International Journal of Security and Its Applications
Vol.9, No.12 (2015), pp.367-384
http://dx.doi.org/10.14257/ijsia.2015.9.12.34

Performance Comparison of Keccak, Skein, Grøstl, Blake and JH: SHA-3 Final Round Candidate Algorithms on ARM Cortex A8 Processor

Rajeev Sobti[1*] and G. Geetha[2]

[1]School of Computer Science and Engineering, Lovely Professional University, Phagwara, Punjab, 144411, India*
[2]Divison of Research and Development, Lovely Professional University, Phagwara, Punjab, 144411, India
[1]sobtirajeev@gmail.com, [2]geetha.15484@lpu.co.in*

Abstract

Taking cognizance of various attacks on MDx family of Hash function including SHA-0 and SHA-1, National Institute of Standards and Technology came up with a public competition for new Hash standard SHA-3 to augment or replace the current Hash standard SHA-2. NIST provided a reference platform (Wintel machines) and compiler (Microsoft Visual Studio) on which the submitted algorithms were to be evaluated. The five algorithms that reached the final round of SHA-3 public competition were evaluated by us on ARM Cortex A8 architecture (architecture other than the reference platform). Performance evaluation of these final round candidate algorithms were done on Cortex-A8 based OpenBoard AM335x from PHYTEC running Linux 3.2.0. The results in Cycles consumed per byte are shared in this paper.

Keywords: *Hash Functions, SHA-3, Performance Evaluation, Cortex A8, Cycles per Byte*

1. Introduction

A cryptographic hash algorithm, also known as hash function, provides a random mapping from an arbitrary length input message to fixed size message string known as message digest. Hash functions have a number of security applications like achieving integrity and authentication, implementing digital signatures and digital time stamping, generating pseudo random numbers and session keys, constructing block ciphers, indexing data in hash tables, detecting accidental data corruption as checksums etc. [1]. Different ways have been used in designing Iterated Hash functions like hash functions based on block ciphers, based on modular arithmetic, and dedicated functions. Dedicated hash functions based on cellular automata have also been suggested [2]. Majority of the hash functions have been based on Merkle Damgard structure and its improvement like Wide Pipe Iterated hash design and Hash Iterated Framework (HAIFA). For example, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 [3] are all influenced by the Merkle and Damgard's iterated hash design.

If we look at the history of hash functions, MD4 was the first widely used dedicated hash function. Developed by Rivest in 1990, MD4 started encountering attacks after which Rivest created MD5, a stronger function in 1992. However, Den Boer and Bosselaers [4] and Dobbertin [5] reported semi free start collision and pseudo collision attack on MD5.

[1] *Corresponding author. Tel. : +91-98726 88767
Email Address: sobtirajeev@gmail.com

Secure Hash Algorithm (SHA) developed by the National Institute of Standards and Technology (NIST), initially published as Federal Information Processing Standard (FIPS 180) in 1993, was also based on MDx family of hash functions. A revised version was issued as FIPS180-1 in 1995 and is generally referred to as SHA-1. In Crypto' 98, Chabaud and Joux [6] reported attacks on SHA-0. In 2002, NIST came up with revised version of the standard known as FIPS180-2 and defined three new versions of SHA with digest lengths of 256, 384 and 512 (also called SHA-256, SHA-384, and SHA-512 respectively and SHA-224 was added to these few years later). All these algorithms are collectively known as SHA-2.

This scenario up to year 2004, has been nicely summarized by John Kelsey from NIST in [7]. As per Kesley "By year 2004, we as a cryptographic community thought we knew what we were doing". We were well aware that MD4 had been broken and MD5 was known to have weaknesses as reported by Den Boer, Bosselaers and Dobbertin, but was still widely used. SHA-0 was having weaknesses and thus not used. SHA-1 was considered to be very strong. SHA-2 looked like the future and Merkle Damgard was considered a normal way to build hash functions.

The situation changed in years 2004 and 2005, after the practical attack on MDx family followed by attacks on SHA-0 and SHA-1 created doubts about the security of existing hash functions. Joux [8] showed generic multi collision attack on Merkle Damgard based hashes and reflected that cascaded hashes don't help security much. A number of attacks were reported mainly by Biham and Chen [9], Biham et.al. [10] and also by a team of researchers, led by Xiaoyun Wang from Shandong University in Jinan, China. The same team also broke HAVAL-128 and RIPEMD and SHA-1 [11-14]. Looking at the variety of hash functions attacked by this team, it looked as if their approach might explore vulnerability in all cryptographic hashes in the MDx family, including all variants of SHA. In 2006, Hoch and Shamir [15], extended the Joux's multi-collision attack and showed that a large class of natural hash functions is susceptible to a multi-collision attack, and hinted that the techniques developed here will help in creating multi collision attacks against even more complicated types of hash functions. Such a conclusion was perhaps hinting to probable attack on SHA- 2 family of hash functions. Pre-image attacks on 41 steps SHA-256 and 46 steps SHA-512 presented in [16] reduces the security margin of SHA-256 and SHA-512 also. SHA-256 and SHA-512 having 64 and 80 steps, respectively, seemed then secure but created a doubt for future. All these attacks called all the widely used hash functions into question. A question, knocking on everybody's mind, was what could be the repercussions if SHA-2 was compromised or successfully cryptanalyzed or broken? Then the world would not be left with any options because SHA-2 was the best that we had at that time.

Motivated by the big success of AES (Advanced Encryption Standard) competition that took place few years ago and forced by recent attacks on existing family of widely used hash functions, NIST decided to hold a public competition in search of a new hash standard. The idea was to have the new algorithm parallel to current SHA-2 and this new algorithm is to be used if worst case scenario occurs i.e. SHA-2 faces some serious challenge. In Nov 2007 NIST announced a public competition [17] for development of a new cryptographic hash standard named "SHA-3" to augment the hash algorithms currently specified in the Federal Information Processing Standard (FIPS) 180-4, Secure Hash Standard [3].

By October 31, 2008, NIST received sixty-four entries (in comparison to the 15 entries received in AES Competition). NIST selected fifty-one algorithms that fitted their guidelines and advanced to the first round on December 10, 2008. There was a lot of cryptanalysis and many hash functions were broken and some were found unappealing on performance parameters. Fourteen algorithms advanced to the second round on July 24,

2009. A year was allocated for the public review of the fourteen second-round candidates. Significant feedback was received from the cryptographic community. Based on the public feedback and internal reviews of the second-round candidates, NIST selected five SHA-3 finalists – Keccak [18], Skein[19], Grøstl [20], Blake [21] and JH [22], who advanced to the third (and final) round of the competition on December 9, 2010. All finalists were tweaked in the final round. BLAKE and JH increased the number of rounds, Grøstl changed the Q permutation, Keccak modified the padding technique and Skein tweaked the Key Schedule constant. Keccak was announced winner on 2nd October 2012. Keccak team made a presentation at NIST on February 2, 2013 and NIST's SHA-3 standardization plans were presented at various forums as mentioned in [7, 23-25]. NIST announced Draft FIPS 202, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions" [26] on May 28, 2014 and invited public comments on the same. However, SHA-3 standard has not been implemented yet. NIST policy on hash function [27] currently suggests no need of transiting applications from SHA-2 to SHA-3.

Intrigued by the following two points, we decided to carry out the performance evaluation of SHA-3 final round candidate algorithms on ARM Cortex architecture.

