100/1000 Mbps Ethernet PHY[15]. Linux Kernel version 3.2.0 is used at target end. Minicom is used to access Linux board and arm gcc complier is used to compile the each algorithm.

6.2 Remote access of Linux Machines for Sharing files between each other

RS-232 interface is used for communicating board with host computer and Ethernet connector is used for sending and receiving files between board and computer with usage of TFTP server or scp command [15]. To transfer the files between board and computer a network connection must be established first. Configure the IP address for host by selecting the option network connection and modify the related option or issue below command [13].

```
$ ifconfig eth0 192.168.1.10 up
```

On Target side issue the same command with a different IP as below

```
$ ifconfig eth0 192.168.1.11 up
```

It must be noted the IP of Host and target side should not be same.

6.3 Enabling Cycle Count Register with Kernel Loadable Module

Linux is the ability to extend at runtime the set of features offered by the kernel. This means that you can add functionality to the kernel (and remove function as well) while the system is up and running. Each piece of the code can be added to the kernel in called module [16].

To get the count from processor, performance counters must be enabled for ARM Cortex A8 processor. A short kernel loadable module is written because, by default these counters are disabled [16].The description of enabling cycle count register is find at [6]. Modules are pieces of code that can be loaded and unloaded into the kernel upon demand**.** They extend the functionality of the kernel without the need to reboot the system [16].

To load the gurpreet.ko module below command is used.

```
insmod gurpreet.ko
```

In a similar way, to remove a module, rmmod utility is used by below command.

```
rmmod gurpreet.ko
```

Minicom is used for accessing the Linux board on Host PC. Due to huge amount data is be analyzed, the file handing capability of 'C' language is used.

7. STEPS FOR ANALYSIS

7.1 At Host PC Side:

1) Create Linux kernel loadable to enable performance counter register.
2) To compile algorithm one by one for

ARM Cortex A8 processor

```
arm-cortexa8-linux-gnueabi-        gcc algo1.c –o algo1
```

3) Transfer the kernel loadable

```
scp  gurpreet.ko  root@192.168.1.11:/home .
```

4) Transfer the object file(s) one by one to Board

```
scp algo1 root@192.168.1.11:/home .
```

5) Transfer the test vector file one by one to Board

```
scp  LongMsgKAT.txt root@192.168.1.11:/home
```

7.2 At Board Side:

1) Type sudo – minicom to access the board.

2) Insert the module by typing below command *insmod gurpreet.ko*
3) Check whether the module is inserted properly *lsmod*

4) Run object code (One by One).

./algo1
5) As per above steps it will give output.txt with message length and no. of cycles consumed for that particular algorithm.

At last again, *scp 192.168.1.11/home /homeoutput.txt* to transfer the data from board to host PC for analysis purpose.

8. PRACTICAL RESULTS

The tables below shows cycles consumed for Blake 224, 256, 384 and 512 both for short and long messages. Due to large number of readings, few readings have chosen where our graphs changes abruptly.

Table 3: Summery of cycles consumed by Blake -224 and Blake-256 for short messages:

Sr. No.	Input Bytes	Blake 224	Blake 256
1	0	15037	15067
2.	54	27380	27425
3.	121	39705	39786
4.	187	52211	52697
5.	249	64591	64776

Table 4: Summery of cycles consumed by Blake -384 and Blake-512 for short messages:

Sr. No.	Input Bytes	Blake 384	Blake 512
1	0	49877	49404
2.	113	101159	101421
3.	129	94525	94753
4.	241	140343	140822
5.	255	140274	140451

Table 5: Summery of all Blake of Long Messages:

Sr. No.	Input Bytes	Blake 224	Blake 256	Blake 384	Blake 512
1.	256	70331	69282	151874	149404
2.	453	101721	102297	187323	188786
3.	650	138988	139047	279931	282588
4.	1138	225896	227279	462608	463970
5.	1957	388570	388827	765829	747974
6.	2257	450209	454639	834582	833362
7.	2713	538889	538196	1047965	1030138
8.	3548	701427	701913	1320269	1326058
9.	4131	813228	814152	1563979	1567978
10.	4288	852431	852252	1600076	1575554

