EFFICIENT IMPLEMENTATION OF SECURE HASH ALGORITHM (SHA-3) CANDIDATE SKEIN USING CAVIUM NETWORKS MULTIPROCESSOR PLATFORM

Submitted By

Aisha Malik

2010-NUST-MS-PhD-Elec(Comm-N)-05

Supervisor:

Dr. Arshad Aziz

Thesis Submitted In partial fulfillment to the requirements for the award of
the degree of Master of Science in Electrical Engineering
With Specialization in Communications

at the

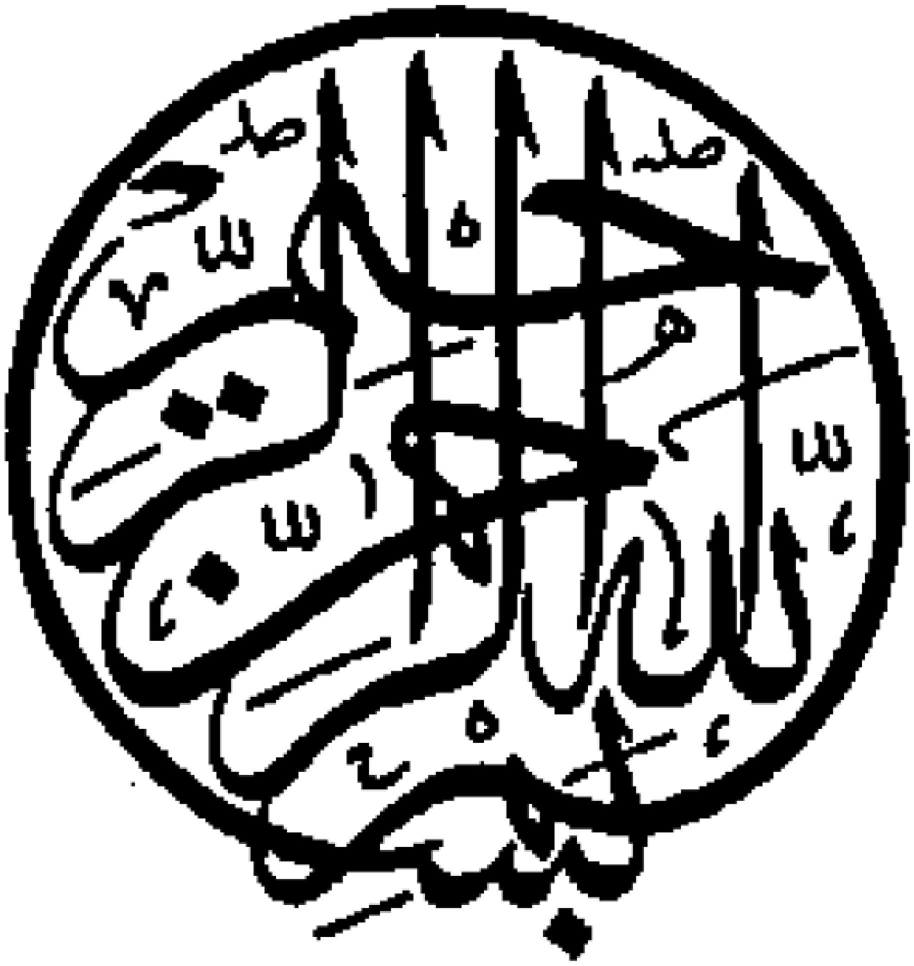
Department of Electronics and Power Engineering

Pakistan Navy Engineering College, Karachi

National University of Sciences and Technology,

H-12, Islamabad, Pakistan

December 2012


بسم الله الرحمن الرحيم

Dedicated to my loving family and friends

ACKNOWLEDGMENTS

All praise is to Allah, who bestowed upon me with the opportunity to acquire higher education. Every moment of my life is an utter depiction of his mercy. He has showered upon me the blessings more than I deserved.

This work was a critical challenge because of very little help available for the platform, lack of programming skills and the use of Linux for the first time. All these were required to work on this research that I didn't have any background of. It is with Dr. Arshad's continuous support that kept me motivated. It was an honor to work with him. I would to like to thank my GEC members, Dr. Pervez Akhtar, Dr. Athar Mahboob and Dr. Khawaja Bilal Mehmood for being cooperative through their busy schedules.

I would like to pay my gratitude to Abdul Qadeer, Asad Ather, Ayaz Khan and all those who shared their knowledge with me. Their help came to me as a mercy from Almighty Allah through toughest times. I am grateful to all of them for not turning me down when I approached them for guidance. It has improved my faith in Allah.