A) For SHA-3 competition, NIST announced the 'Reference Platform and Complier' [17] on which the candidate algorithms would be evaluated. Reference platform and compiler, as announced by NIST was Wintel personal computer, with an Intel Core 2 Duo Processor, 2.4GHz clock speed, 2GB RAM, running Windows Vista Ultimate 32-bit (x86) and 64-bit (x64) Edition and ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition. Because of this 'Reference Platform', there was high probability that the final round candidates would have optimized their code for the 'Reference Platform and Compiler'. However, there is a considerable domain of architecture like 8 bit or 16 bit architectures used by Smart Cards, ARM based microcontrollers and processors used in embedded and mobile devices etc. that may have skipped the attention of cryptographers of final round. It is quite possible that an apparently fast function on reference platform is not performing well on embedded system like mobile device. So an analysis of SHA-3 final round candidate algorithms on platforms other than 'Reference Platforms' is desired. In fact, NIST's notice introducing the competition [17] also invited the public to conduct similar tests and compare results on additional platforms.

B) Secondly the major finding, that has come out of the evaluations of five finalists, is that the security of these algorithms have been carefully analyzed in the last 2 – 3 years and cryptographic community does not expect serious flaws or vulnerabilities to be found in any of them. No algorithm was knocked out by cryptanalysis and different algorithms have different depth of cryptanalysis. However, performance-characteristics vary from one platform to other. The same fact has been consolidated by NIST's scientists Tim Polk, Quynh Dang and John Kelsey at multiple forums [7, 23-225].

So keeping the above two points in mind, we concentrated on evaluating performance (rather than cryptanalysis) of final round algorithms on platform other than 'Reference Platform' and tried to conclude which one among them performs better on target architecture (other than 'Reference Platform').

Organization of the paper: This paper will share the performance result of under discussion hash algorithms on ARM Cortex Application series processor. Organization of the paper is as follows. In Section 2, we have given the reason as to why we opted to evaluate the algorithms on ARM cortex A8. Section 3 introduces all the five candidate algorithms. In Section 4, we share the methodology and the tools used for evaluation and in Section 5 we present the results followed by conclusion in Section 6.

2. Selection of Target Platform and the Rationale behind It

One of the prime decisions was to look for the target platform on which these final round candidate algorithms were to be analysed. The choice of target platform was viewed as a two-step decision. In the first step, we decided to go for architecture prevalent in Embedded and Mobile platforms and in the second step we zeroed down to ARM Cortex ‘A’ series processors that are leading in Embedded and Mobile platforms. We present here the motivation and rationale behind this.

2.1. Decision – 1: Going for Embedded and Mobile Platform

The following two observations, drove us to narrow down our choice:

A) The reference platform, announced by NIST, was a general purpose system (Windows Vista Ultimate 32-bit (x86) and 64-bit (x64) Edition and ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition). However the ‘General purpose computers’ consist of hardly 1% of total computing devices. The remaining 99% comprises of Embedded and Mobile devices like mobile phones, digital TVs, mass storage controllers, smart cards, smart sensors, automotive body electronics, printers, networking devices like routers etc. The architecture of these Embedded and Mobile systems is considerably different from general purpose machines and is generally characterized by low power consumption, small size, and limited processing resources. Importantly, Embedded and Mobile systems, particularly smart phones and netbooks etc., do make use of hash functions for various security applications. So evaluating these hash functions on architecture from Embedded and Mobile segment becomes crucial.

B) Another factor that buttressed our decision is the present surge in mobile usage. In recent years the trend of mobile devices have increased considerably and people prefer to use mobile devices as compared to desktops. The statistics from various sources vindicates the above statements. For example International Trade Union’s (ITU) latest ICT Facts and Figures 2015 report [28] clearly showed exponential surge in mobile broadband penetration globally. The statistics presented in [29, 30] unambiguously reflects the surge in mobile usage and its domination over desktops. As evident, mobile usage is increasing day by day and these mobile platforms are being used to access internet, read emails, do purchases and all such things do require use of hash functions for security applications. So opting for architecture prevalent in Mobile platform was quite understandable.

2.2. Decision – 2: Zeroing Down to ARM Cortex Application Series Architecture

Once the choice narrowed down to Embedded and Mobile platforms, the decision was to select from prevalent architectures in this segment like ARM, MIPS, AVR32 etc. The final decision to select ARM architecture was based on its market dominance and performance characteristics. The following paragraphs illustrate the same:

A) ARM design cores that are based on RISC (Reduced Instruction Set Computing) architecture. ARM cores have been specifically designed to be small and reduce power consumption and have high code density. Both these features make it apt for battery operated devices and also for devices having limited on-board memory like Mobile Phones, PDAs etc. Certain features like; variable instruction cycle for specific instructions (e.g. load-store-multiple instruction), Inline barrel shifter, Thumb 16-bit and Thumb2 instruction set providing high code density, ability of core to switch between ARM and Thumb instruction state, seven different operating modes, multiple addressing modes including modes that allow direct bit shifting and conditional execution of instructions, make ARM architecture unique and also improve the performance. Multiple extensions developed by ARM like Jazelle, Security (TrustZone), SIMS and NEON technologies also give ARM an extra edge in this Embedded and Mobile market.

B) Based on our study, it has been found that ARM (Advanced RISC machines) has grown to become world's leading microprocessor IP (Intellectual Property) company, and the ARM processor portfolio covers every area of microprocessor applications, from very low-cost embedded microcontrollers, to high-performance multicore processors. ARM designs scalable and energy efficient-processors and its related technologies that are found in many of digital devices in different market segments, including Smartphones, Smart watches, Netbooks, eReaders, PDAs, Digital TVs, Home Gateways, Automotive breaking systems, Storage Controllers, Microcontrollers, Smart sensors, Servers and Networking etc. ARM core is being used by more than 1000 partners. ARM Partner companies which make chips based on ARM architecture include Apple, Atmel, Broadcom, Freescale Semiconductor, Nvidia, Samsung Electronics, ST Microelectronics, Texas Instruments and Qualcomm. Globally ARM is one of the most widely used processor architecture. Till date, more than 50 billion ARM processors have been sold and out of this 10 billion were shipped in 2013 alone. As per ARM Holdings 2013 Strategic report [31], ARM technology now reaches 75% of people in the world and ARM partners are shipping more than 2.5 billion ARM based chips every quarter. Another statistics states that ARM based chips are found in nearly 60 percent of the world's mobile devices and if the chips are laid end-to-end, they would encircle the globe a dozen time [32]. All this vindicates Steve Fuber's statement in [33] that ARM 32-bit architecture is the most widely used architecture in mobile devices, and most popular 32-bit one in embedded systems. The way ARM has dominated this segment of computing, it was quite logical to evaluate the algorithms under discussion on ARM architecture.