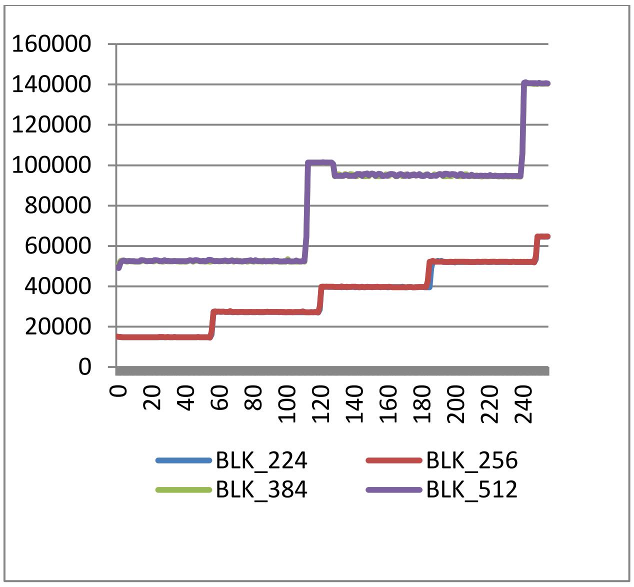

Figure 6: Graphs for cycles consumed for Short Messages.

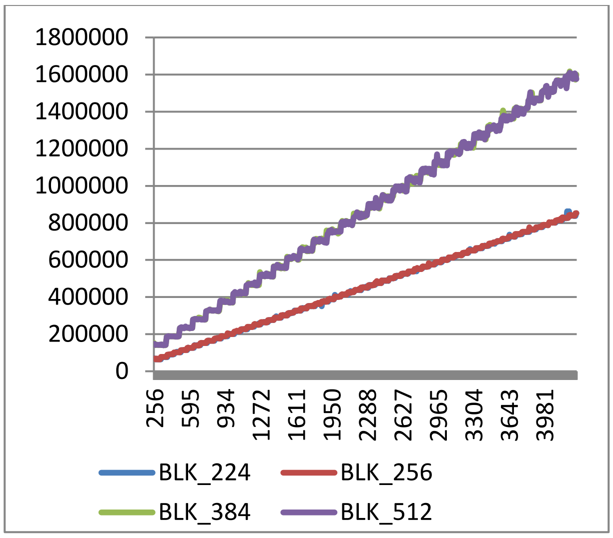

Figure 7: Graphs for cycles consumed for long messages

8.1 For Short Messages:

Blake 224 and Blake 256 are consume almost same cycles as below:

- When input is up to 57 bytes then approximately **15037** cycles consumed.

-When input is above 58 bytes and below 121 bytes the cycles consumed **1.7 times** more from initial bytes.

-When input is above 121 bytes and below 187 bytes the cycles consumed **2.4 times** more from initial bytes.

-When input is above 187 bytes and below 256 bytes the cycles consumed **3.2 times** more from initial bytes.

-When input is above 256 bytes the cycles consumed **4.0 times** more from initial bytes.

Blake 384 and Blake 512 consume almost same cycles as below:
- When input is up to 112 bytes then approximately **64204**

cycles consumed.

- When input is above 112 bytes and below 127 bytes the cycles consumed **3.72 times** more from initial bytes.

- When input is above 129 bytes and below 240 bytes the cycles consumed **2.69 times** more from initial bytes.

- When input is above 241 bytes, the cycles consumed **2.17 times** more from initial bytes.

8.2 For Long Messages:

Blake 224 and Blake 256 are consume almost same cycles and Blake 384 and Blake 512 consume almost same cycles but with linear characteristics with respect to input bytes.
- When input is 264 bytes then cycles consumed in Blake 384 and Blake 512 are **2.17 times** more from Blake 224 and Blake 256.

- When input is 1075 bytes then cycles consumed in Blake 384 and Blake 512 are **1.96 times** more from Blake 224 and Blake 256.
- When input is 4233 bytes then cycles consumed in Blake 384 and Blake 512 are **1.89 times** more from Blake 224 and Blake 256.

9. ADVANTAGES AND DISADVANTAGES OF BLAKE

The performance of Blake is fast in software and hardware manner. Simple speed/confidence is trade-off with number of rounds. There is also trade-off for hardware implementation between parallelism and throughput/area. As per algorithm design it is simple and interfaces of hashing with salt. For security point of view Blake is resistant to length-extension, generic second-pre image attacks and side-channel attacks.