I am thankful to my youngest sister who has been with me at all times. Last but not the least I am extremely grateful to my parents without whom I wouldn't have been able to achieve any of the goals successfully.

ABSTRACT

Hash algorithms are one of the cryptographic primitives and play a key role in the security of every day applications that require data encryption, message authentication, message integrity and non-repudiation. The current Secure Hash Standard (SHS) followed by the IT industry worldwide is defined by National Institute of Standards and Technology (NIST). However, recent developments in cryptanalysis on SHA-1 and SHA-2 have rendered SHS's security at stake. As a response, NIST organized a public hash function competition in 2007. Since the announcement of third round results, the five candidates competing for the SHA-3 title are under continuous public evaluation from security, performance, flexibility and simplicity perspective.

Skein is one of the five candidates that are being subjected to test analysis and performance evaluation on various software and hardware platforms. This work investigates the performance of sequential and parallel processing of Skein on two platforms that have not been explored yet: Intel core-i5 2450M processor and Cavium Networks CN5860 OCTEON Plus processor. Conventional Skein has been implemented for its evaluation using single core. However, to explore its performance using multiple cores, parallel processing algorithm has been designed and implemented on the two platforms using multithreading. Hash tree structure has been used for data independency.

TABLE OF CONTENTS

LIST OF FIGURES 3

LIST OF TABLES 4

1 INTRODUCTION 5

1.1 Applications: 6

1.2 Scope of Thesis: 7

1.3 Chapter Organization: 7

2 SKEIN – SHA 3 CRYPTOGRAPHIC HASH ALGORITHM 8

2.1 Threefish: 9

2.1.1 Mix and Permutation: 9

2.1.2 Key schedule: 12

2.2 Unique Block Iteration: 13

2.3 Tree- hashing: 15

3 LITERATURE REVIEW 18

3.1 Hardware Implementations: 18

3.2 Software Implementations: 19

4 TARGET PLATFORMS 21

4.1 The Intel core-i5 processor: 21

4.2 OCTEON Plus processor: 21

4.2.1 Simple Executive 24

4.2.2 SMP Linux 24

5 DESIGN AND IMPLEMENTATION 25

5.1 Design Methodology 25

5.1.1 Sequential processing of Skein 25

5.1.2 Parallel processing of Skein 26

5.2 Implementation: 28

5.2.1 Intel Core i5 28
5.2.2 OCTEON evaluation board CN5860 28
6 PERFORMANCE RESULTS 40
7 CONCLUSION AND FUTURE WORK 43
Bibliography 44

LIST OF FIGURES

Figure 2.1: Matyas-Meyer-Oseas Construction diagram 8
Figure 2.2: First four rounds of Threefish-256 block cipher 10
Figure 2.3: The MIX function 11
Figure 2.4: Tweak fields 12
Figure 2.5: UBI Chaining Mode of message M 14
Figure 2.6: UBI stages [7] 15
Figure 2.7: A Merkle tree structure [8] 16
Figure 2.8: Skein Tree structure for different Tree parameters [8] 17
Figure 4.1: OCTEON Plus Evaluation board Block diagram [4] 22
Figure 4.2: Different runtime environment choices 23
Figure 5.1: Hash-Tree model for 16 core parallel processing 27
Figure 5.2: The CN5860 OCTEON Evaluation board [4] 33
Figure 5.3: Hardware configuration –Connections to Development target 33

LIST OF TABLES

Table 2.1: Number of rounds for different Skein variants 9
Table 2.2: Rotation constants for each Nw 11
Table 2.3: Values for word permutation..12
Table 2.4: tweak field values 13
Table 2.5: Type field values 14
Table 2.6: Configuration string field values 15
Table 2.7: Hash-Tree mode parameters 16
Table 5.1: Layout of 1GB compact flash 31
Table 6.1: Sequential Processing of Skein-256 on two platforms 40
Table 6.2: Parallel processing of Skein-256 41
Table 6.3: Performance comparison in terms of throughput...42

1 INTRODUCTION

Rapid advancement in technology has completely transformed the way we communicate today. Modern technologies, on one hand, have made communication quick and instantaneous while on the contrary they have made it insecure and vulnerable to various threats as well. Trade-offs and compromises are made when dealing with such situations but in communication, security aspect cannot be compromised at any cost. When communication takes place via any medium, the major security concerns are:

1. The message communicated is not being overheard by an intruder. If, in any case, a trespasser manages to hear the conversation, he must not be able to decrypt it. This can be prevented by using encryption and decryption algorithms.