C) ARM processor portfolio mainly consists of **i) ARM Cortex Application processors** (Cortex A Series) – These are high performance processors for feature rich operating systems and are mainly used in Smartphones, Netbooks, Smart TVs etc. that have memory management system controlled by rich operating systems like Android, iOS or some sort of Linux flavour etc. **ii) ARM Cortex Embedded Processors** (Cortex 'M' Series) – developed primarily for microcontroller domains where need is extremely low gate count and lowest possible power consumption. Speed and deterministic behaviour is the objective of these 'M' series embedded processors . iii) **ARM Cortex Real Time Embedded Processor** (Cortex 'R' Series) **-** Objective is exceptional performance for real time applications like Automotive Braking system. Out of the above three, Cortex 'A' series processors are most commonly used in devices (like Smartphones, Netbooks, Smart TVs, Networking devices and servers) that run applications requiring use of hash functions for security concerns like achieving integrity and authentication, implementing digital signatures and digital time stamping, generating session keys etc. So our decision zeroed down to evaluating performance of algorithms on **ARM Cortex 'A' series architecture.**

3. Algorithms

3.1. Keccak

The SHA-3 winner Keccak has nothing that looks like MD4 / MD5. It is based on Sponge construction. Sponge construction is an iterated construction for building a hash function 'Keccak-F' with variable length input and arbitrary output length based on a fixed length permutation (*Keccak -f*) operating on fixed number of bits **'b'**. The building block *(Keccak-f)* uses one of the permutation out of seven permutations. The set of seven permutations are named as ***Keccak-f*[*b*]** and width b is defined as **b = 25 x 2^l** {***l*** varies from 0 to 6}

As per above relation, width **'b'** can be 25, 50, 100, 200, 400, 800 or 1600. *Keccak – f[b]* is characterized by two parameters: bitrate **'r'** and capacity **'c'** and holds the relation **b = r + c.** The permutation *Keccak – f[b]* operates on state **'a'** . The inner state **'a'** is a three-dimensional array of elements of GF (2) and is written as **a[5][5][2^l].** For *Keccak – f[1600]***,** state **'a'** will be three dimensional array a[5][5][64] and for *Keccak – f [100]*, state '**a**' will be three dimensional array a[5][5][4]. Initially, all the bits of state are initialized to zero. The input message is padded using multi-rate padding and divided into blocks of **'r'** bits each. The sponge construction then proceeds in two phases: 'Absorbing Phase' in which each block of input message is XORed with **'r'** bits of state followed by application of Keccak-*f*[b]. All this is succeeded by 'Squeezing Phase' , in which first **'r'** bits of the state are returned as output block, interweaved with application of *Keccak-f[b]*. The number of blocks is decided by the user depending on the desired hash output size. The last **'c'** bits of the state are neither affected by the input messages nor outputted during the squeezing phase.

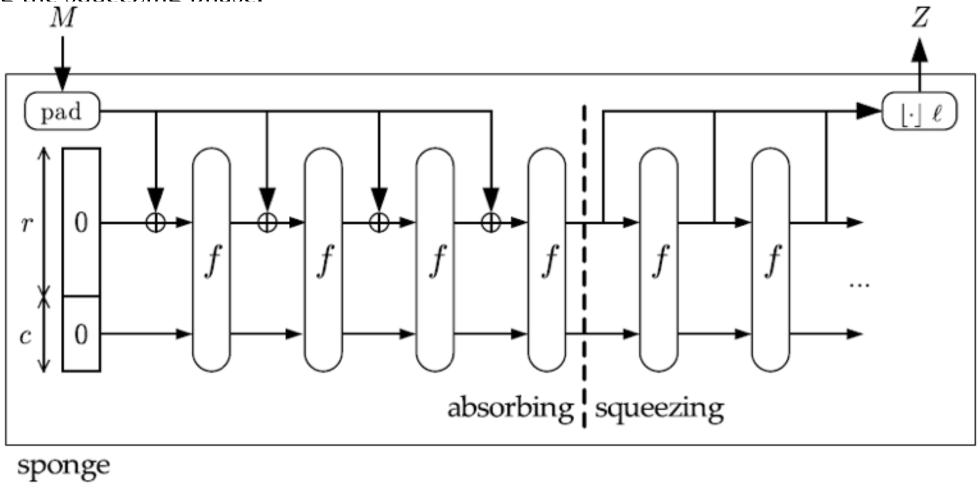

Figure 1. Keccak Sponge [18]

Each permutation *'f'* (also termed as *Keccak – f[b]*) as mentioned in the Figure 1 is an iterated permutation that make use of $\boldsymbol{n_r}$ rounds (indexed from 0 to $n_r – 1$) and each round **R** carries out multiple operation in GF(2). Details can be had from [18]. For SHA-3, NIST needed four hash sizes i.e. 224, 256, 384 and 512 bits and for these sizes, Keccak recommended the following fixed output length variants:

For output size of 224: b = 1600, r = 1152, c = 448; for output size of 256: b = 1600, r = 1088, c = 512; for output size of 224: b = 1600, r = 832, c = 768; for output size of 224: b = 1600, r = 576, c = 1024. For further details, please refer [18].

3.2. Skein

Skein is defined for three different internal state sizes – 256 bits, 512 bits and 1024 bits. However, Skein – 512 was the prime proposal and the same is evaluated in this paper as well. Even for 512 bit output, Skein makes use of 512 bit internal state which can be exploited using length extension attack. To avoid length extension attack, Skein-1024 may be used for 512-bit hash output. Skein can produce variable length hash output as desired by the user. Skein uses Tweakable block cipher - Threefish as the basic building block and UBI (Unique Block Iteration) chaining mode to process arbitrary input size to generate desired output.

Threefish makes use of three mathematical operations: XOR, Addition and Rotation (with a constant) and all operations are done on **64-bit words**. The core of Threefish is MIX function and is described in the Figure. 2 and use of the same in different rounds is shown in Figure 3. Every round of Threefish–512 makes use of four MIX functions followed by a permutation, named 'Permute' of the eight 64-bit words. A sub-key is inputted every four rounds. The rotation constant of MIX functions are chosen that are repeated every eight rounds. UBI calls multiple instances of Threefish. For details refer [19].

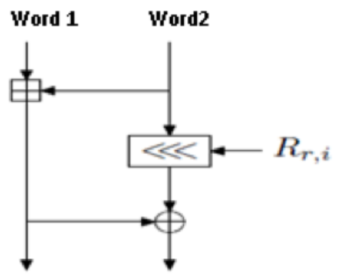

Figure 2. MIX Function of Threefish [19]

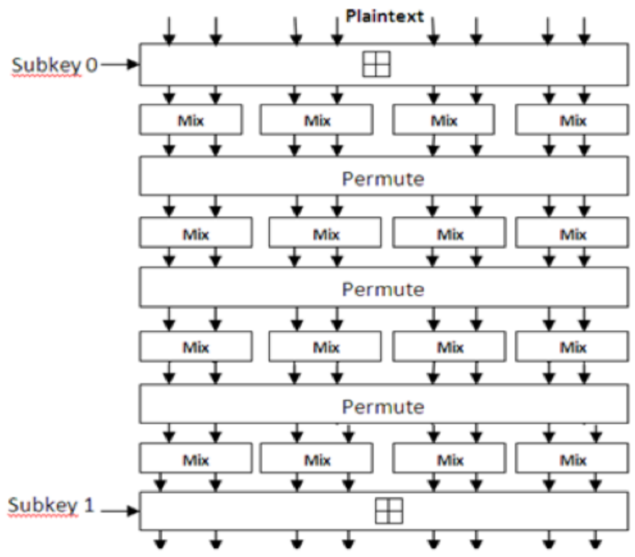

Figure 3. Four of the 72 Rounds of Threefish 512 [19]

3.3. Grøstl

Grøstl is an iterated hash function and its compression function is built from two large, distinct, fixed permutations. It is a byte oriented SP – Substitution and Permutation network and makes use of S boxes and diffusion layer similar to AES. It is based on the wide pipe design i.e. size of internal state is considerably larger than the hash output size. Grøstl can output message digest of any number of bytes from 1 to 64 i.e. 8 bits to 512 bits in steps of 8 which covers SHA-3 submission requirement i.e. to have message digests of 224, 256, 384 and 512 bits.

