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, portable 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         | $2^{32}$        | $2^{32}$        | $2^{256}$       | $2^{256}$       |
| 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]:

---

International Journal of Computer Applications (0975 – 8887)  
Volume 123 – No.13, August 2015
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 = A831C6B
C6 = 510E527F  C7 = 9B05688C
C8 = 1F83D9AB  C9 = 5BE0CD1

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]:

<table>
<thead>
<tr>
<th>σ0</th>
<th>σ1</th>
<th>σ2</th>
<th>σ3</th>
<th>σ4</th>
<th>σ5</th>
<th>σ6</th>
<th>σ7</th>
<th>σ8</th>
<th>σ9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>11</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>12</td>
<td>0</td>
<td>5</td>
<td>11</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>15</td>
<td>14</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>5</td>
<td>13</td>
<td>0</td>
<td>9</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>2</td>
<td>12</td>
<td>11</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>10</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>13</td>
<td>15</td>
<td>3</td>
<td>15</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>4</td>
<td>14</td>
<td>0</td>
<td>5</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>14</td>
<td>1</td>
<td>13</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>15</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>7</td>
<td>6</td>
<td>15</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>14</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>4</td>
<td>13</td>
<td>9</td>
<td>13</td>
<td>11</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Although, these three steps are same in all Blake 224, 256,384,512 but with minor changes is explained later on this context.
2.2 Round function
When the state $v$ is initialized, the compression function iterates a series of 14 rounds. The notation of round is $r$, $G_i(a, b, c, d)$ sets $G_i$ function is performed in to three important function [7][8].

1) $+\text{ addition modulo } 2^{32} \text{ or (modulo } 2^{64} \text{).}$
2) $\oplus \text{ Boolean exclusive OR (XOR).}$
3) $\gg k \text{ rotation of } k \text{ bits towards less significant bits.}$
4) $\ll k \text{ rotation of } k \text{ bits towards more significant bit}$

Figure 2: The $G_i$ function.
The sequence $G_0$ to $G_3$ is called a column step and similarly, the last four calls $G_4$ to $G_7$ are called diagonal step.

Figure 3: $G$ functions in column step
Round Function iterated 14 times in Blake-256 and the $G$ - function repeated 112 times in Blake 256 (32-bits word).

2.3 Finalization:
When the round function is done, the new chain value $h'0$, to $h'7$ is take out from the state $v_0$ to $v_{15}$ with input of the initial chain value $h_0$ to $h_7$ and the salt $s_0$ to $s_3$:

$$
\begin{align*}
    h'0 & = h_0 \ xor s_0 \ xor v_0 \ xor v_8 \\
    h'1 & = h_1 \ xor s_1 \ xor v_1 \ xor v_9 \\
    h'2 & = h_2 \ xor s_2 \ xor v_2 \ xor v_{10} \\
    h'3 & = h_3 \ xor s_3 \ xor v_3 \ xor v_{11} \\
    h'4 & = h_4 \ xor s_0 \ xor v_4 \ xor v_{12} \\
    h'5 & = h_4 \ xor s_1 \ xor v_5 \ xor v_{13} \\
    h'6 & = h_6 \ xor s_2 \ xor v_6 \ xor v_{14} \\
    h'7 & = h_7 \ xor s_3 \ xor v_7 \ xor v_{15}
\end{align*}
$$

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 $\approx 895 \mod 1024$. 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 mobile 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 manufacturers 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 SSPEC100 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 using 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 compiler 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].

```shell
$ ifconfig eth0 192.168.1.10 up
```
On Target side issue the same command with a different IP as below

```shell
$ 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.

```bash
insmod gurpreet.ko
```
In a similar way, to remove a module, rmmod utility is used by below command.

```bash
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: Summary of cycles consumed by Blake-224 and Blake-256 for short messages:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Input Bytes</th>
<th>Blake 224</th>
<th>Blake 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>15037</td>
<td>15067</td>
</tr>
<tr>
<td>2</td>
<td>54</td>
<td>27380</td>
<td>27425</td>
</tr>
<tr>
<td>3</td>
<td>121</td>
<td>39705</td>
<td>39786</td>
</tr>
<tr>
<td>4</td>
<td>187</td>
<td>52211</td>
<td>52697</td>
</tr>
<tr>
<td>5</td>
<td>249</td>
<td>64591</td>
<td>64776</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Input Bytes</th>
<th>Blake 384</th>
<th>Blake 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>49877</td>
<td>49404</td>
</tr>
<tr>
<td>2</td>
<td>113</td>
<td>101159</td>
<td>101421</td>
</tr>
<tr>
<td>3</td>
<td>129</td>
<td>94525</td>
<td>94753</td>
</tr>
<tr>
<td>4</td>
<td>241</td>
<td>140343</td>
<td>140822</td>
</tr>
<tr>
<td>5</td>
<td>255</td>
<td>140274</td>
<td>140451</td>
</tr>
</tbody>
</table>

### Table 5: Summary of all Blake of Long Messages:

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

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...
cycles consumed.
- When input is above 112 bytes and below 127 bytes the cycles consumed \textbf{3.72 times} more from initial bytes.
- When input is above 128 bytes and below 240 bytes the cycles consumed \textbf{2.69 times} more from initial bytes.
- When input is above 241 bytes, the cycles consumed \textbf{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 \textbf{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 \textbf{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 \textbf{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 vendors.

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