2. Even if an intruder is successful to eavesdrop, he should not be able to modify the message before it reaches the intended recipient.

3. An intruder or opponent must not succeed in faking the sender's identity or pretend to be the originator of the message.

4. In a situation where the second party is the opponent, he must not have any way of refuting the delivery or transmission of the message or that any communication was taking place. Digital signatures are a means to address this issue.

Thus, in technical terms, a security system or cryptographic system is complete if it caters to our security concerns of confidentiality, Integrity, Authenticity and Non-repudiation. Cryptographic algorithms that address above security issues are called cryptographic primitives. Cryptographic Hash algorithms are the key primitives to security algorithms and form the foundation for almost every security protocol used in information infrastructure. These primitives when combined together form a cryptographic system. The input to a simple hash function is variable argument known as Message. The output it returns is of fixed size called Message Digest. For hash function to qualify as a Cryptographic hash function, additional criteria need to be satisfied.

First, given the message digest, it must be impractical to calculate the input message that generated that digest. This is known as Pre-Image Resistance. Second, given one fixed input message, it must be impractical to find another input message that results in the same message digest as that of the first input message. This property is known as second pre-image resistance. Third, it must be computationally impractical to find two arbitrarily input messages that result in the same message digest. This property is known as collision resistance. Last, its performance must be practically efficient [1].

In recent years, cryptanalysts have successfully attacked hash algorithms that are widely used by the IT industry, thus rendering the industry's integrity at stake. Collisions have been found in MD4, MD5 and SHA-0 [2] and published against SHA-1. This appealed National Institute of Standards and Technology (NIST) to review and evaluate the status of their currently approved hash algorithms. A thorough evaluation led to the decision of developing a new standard hash algorithm through a public competition similar to the one held for the development of Advanced Encryption Algorithm (AES). NIST revised its requirements and evaluation criteria and officially announced the public competition on 2nd November, 2007 for the creation of a New Cryptographic Hash function that would be entitled as SHA-3 family. Since then, the competition has progressed through three rounds. NIST accepted sixty-four submissions by 31st October, 2008, out of which fifty-one advanced to the initial round in December, 2008. The entries reduced to fourteen in number in the second round in July, 2009. The third round resulted in five final candidates to compete for the SHA-3 title. The competition is scheduled to end in 2012 with the selection of a winning hash function that would be entitled as SHA-3 [3]. The five finalists are Keccak, JH, Grostl, BLAKE and Skein.

1.1 Applications:

Cryptographic Hash algorithms have wide applications in the IT industry. They can be used for the purpose of constructing digital signatures, message authentication codes and secure password log – in [3]. They also act as a prime building block in authentication protocols and are employed in the development of encryption algorithms.

1.2 Scope of Thesis:

The primary purpose of this thesis work is to exploit a novel multiprocessor platform Cavium Networks OCTEON Plus that is available in the college since 2010. The resource has been availed to make a software implementation of Skein on the untested platform for experimental purposes. A second multiprocessor platform, Intel Core-i5 has also been used for the software implementation of Skein using C language. Intel core-i5 is in the market for quiet long but is still unproven for Skein performance whereas OCTEON is a novel platform in the market that is yet to be explored for its multiprocessing features [4].

1.3 Chapter Organization:

CHAPTER 1 provides with an introduction to the security concerns in communication and the basis of cryptographic hash functions. It then gives an overview of applications of hash function and the scope of Thesis. CHAPTER 2 elaborates on the subject Hash function algorithm Secure Hash Algorithm (SHA-3) candidate Skein, discussing its core components in detail. CHAPTER 3 discusses important contributions to the performance evaluation of Skein since its publication. This includes both hardware and software implementations. CHAPTER 4 is a brief description of Intel's core-i5 and Cavium Networks CN5860 OCTEON Plus platform relevant to thesis work. CHAPTER 5 describes the design methodology used for the sequential and parallel implementation as well as the step by step implementation procedure of the algorithm on the two platforms. In CHAPTER 6, the results obtained from the work are presented. CHAPTER 7 concludes the thesis work providing with future work to improve the acquired accomplishment.

2 SKEIN – SHA 3 CRYPTOGRAPHIC HASH ALGORITHM CANDIDATE

Skein is among the final candidates that have reached to the final round of NIST SHA-3 cryptographic hash function competition. Skein forms a family of cryptographic hash functions with three standard variants: Skein-256, Skein-512 and Skein-1024 where 256, 512 and 1024 are the internal state sizes in bits as explained in section 2.1. Skein consists of three elements, Threefish, UBI and an Optional argument system [5].