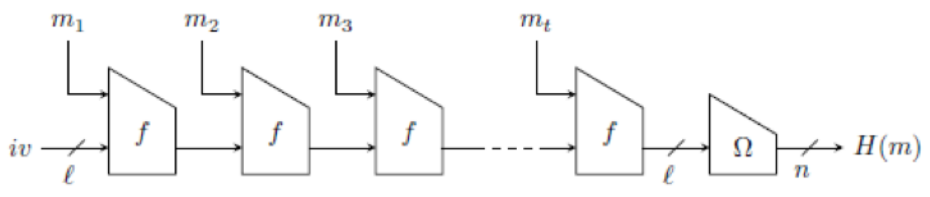

Figure 4. Working of Grøstl [20]

For output of 256 bits or less, Grøstl used the state size (and block size) of 512 bits and for higher output sizes, it used 1024 bits as state (and block) size. The input message is divided into block of 'l' bits each (512 or 1024 as stated above) and an initial value h_0 = IV is defined, and subsequently the message blocks m_i are processed as $h_i = f(h_{i-1}, m_i)$ where 'i' varies from 1 to maximum number of blocks. The Final output is truncated to the desired width in a final output transformation Ω. The Figure 4 details the structure of Grøstl.

The compression function 'f' is based on two permutation functions **P** and **Q** and can be defined as $f(h,m) = P(h \oplus m) \oplus Q(m) \oplus h$. Two types of compression functions have been defined, one for hash size up to 256 bits and the other for hash sizes of more than 256 bits. For 256-bit hash size, the state size is defined as 8x8 matrix and for larger hash size, the state matrix has 8 rows and 16 columns. Just like the AES, each round consists of four operations: Add-Round-Constant, Substitute-Bytes, Shift-Bytes and Mix-Bytes and value of constants vary for P and Q blocks. S boxes are similar to AES. For 256-bit hash sizes, a total of 10 rounds are operated and for longer hash sizes, 14 rounds are operated. The final output is obtained using $\Omega(x) = trunc_n\,(P(x) \oplus x)$ where $trunk_n$ discards all but trailing n bits of x. For details of various round operations, please refer [20].

3.4. Blake

The Blake hash function is based on HAIFA iteration mode. It operates on 32-bit word size for 256-bit hash output and works on 64-bit word size for 512-bit hash output. The Block size is of 512 bits and 1024 bits respectively for 256 and 512-bit hash value. Blake-224 is derived from Blake-256 and Blake-384 is derived from Blake-512 respectively with changed initial values. The compression function makes use of local wide pipe design and make use of salt, counter (no. of message bits processed so far) to compress each message block distinctively. As illustrated in Figure 5, the large inner state is initialized from salt, counter and initial values and this state is updated with message dependent rounds. At the end, state is compressed to return chain value for next call of compression function for another message block.

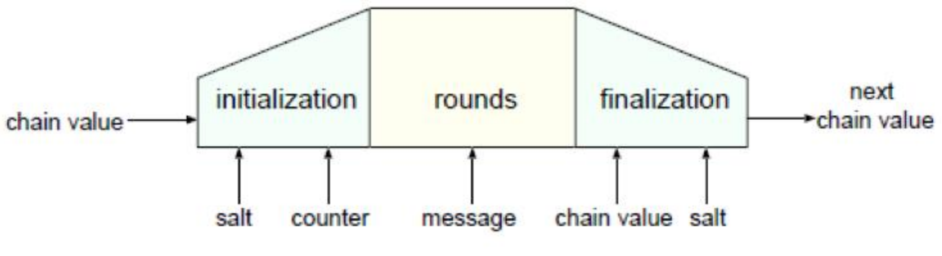

Figure 5. Local Wide Pipe of Blake [21]

The compression function takes Chain value (8 words - h_0 to h_7), Counter (2 words - t_0, t_1), Salt (4 words - s_0 to s_3) and Message block (16 words - m_0 to m_{15}) as input and generate a new chain value of eight words (h'_0 to h'_7) as output. The initial value used in Blake is same as SHA-2 and Blake also makes use of 16 constants (c_0 to c_{15}) and ten permutations. The inner state of compression function is of 16 words (32-bit or 64-bit word depending on hash size) and the same is arranged in a matrix of 4x4. The compression function of BLAKE-224/256 iterates a series of 14 rounds and BLAKE-384/512 iterates a series of 16 rounds. In each round, all the four columns of inner state are updated independently, and thereafter four disjoint diagonals are updated. While updating column or diagonal, two message words are injected according to round dependent permutation. To minimize similarity, each round is parameterized by a distinct constant. After all the rounds of compression function, a new chain value is extracted from state v_0 to v_{15} with an input of initial chain value and salt. For details, please refer [21]

3.5. JH

JH makes use of large block cipher with constant key to generate compression function and utilize generalized AES design methodology to design a large block cipher from small components [22]. The compression function structure, as given in Figure 6, compresses 512 bit message block $M^{(i)}$ and 1024 bit $H^{(i-1)}$ into 1024 bit $H^{(i)}$.

The JH hash function consists of the following steps : **(a)** Pad the message M so that it is in multiples of 512 bits, **(b)** Parse the padded message into N 512 bit blocks named as $M^{(1)}$, $M^{(2)}$, to $M^{(N)}$ and each 512 bit block is expressed as four 128 bit words e.g. $M^{(i)}_0$, $M^{(i)}_1$, $M^{(i)}_2$, $M^{(i)}_3$ are four 128 bit words of ith block **(c)** Set the initial value $H^{(0)}$ (d) Compute $H^{(N)}$ by compressing $M^{(1)}$, $M^{(2)}$, … $M^{(N)}$ iteratively as mentioned in Figure 6. **(e)** Generate the message digest by truncating $H^{(N)}$. The last 224, 256, 384 or 512 bits of $H^{(N)}$ are selected as message digest for JH-224, JH-256, JH-384, KH-512 respectively. For operations carried out by E_d and other details please refer [22].

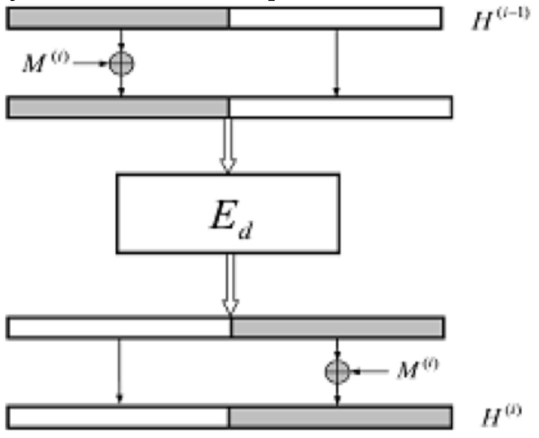

Figure 6. Compression Function of JH [22]

4. Methodology and Tools Used

4.1. Hardware and Software Used

As concluded in Section II titled 'Selection of Target Platform and the Rationale behind It', it was decided to evaluate SHA-3 final round candidate algorithms on ARM Cortex Application series architecture. From the Cortex Application series, we picked

ARM Cortex-A8 processor which was introduced in 2005 and was the first processor supporting the ARMv7-A architecture. For evaluation, we used Cortex-A8 based OpenBoard AM335x from PHYTEC [34]. The phyCORE-AM335x System on Module supports the Texas Instruments Sitara^{TM} AM335x family of processors which delivers up to 720 MHz high-performance, 512 MB DDR3 and 512 MB NAND-Flash. The Board runs Linux 3.2.0.

4.2. CPB as Performance Parameter

For evaluating the performance of various algorithms on Cortex A8 based processor, we used 'Cycles per Byte' (CPB) as a performance parameter in place of execution speed in units of time. CPB refers to number of cycles consumed by the hash function divided by the number of input bytes. 'Cycles per Byte' (CPB) is architecture oriented, and will not change with the frequency of the device used. Thus it is a better measure than execution speed.