As limitations, BLAKE-256 and BLAKE-512, Message length is limited to respectively 2^{64} and 2^{128} and Fixed points found in less time than for an ideal function (but not efficiently). It is also resistance to Joux's multicollisions which similar to SHA-2 .

10. CONCLUSION AND FUTURE WORK:

The method of consuming cycles is very much same for 224, 256 output bytes and 384, 512 output bytes for short as well as for long messages for Blake. For security point the applications like password hashing etc., it's better to use 256 instead of 224 and 512 instead of 384 because more byte length provide more security with the equal utilization of clock cycles.

This paper only BLAKE algorithm is evaluated from SHA-3 finalists. But there are other four SHA-3 finalists such as Grøstl , JH , Keccak , Skein algorithms which can be evaluated on ARM Cortex A8 and many other processor architectures on which the performance analysis of algorithm from ARM such as ARM Cortex A9, ARM Cortex A15 , PIC32 MIPS core processor , Power PC architecture from some other architecture venders.

Memory footprint such as RAM, ROM utilization can also be evaluated and further reduced with code optimization techniques.

11. ACKNOWLEDGMENTS

With deep sense of gratitude we express our sincere thanks to our esteemed and worthy **Dr. G Geetha** Professor & Dean, Lovely Professional University, India, for her valuable guidance in carrying out this work. At Last, we thanks to management and staff of Imbuent Technologies Pvt. Ltd. for guiding and allowing us to use their resources for this work.

12. REFERENCES

[1] Forouzan Behrouz A. (2007),"Cryptography & Network – Special Indian Edition",TataMcgraw-Hill Publishing Company Limited, New Delhi.

[2] Sobti Rajeev, Geetha G. (2012) , "Cryptographic Hash Functions: A Review", International Journal of Computer Science Issues, vol. 9, 2, pp. 461-479.

[3] Stallings William (2006). "Cryptography and Network Security", Pearson Education, India.

[4] Andreeva Elena , Mennink Bart , Preneel Bart , SkroboMarjan (2012), "Security Analysis and Comparison of the SHA-3 finalists BLAKE, Grøstl, JH, Keccak, and Skein", Proceedings of 5th International Conference on Cryptology in Africa, Ifrance, Morocco, vol. 7374, pp. 287-305.

[5] Jararweh Yaser, Tawalbeh Lo'ai, Tawalbeh Hala, Moh'd Abidalrahman (2012) , "Hardware Performance Evaluation of SHA-3 Candidate Algorithms", Journal of Information Security,vol. 3, pp. 69-76.

[6] Schwabe Peter, Yang Bo-Yin, Yang Shang-Yi (2012), "SHA-3 on ARM11 Processors",International Conference on Cryptology in Africa, Ifrance, Morocco,vol. 7374, pp. 324-341.

[7] Gibi Sunny, C Saranya (2014),"Cryptography Based On Hash Function BLAKE 32 in VLSI",

[8] Aumasson Jean-Philippe, Henzen Luca, Meier Willi, Phan Raphael C.-W (2011)., "SHA-3 proposal BLAKE" ,Submission to NIST (Round 3), 2011. [online] http://csrc.nist.gov/groups/ST/hash/sha3/Round3/submissions_rnd3.html

[9] NIST (2015) "National Institute of Standards and Technology competition Website" [online] http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[10] Sloss Andrew, SymesDominic , Wright Chris , Rayfield John (2004),"ARM System Developer's Guide Designing and Optimizing System Software", Elsevier Inc., San Francisco, CA

[11] Cortex™-A8 (2010) "Technical Reference Manual", 2006-2010 ARM Limited. [online] http://www.arm.com/

[12] Cortex™-A Series (2012) "Cortex-A Series Programmer's Guide", 2011, 2012 ARM Limited.[online] http://www.arm.com/

[13] Helmke Matthew, Hudson Andrew, Hudson Paul (2012), "Ubuntu Unleashed 2012 Edition", Pearson Education, Inc, USA.

[14] Hallinan Christopher (2010), "Embedded Linux Primer: A Practical Real- World Approach -2nd Edition", Prentice Hall ,India.

[15] SDK Manual (2013) "OpenBoard-AM3359 Software Development kit for Linux", Release 1.0 January, 2013 [online] http://www.phytec.in/

[16] Corbet Jonathan , Rubini Alessandro, Kroah-Hartman Greg (2005), "Linux Device Drivers, 3rd Edition", O'Reilly Media, Inc., Sebastopol, CA.