Threefish is the core of Skein hash function. It is a tweakable block cipher based on Matyas-Meyer-Oseas (MMO) construction as in figure 2.1 with additional features of Unique Block Iteration (UBI) and Optional Argument System. An MMO construction takes a message M_i of n bits as an input plaintext P to the block cipher and the last output hash value H_{i-1} as the key K and gives a hash value H_i for the given M_i as its hash value using

$$H_i = E_{H_{i-1}}(M_i) \otimes M_i \qquad (1)$$

where E can be any block cipher. Below is a description of two of the Skein elements.

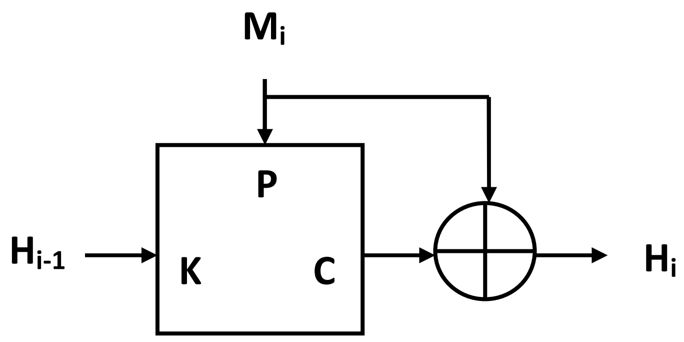


Figure .: Matyas-Meyer-Oseas Construction diagram

2.1 Threefish:

The core of Skein is Threefish, a Tweakable block cipher. It is a compression function that takes fixed size blocks of N_b bits as input. It is defined for three standard block sizes 256 bits, 512 bits and 1024 bits according to the NIST requirements [6]. These are known as the internal state sizes. Additionally a block cipher key IV and a tweak T are also required as inputs to the compression function. The key is the same size as the input message block whereas tweak is always a 128 bit value.

Threefish is sequential in nature. It performs the MIX and PERMUTE operations for a number of rounds. For Skein-256 and Skein-512, the number of rounds $N_r = 72$ and for Skein-1024, $N_r = 80$ as in Table 2.1. Figure 2.3 shows a detailed working of Threefish block cipher for Skein-256. The message block which in case of Skein-256 is 256 bits, is further divided in to unsigned 64 bit words N_w. These inputs are initially added with the block cipher key also known as initialization variable which is also in 64 bit words.

Table .: Number of rounds for different Skein variants

Block/key Size	**Number of Words N_w**	**Number of Rounds N_r**
256	4	72
512	8	72
1024	16	80

2.1.1 Mix and Permutation:

A MIX function operates on two 64 bit words x_0 and x_1. Within each MIX function are the operations of simple addition, left-shift-rotate and XOR. The bits of input are shift-rotated to the left by a constant that is different for N_w as given in Table 2.2. The MIX function returns two 64 bit words using the following equations:

$$y_0 = (x_0 + x_1) \bmod 2^{64} \qquad (2)$$

$$y_1 = (x_1 <<< (R_{(d \bmod 8),j}) \otimes y_0) \qquad (3)$$

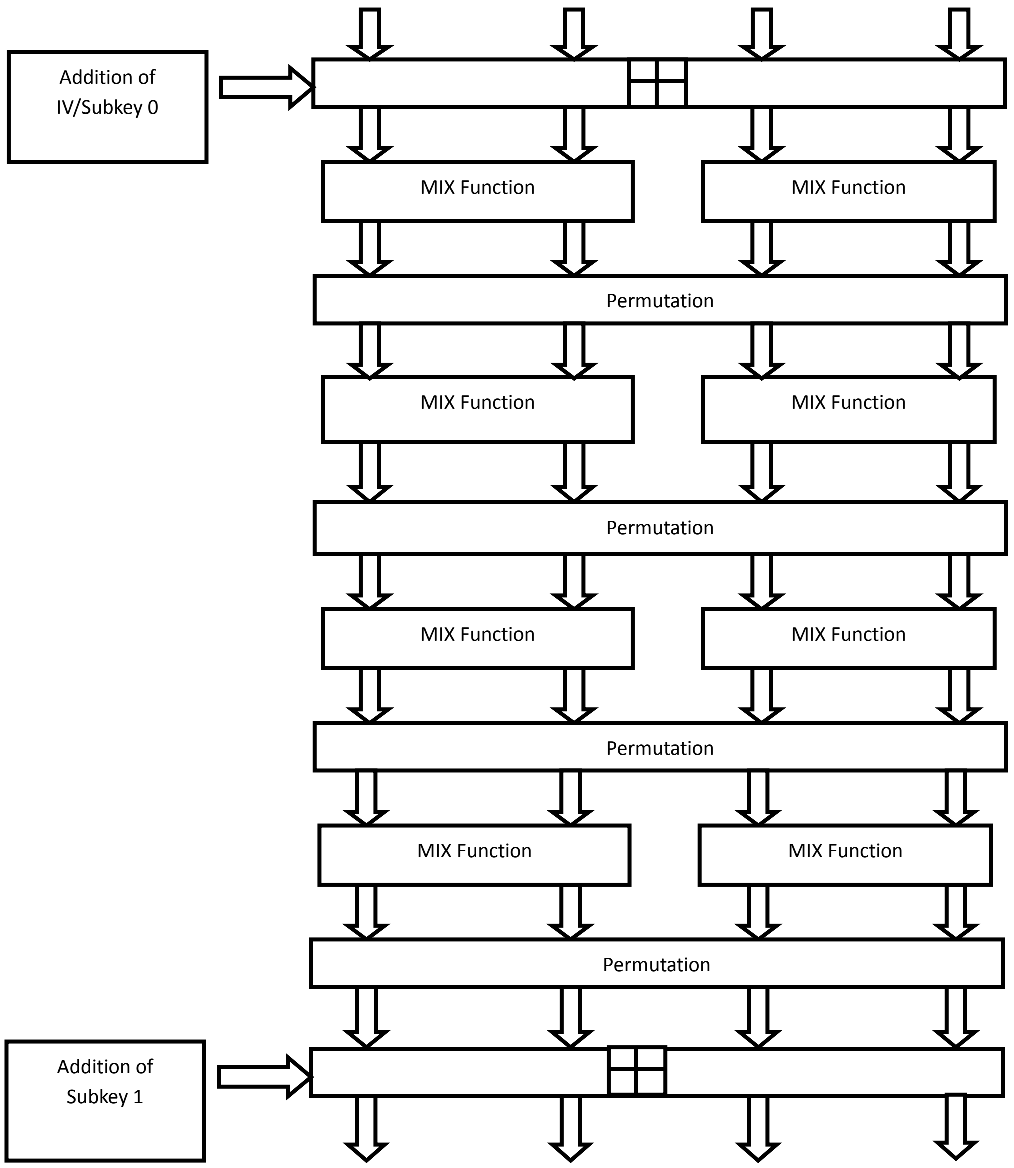


Figure .: First four rounds of Threefish-256 block cipher

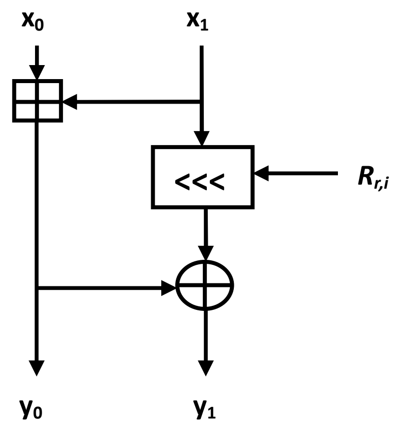


Figure .: The MIX function

Table .: Rotation constants for each N_w

Nw		**4**		**8**				**16**							
j		**0**	**1**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
d =	0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
	1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
	2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
	3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
	4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
	5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
	6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
	7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

The output of MIX function is fed to the word permutation function. As given in Table 2.3 below, permutation is different based on the value of N_w. For example, for N_w=4, the output of MIX function y_1 is swapped with y_3.The output of permutation is the output of a single round. After every N_w rounds, a subkey is added to the output of the round. For N_w=4, a total of 18 subkeys are required for N_r =72. The subkeys are calculated using the Key-scheduler.

Table .: Values for word permutation

		i =															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	4	0	3	2	1												
$N_w =$	8	2	1	4	7	6	5	0	3								
	16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

2.1.2 Key schedule:

The key-schedule requires an initial block cipher key also known as initialization variable IV and a tweak to calculate the subsequent subkeys. The tweak is a 128 bit value that is based on several fields as shown in Figure 2.4. The only fields that change commonly are the First, Final, Position, Type. TreeLevel field is used when using Tree hashing mode of Skein. The key-schedule is defined as:

$$k_{s,i} = IV_{(s,i) \bmod (Nw+1)} + t_{(s+1) \bmod 3} \quad (4)$$

$$k_{s,i} = IV_{(s,i) \bmod (Nw+1)} + t_{s \bmod 3} \quad (5)$$

$$k_{s,i} = IV_{(s,i) \bmod (Nw+1)} + t_{(s+1) \bmod 3} \quad (6)$$

$$k_{s,i} = IV_{(s,i) \bmod (Nw+1)} + s \quad (7)$$

Where i = 0 to N_w and s = 1 to 18. Each equation results in a 64 bit value which when combined together forms a single subkey. For each subsequent key, the above key-schedule is used.