Secondly, for computing the cycles per byte, the cycles after computation were divided by total bytes after padding e.g. if we provide an input of 100 bytes to the Grøstl - 512 algorithm it will make a single block of 1024 bits after padding because it takes an input of at least 1024 bits for generating 512 bit hash output size, so the cycles consumed were divided by 1024 rather than 100.

4.3. Hash Function as a Whole rather than Compression Phase alone

Hash functions generally have three stages i.e. Initialization, Compression and Finalization. Initialization involves the padding, initializing the internal state & parameters and generation of lookup tables etc. The compression phase is the main part and consists of multiple calls to compression function depending on the number of input blocks. In every call to compression function, the internal hash state is updated using the current state and one message block. The amount of time spent in this phase is directly proportional to the message length and accounts for the largest part of overall execution time or cycles consumed. The Finalization phase usually involves the processing of final block (including padding block), and a final call to output transformation that gives the resulting value. Many researches sometimes measure the compression phase only for benchmarking purpose as it is a major part of the total execution time. Undoubtedly, Initialization and finalization phase relatively take lesser time than compression phase. However, these phases also vary considerably with the algorithm concerned. So, in our performance evaluation, we measured the Clock cycles consumed by all the three phases rather than only the compression phase.

4.4. Computation of Cycles using System Control Coprocessor

To measure the cycle consumed, we used System Control Coprocessor CP15 available with Cortex A8. The CP15 is used to control and provide status information like cycle counts. Out of various system control registers of CP15, we used 'c9' register. 'c9' register was used with the following operation register to compute clock cycles [35]:

a) c9, Performance control monitor register (c12 with operation code 0)
b) c9, Count enable set register (c12 with operation code 1)
c) c9, Overflow Flag status register (c12 with operation code 3)
d) c9, Cycle count register (c13 with operation code 0)
e) c9, User enable register (c14 with operation code 0)
f) c9, Interrupt enable clear register (c14 with operation code 2)

To access these registers MRC and MCR instructions were used. MRC is used to read from co-processor register and MCR is used to write on co-processor register. For example, to write on Performance control monitor register, the following instructions might be used:

MCR p15, 0, <Rd>, c9, c12, 0

Here 'p15' refers to CP15 co-processor, 'Rd' refers the general register whose content will be written on c12. c9 is primary register and c12 refers Performance control monitor register. Using MRC and MCR instructions, initially 'User enable register' was set from kernel module and to do that loadable kernel module was written. Once User enable register was set, Performance monitor control register was written to enable all counters, clear overflows and program the performance monitor control register to disable divider so that counter increments after each cycle rather than after 64 clock cycles. After setting the Performance monitor control register, Cycle count register was read before and after the desired function and the difference between two values gave us the cycles used by the desired function.

4.5. Averaging the Cycle Count and Subtracting the Overhead

Two calls to a function that reads Cycle Count register [one before a function *f*() and another after function *f*()] will measure exact cycles consumed by the function *f*() as well as cycles spent in other processes or in the kernel. There is no way to restrict the measurement to a single thread. To reduce this effect, we measured cycle consumption multiple times and then calculated average to record the readings. Also the above function used to read Cycle Count Register is not free, it also has some overhead. But this overhead is fixed and we computed it multiple times on an idle system and then averaged out to find exact overhead.

4.6. Optimized 32 bit Implementation of Hash Algorithms Used

As desired for SHA-3 submission, the five finalist algorithms can produce hash output of 224, 256, 384 and 512 bits. The federal notice for SHA-3 competition [17], asked for reference implementation as well as two optimized implementation – one optimized for 32 bit platform and another for a 64 bit platform. As our target platform (Cortex A8) is a 32 bit platform, so we used 32 bit optimized submission and implemented all five SHA-3 finalists for 224, 256, 384 and 512 bit hash output. The algorithms are coded with minimum optimization as our purpose was to test the performance analysis of the code with minimum optimization on target platform (i.e. ARM platform) to get realistic results rather than optimized results which will vary according to the level of optimization of the code.

5. Results and Discussion

For each algorithm, the results were obtained for four different hash sizes i.e. 224, 256, 384 and 512 bits. For each hash size of an algorithm, we recorded results for 2554 different inputs – input values starting from 0 bit, 1 bit up to 34304 bits. The various input values chosen were same as KAT – Known Answer Test values asked by NIST [17]. The input values in KAT were classified as 'Short' and 'Long' message values and the results presented here are also categorized for 'Short' and 'Long' messages separately. The 'Short' input values range from 0 bit to 2040 bits (255 bytes) in a step of 1 bit each and 'Long' input values vary from 2048 bits to 34304 bits (256 byte to 4288 bytes) in steps of 64 bits each. So for hash size of an algorithm, 2554 inputs were given and results recorded for analysis.

After recording the results for all five algorithms for four different hash sizes (224, 256, 384 and 512 bit), we observed that the results of 224 bit hash and 256 bit hash sizes are same i.e. CPB taken by an algorithm for 224 bit hash and 256 bit hash are almost the same and this behaviour is evident in all the five algorithms. However for 384 bit hash and 512 bit hash the results are quite close but not the same as in the case of 224 and 256 bit hash. So, in this section we have presented the result graphs of all the five algorithms for 256 bits, 384 bits and 512 bit hash sizes but not for 224 bits.

5.1. Results for Short Messages

First, we present the result for Short messages. Figure 7, 8, and 9 represent the result. The major observations are:

5.1.1. Blake, Keccak and Skein are Competing Closely: For Short messages (i.e. up to 255 bytes); performance of Blake, Keccak and Skein is considerably close. A) **For 224 and 256 bit hash output**, Blake stands out as the most efficient algorithm, closely followed by Keccak. Skein is a bit far at number three. B) However **for 384 bits output**, Keccak outperforms Blake. For shorter messages, Blake's performance is better than Skein but as message size increases, Skein starts improving and performs almost as good as Blake. C) **For 512 bits hash output**, these three algorithm are quite close. For shorter messages, Keccak is slighter better than Blake followed by Skein. However as the message size increases, performance of all algorithms come quite close to each other.

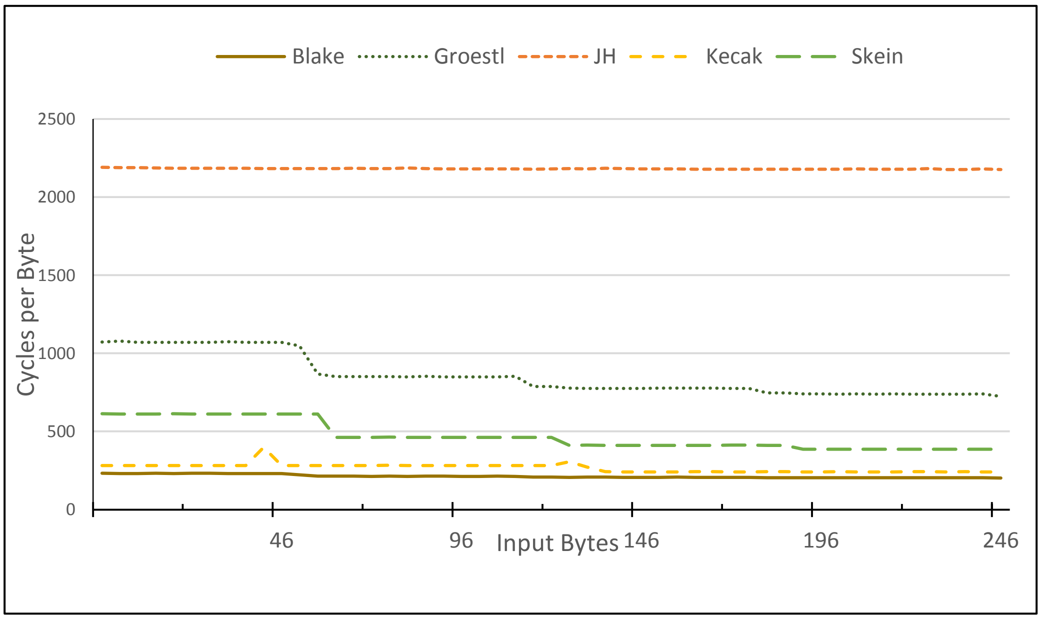

Figure 7. Results for 256 bit Hash (Short Messages)

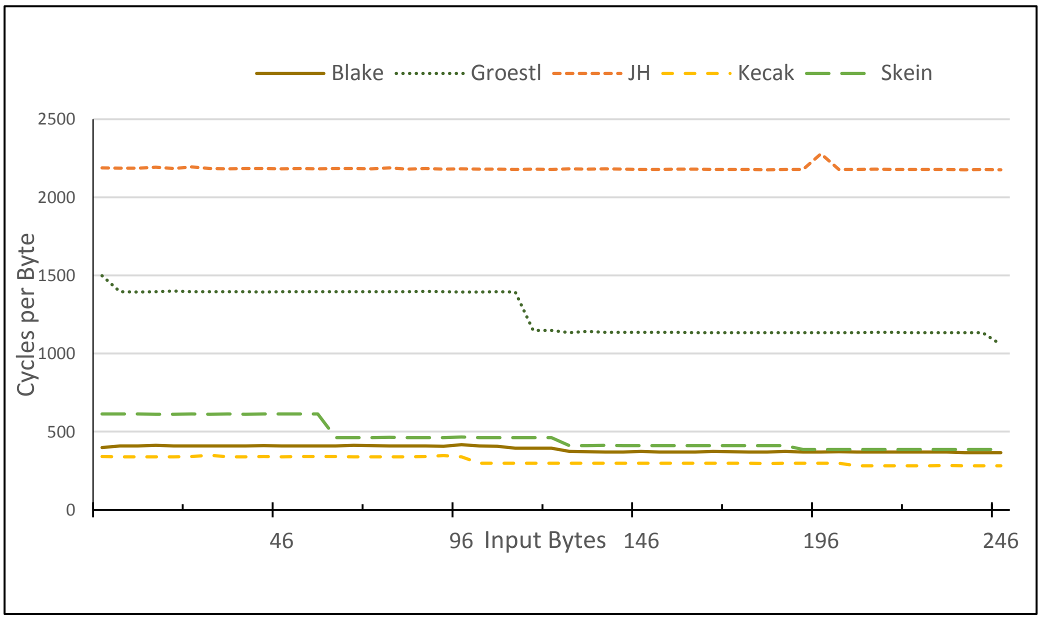

Figure 8. Results for 384 bit Hash (Short Messages)

5.1.2. Grøstl at No. 4 and JH at Last: For all output sizes (224, 256, 384 and 512 buts), **Grøstl** stands at No. 4 and JH at the last. For 224 and 256 bits output, on an average, Grøstl takes approximately **1.8 times more** clock CPB than No. 3 performer and for 384 and 512 bit output, Grøstl takes about **2.6 times more** clock CPB than No. 3 performer. For all output sizes (224, 256, 384 and 512 buts), **JH** stands at No. 5. For 224 and 256 bits output, on an average, JH takes approximately **2.5 times more** clock cycles than No. 4 performer and for 384 and 512 bit output, JH takes about **1.7 times more** clock cycles than the other 3 performers.

5.1.3. CPB Improves with Input Size: CPB consumed by an algorithm starts improving as the input size increases. This is observed in all algorithms except in case of JH.

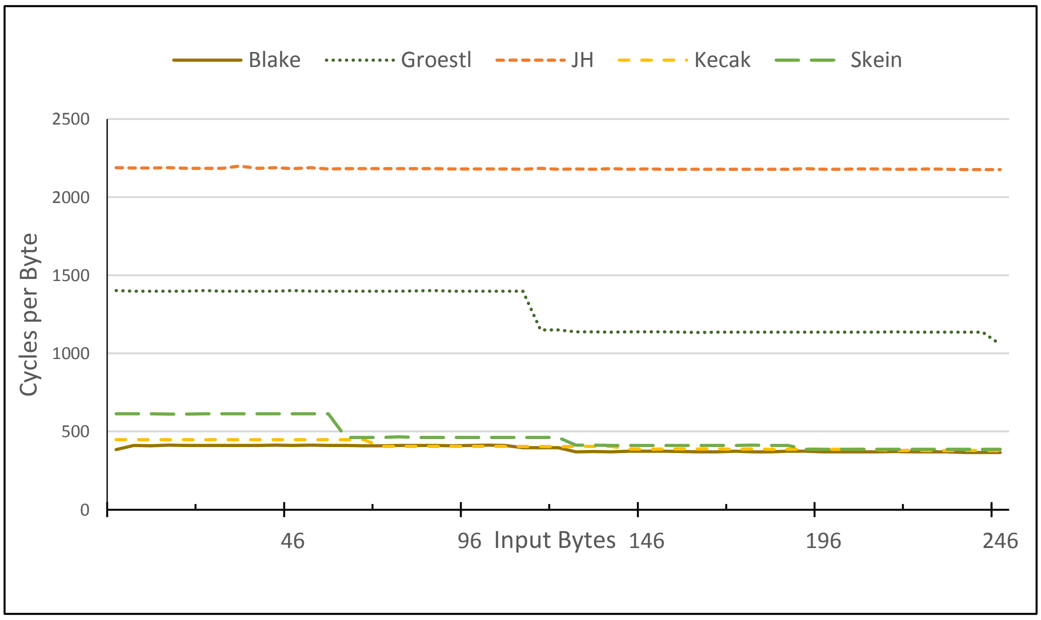

Figure 9. Results for 512 bit Hash (Short Messages)

5.2. Results for Long Messages

For Long messages (256 to 4288 bytes) results are shown in Figure 10, 11 and 12. The results are elaborated in similar fashion as done for Short messages:

5.2.1. Blake, Keccak and Skein are Competing Closely: For Long messages also; Blake, Keccak and Skein compete quite closely. A) **For 224 and 256 bit output**, Keccak and Blake performs almost equal and stands out from other algorithms. Skein is closely at number 3. B) **For 384 bit output**, Skein narrowly outperforms Blake, but Keccak is slightly better than Skein. C) **For 512 bit output**, Skein narrowly overtakes Keccak also and stood at No. 1. Keccak at No. 2 and Blake at No. 3. Grøstl and JH is at No. 4 and 5 in this case as well.

5.2.2. Grøstl at No. 4 and JH at No. 5: Just like Short messages, for all output sizes (224, 256, 384 and 512 buts), **Grøstl** stands at no. 4 and JH at 5**. For 224 and 256 bits output**, on an average, Gorestl takes approximately **2 times more** clock CPB than number 3 performer and **for 384 and 512 bit output**, Grøstl takes about **2.5 times more** clock CPB than No. 3 performer. **For 224 and 256 buts output**, on an average, JH takes approximately **3.35 times more** clock cycles than No. 4 performer and **for 384 and 512 bit output**, JH takes about **2.35 times more** clock cycles than No. 3 performer.

5.2.3. Change in CPB with Input Size: Just like Short Messages , in case of Long messages also the ‘Cycles per Byte’ consumed by an algorithm improves (reduces) as the input size increases. However, this change is less evident compared to change observed in ‘Short’ messages. This trend is considerably visible in Grøstl and least visible in JH. For other three algorithms CPB do reduce but compared to Grøstl the change is leser.