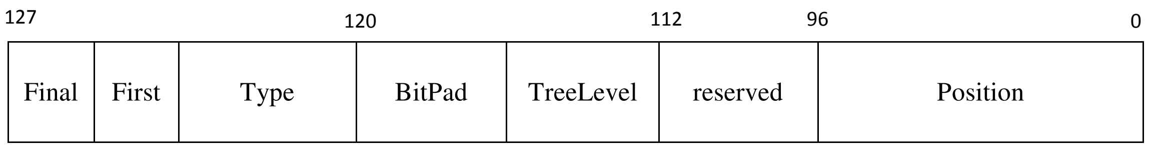


Figure .: Tweak fields

Table .: Tweak field values

Name	**Bits**	**Description**
Position	0-95	The number of bytes in the string processed so far (including this block)
Reserved	96-111	Reserved for future use, must be zero
TreeLevel	112-118	Level in the hash tree, zero for non-tree computations
BitPad	119	Set if this block contains the last byte of an input whose length was not an integral number of bytes. 0 otherwise
Type	120-125	Type of the field (Config, Message, output, etc.)
First	126	Set for the first block of a UBI compression
Final	127	Set for the last block of a UBI compression

2.2 Unique Block Iteration:

For messages longer than N_b bits, Skein uses UBI chaining mode. The message M is divided in to n message blocks M_0, M_1, …, M_{n-1} and processes the message block wise. It requires three inputs: A message block of size Nb bits, an initial chaining variable IV and a starting tweak value T that is determined by the UBI as explained in the previous section. Figure 2.5 explains the processing of three block message. As the figure shows, the output of the last message block serves as the chaining variable for the next message block. The number of message blocks is determined by the size of the message M. Additional zeros can be padded to the message block to make its size equal to Nb bits for Threefish processing. The Type field in the Tweak value changes according to the type of input being handled. For example, in figure 2.5, the input is messages so the Type field in tweak used is T_{msg}. When the UBI processes the message, it is known as Message UBI and the Type field for all the message blocks remains the same in their respective tweak values throughout the Message UBI. Other Skein type values are given in table 2.5.

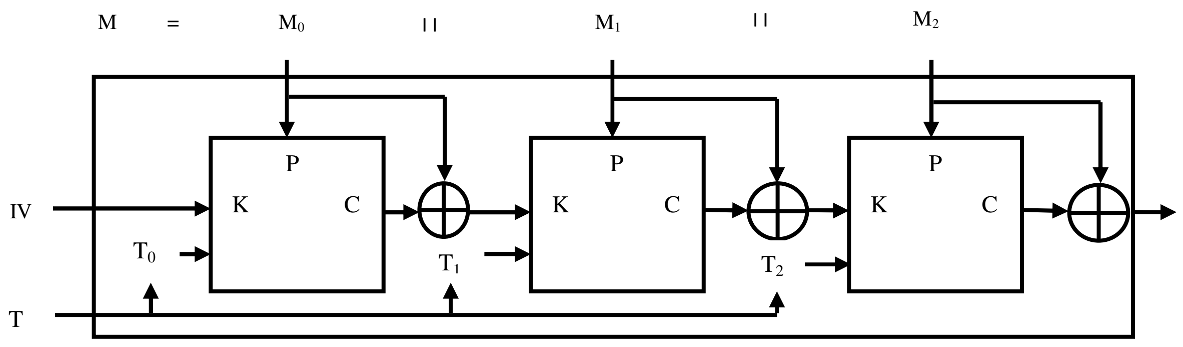


Figure .: UBI Chaining Mode of message M

Table .: Type field values

Symbol	**Value**	**Description**
Tkey	0	Key (for MAC and KDF)
Tcfg	4	Configuration Block
Tprs	8	Personalization string
TPK	12	Public Key (for digital signature hashing)
Tkdf	16	Key identifier
Tnon	20	Nonce (for stream cipher or randomized hashing)
Tmsg	48	Message
Tout	63	Output

Two additional stages are used before and after Message UBI known as Configuration UBI and Output UBI respectively as shown in figure 2.6. Configuration UBI takes a configuration string C of 256 bits as input. The string is a constant that has different fields as given in Table 2.7. Tree field values change when tree-hashing mode is used and remains 0 otherwise. The Ouptut UBI stage is used after the Message UBI that transforms the output of Message UBI in to desired number of bits. The output of this stage is the actual Message Digest.