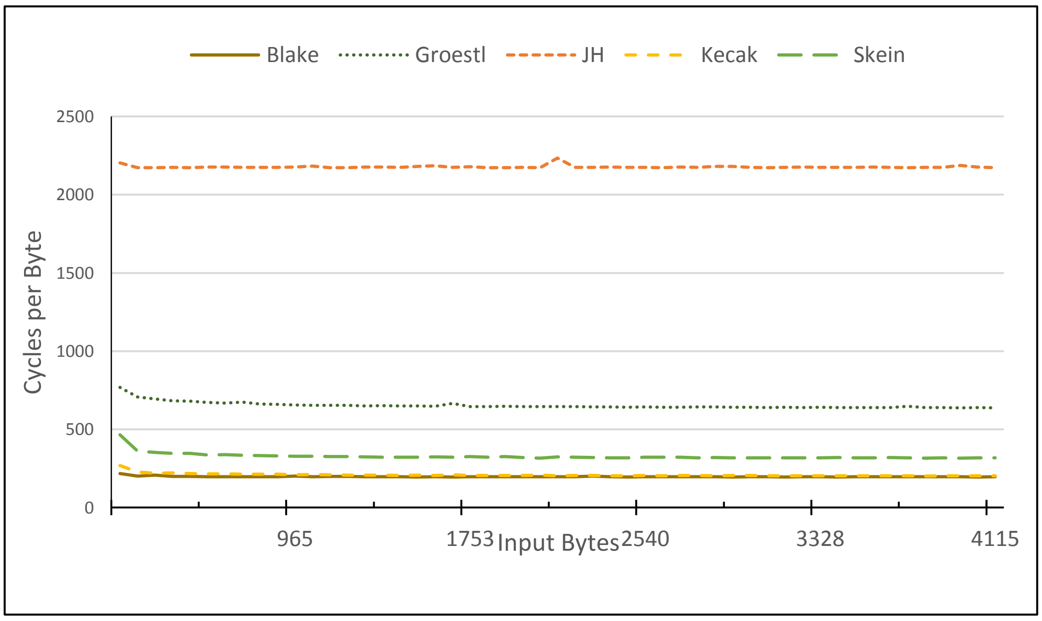

Figure 10. Results for 256 bit Hash (Long Messages)

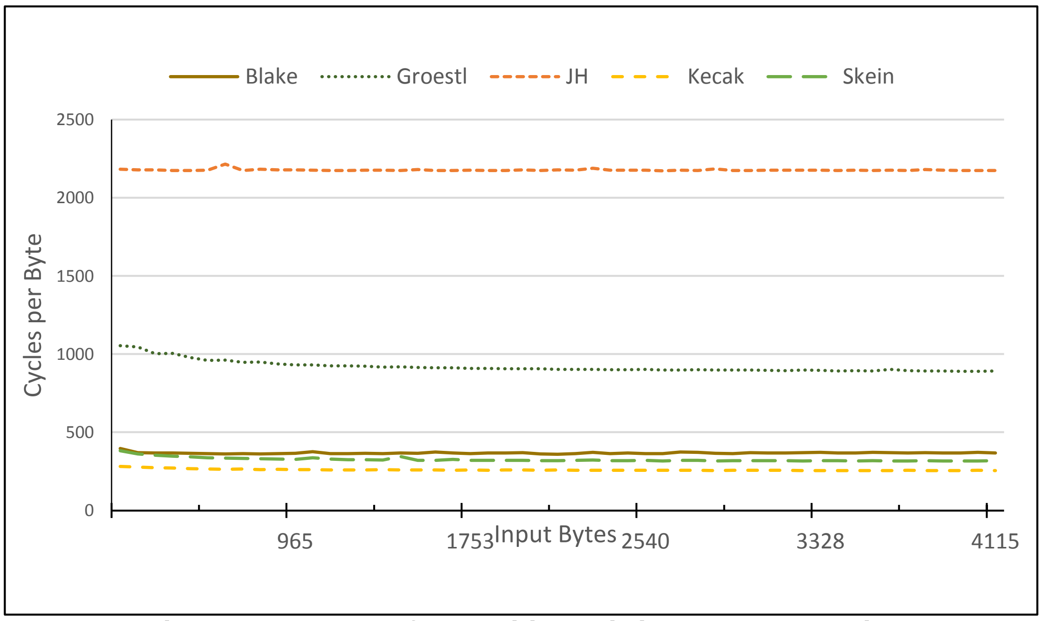

Figure 11. Results for 384 bit Hash (Long Messages)

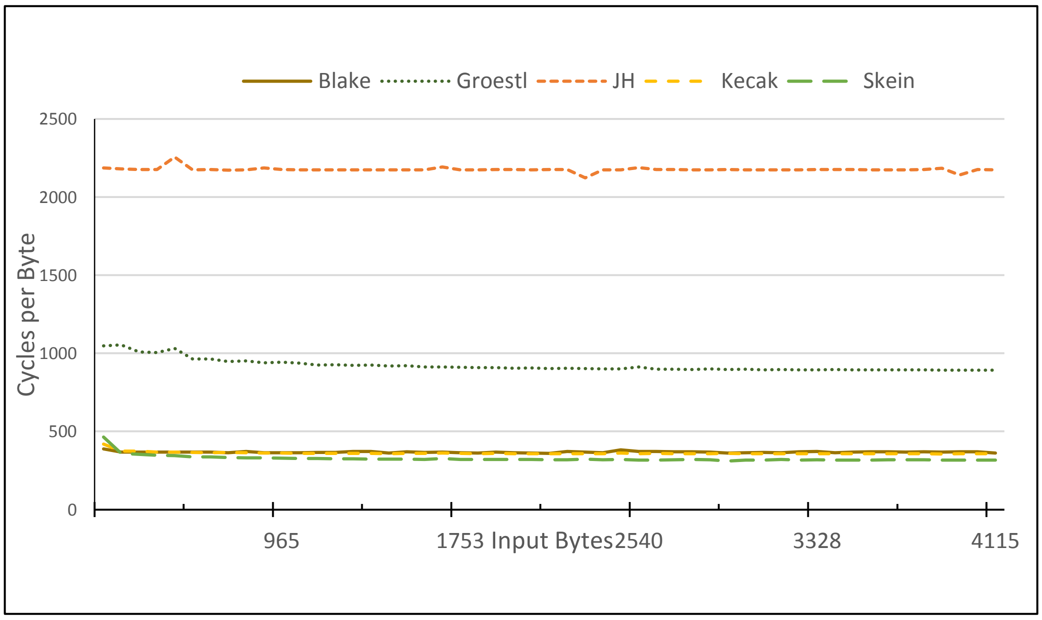

Figure 12. Results for 512 bit Hash (Long Messages)

5.3. Few Common Observations for Short as well as Long Messages

A) As we increase message digest from 224/256 bit to 384 / 512 bits, the cost in terms of CPB increases in Keccak, Blake and Grøstl i.e. more cost for better security margins but Skein is better in this term as for Skein CPB does not increase, as we increase the size of message digest from 224/256 bits to 384/512 bits.

B) As the input size increases, the CPB decreases for all algorithms except JH. This fact was evident from graphs in previous figure 7 to 12 also.

C)Certain spikes are also visible in the graphs and this is on account of some system interrupts that could have happened while recording the clock cycles on our target board.