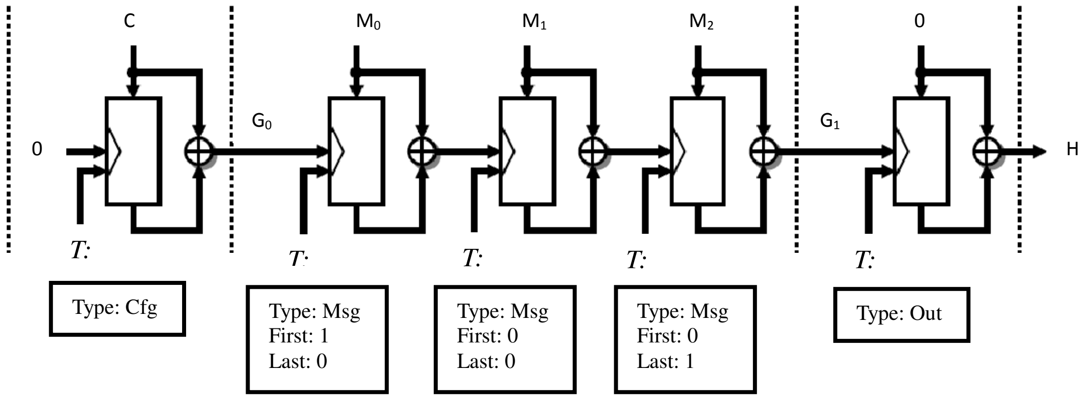

C
M0
M1
M2
0
0
G0
G1
H
T:
T:
T:
T:
T:
Type: Cfg
Type: Msg First: 1 Last: 0
Type: Msg First: 0 Last: 0
Type: Msg First: 0 Last: 1
Type: Out

Figure .: UBI stages [7]

Table .: Configuration string field values

Offset	**Size in Bytes**	**Name**	**Description**
0	4	Schema identifier	The ASCII string “SHA3” =(0x53,0x48,0x41,0x33), or ToBytes (0x33414853,4)
4	2	Version number	Currently set to 1:ToBytes(1,2)
6	2		Reserved, set to 0
8	8	Output Length	ToBytes (No, 8)
16	1	Tree leaf size en-coding.	Y_L
17	1	Tree fan-out encod-ing.	Y_F
18	1	Max. tree height	Y_{MAX}
19	13		Reserved, set to 0

2.3 Tree- hashing:

As explained earlier, Skein is sequential in nature and thus, by default, processes a message sequentially. However, for very long messages, calculating the message digest may take long time. Skein can be used in Tree hash mode to process message blocks simultaneously by utilizing multiple cores in a machine. For using Skein in Tree-hash mode, the process remains the same as sequential process with the only difference of Treelevel field in the Tweak and Tree parameters in the configuration string. The UBI chaining mode is replaced by a Tree structure as depicted in figure 2.7. The tree hash mode of Skein is based on Merkle Tree structure [2]. The tree is processed from top to bottom. Each node in the tree takes three inputs, IV, M and T. The input message is divided into N_b bit blocks and is distributed to the leaf nodes. The values of tree parameters Leaf size encoding, Fan-out encoding and Maximum tree height construct the tree structure and it varies according to these values. The definitions of tree parameters are given in table 2.7. Tree structures for different tree parameters are given in figure 2.8.

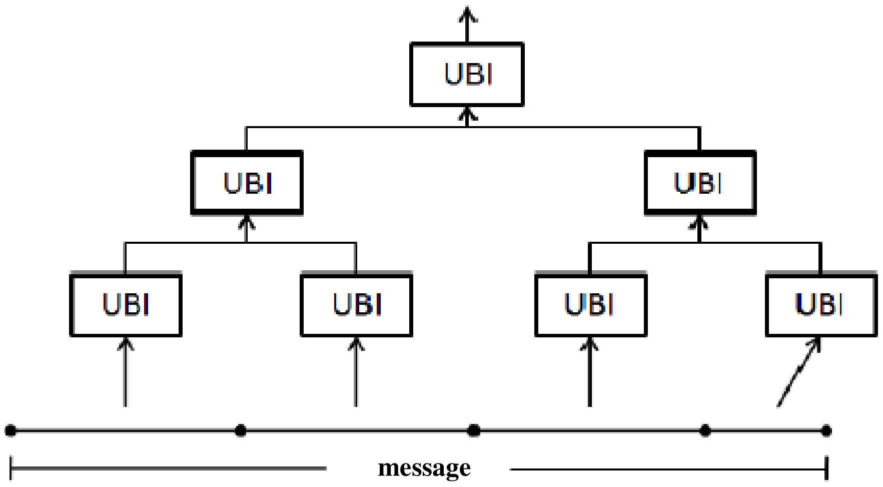


Figure .: A Merkle Tree Structure [8]