6. Conclusion and Future Work

For security applications that require use of Hash functions in ARM Cortex A8 based devices, Grøstl and JH are not a good choice. Skein is a good option to use for long messages because the cost in terms of Cycles per Byte does not increase as we work on bigger message digest of 512 bits (rather than 224/256 bit for better security margins) and Skein is also the fastest of the lot for long messages producing 512 bit output. For short messages, Blake is also a good option as it outperfoms Keccak for 224/256 and 512 bit hash length.

Similar comparsion was carried out on Cortex M3 platform also [36] and the reults were different. As future work, it will be intersting to analyse and reason out performnce of these algorithms after optimization on other ARM architectures like Cortex A9 / A15 etc. or ‘M’/‘R’ series architectures.

Acknowledgements

The authors thank SERSC Korea branch for providing financial support towards publishing this research paper

References

[1] R. Sobti and G.Geetha, “Cryptographic Hash Functions: A Review”, IJCSI International Journal of Computer Science Issues., vol. 9, no. 2, **(2012)**, pp. 461-479.
[2] J.-C. Jeon, “Analysis of Hash Functions and Cellular Automata Based Schemes”, International Journal of Security and Its Applications., vol. 7, no. 3, **(2013),** pp. 303-316.
[3] National Institute of Standards and Technology, “FIPS PUB 180-4, Secure Hash Standard (SHS)”, US Department of Commerce, Washington D. C., **(2012)** March.
[4] B.D. Boer and A. Bosselaers, “Collisions for the compression function of MD5”, Proceedings of the Advances in Cryptology - EUROCRYPT '93, Workshop on Theory and Applications of Cryptographic Techniques, Lofthus, Norway, **(1993)** May 23-27.
[5] H. Dobbertin, “Cryptanalysis of MD5 compress”, presented at the rump session of EUROCRYPT’96, **(1996)** May.
[6] F. Chabaud and A. Joux, “Differential Collisions in SHA-0”, Proceedings of the Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference Santa Barbara, California, USA, **(1998)** August 23-27.
[7] J. Kelsey, “SHA3 Past, Present, and Future”, Workshop on Cryptographic Hardware and Embedded Systems CHES 2013, Santa Barbara, California, USA, **(2013)**.
[8] A. Joux, “Multi-collisions in Iterated Hash Functions. Application to Cascaded Constructions”, Proceddings of the Advances in Cryptology – CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, **(2004)** August 15-19.
[9] E. Biham and R. Chen, “Near-Collision of SHA-0”, Proceedings of the Advances in Cryptology – CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, **(2004)** August 15-19.
[10] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, “Collisions of SHA-0 and Reduced SHA-1”, Proceedings of the Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, **(2005)** May 22-26.
[11] X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, “Cryptanalysis of the Hash Functions MD4 and RIPEMD”, Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, **(2005)** May 22-26.
[12] X.Wang, D. Feng, X. Lai, and H. Yu, “Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”, IACR Cryptology ePrint Archive, **(2004)**, pp. 199.
[13] X. Wang, H. Yu and Y. L. Yin, “Efficient Collision Search Attacks on SHA-0”, Proceedings of the Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, **(2005)** August 14-18.
[14] X. Wang, Y. L. Yin and H. Yu, “Finding Collisions in the Full SHA-1”, Proceedings of the Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, **(2005)** August 14-18.
[15] J. J. Hoch and A. Shamir, “Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions”, Proceedings of the 13th International Workshop, FSE 2006, Graz, Austria, **(2006)** March 15-17.
[16] Y. Sasaki, L. Wang and K. Aoki, “Preimage Attacks on 41-Step SHA-256 and 46-Step SHA-512”, IACR Cryptology ePrint Archive, **(2009)**, pp. 479.
[17] Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family, NIST, Federal Register / Vol. 72, No. 212 / Friday, November 2, 2007 / Notices, [online]. Available: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf, **(2007)**.
[18] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, “The Keccak Reference”, Submission to NIST (Round 3), [online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html, **(2011)**.
[19] B. Schneier, N. Ferguson, S. Lucks, D. Whiting, M. Bellare, T. Kohno, J. Walker and J. Callas, “The Skein Hash Function Family,” Submission to NIST (Round 3), [online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html, **(2011).**
[20] 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 (Round 3), [online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html, **(2011).**
[21] J. P. Aumasson, L. Henzen and W. Meier, “SHA-3 proposal BLAKE”, Submission to NIST (Round 3), [online]: Avialble: http://csrc.nist.gov/groups/ST/hash/sha -3/Round3/submissions_rnd3.html, **(2011)**
[22] H. Wu, “The Hash Function JH”, Submission to NIST (Round 3), [online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html, **(2011)**.
[23] Q. Dang, “SHA-3 Update”, IETF 86, **(2013)** March.

[24] B. Burr, "SHA3 Where we've been, where we're Going (update to RSA 2013 talk)", DIMACS Workshop on Current Trends in Cryptography, **(2013)** May.
[25] Q. Dang, "NIST Draft FIPS 202: SHA-3 Permutation- Based Hash Standard-Status", IETF 87, **(2013)** August.
[26] National Institute of Standards and Technology, "Draft FIPS 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", **(2014)** May.
[27] NIST policy on Hash functions, [online], Available: http://csrc.nist.gov/groups/ST/hash/policy.html, **(2015)** August.
[28] ICT Facts and Figures 2015 , [online]. Available: http://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx, **(2015)**.
[29] The Mary Meeker Internet Trends 2015 Report, [Online]. Available: http://www.kpcb.com/internet-trends, **(2015)** May.
[30] R. Murtagh ,'Mobile Now Exceeds PC: The Biggest Shift Since the Internet Began', [online]. Available: http://searchenginewatch.com/sew/opinion/2353616/mobile-now-exceeds-pc-the-biggest-shift-since-the-internet-began , **(2014)** July.
[31] ARM Holdings plc Annual report – 2013, [Online]. Available: http://financialreports.arm.com, **(2013)** March.
[32] S. Murry, 'ARM's Reach: 50 Billion Chip Milestone', [Online]. Avaialble: http://www.broadcom.com/blog/chip-design/arms-reach-50-billion-chip-milestone-video/, **(2014)** .
[33] J. Fitzpatrick, "An interview with Steve Furber", Communications of the ACM, vol. 54, no.5, **(2011)**, pp. 34–39.
[34] PHYTEC Embedded Pvt. Ltd., "OpenBoard-AM3359 Software Development kit for Linux", [online]. Available: http://www.phytec.in/products/sbc/openboard-am335x.html, **(2013)** January.
[35] Cortex™-A8 Technical Reference Manual, 2006-2010 ARM Limited. [online]. Available: http://www.arm.com/, **(2010).**
[36] R. Sobti, G. Geetha and S. Anand, "Performance Comparison of Grøestl, JH and BLAKE - SHA-3 Final Round Candidate Algorithms on ARM Cortex M3 Processor", Proceedings of the International Conference on Computing Sciences (ICCS '12), Jalandhar, Punjab, India, **(2012)** September 14-15.

Authors

Rajeev Sobti, he is heading School of Computer Science and Engineering, Lovely Professional University, India. He has over 17 years of experience in the industry, teaching and research. His research interests include Cryptography and Computer System Architecture. He has been a member of Consultant Board and Manuscript reviewer for Books on Discrete Mathematics, Operating System from Pearson Education (Singapore) PTE LTD.

G. Geetha, she is heading Division of Research and Development, Lovely Professional University, India. She has nearly two decades of experience in industry, teaching and research. Her research interest includes Cryptography, Information security and Image Processing. She has authored more than 50 refereed research papers. She serves as Editorial Board member and reviewer in various Journals and Conferences. She is an active member of various professional organizations like ISCA, ISTE, CRSI etc.