Table .: Hash-Tree mode parameters

Symbol	**Description**
Y_L	The leaf size encoding. Number of blocks processed by each leaf node 2^{Y_L} ($Y_L \geq 1$)
Y_F	The fan-out encoding. The fan-out of a tree is 2^{Y_F} with $Y_F \geq 1$
Y_{MAX}	The maximum tree height; $Y_{MAX} \geq 2$
N_l	Leaf size, $N_b 2\ Y_L$
N_n	Node size, $N_b 2\ Y_F$

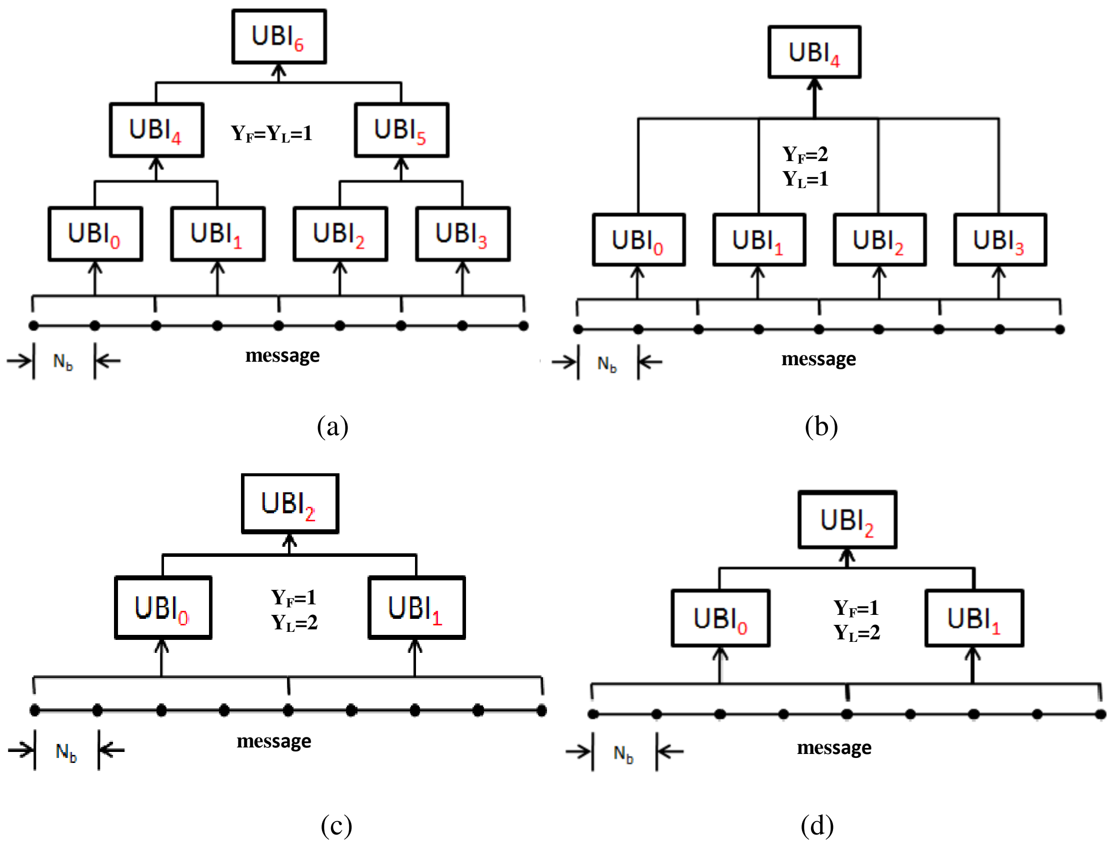


Figure .: Skein Tree structure for different Tree parameters [8]

3 LITERATURE REVIEW

NIST holds performance of a hash function as the second most important criterion for its evaluation after security [9]. This includes computational effectiveness, which not only denotes the data rate of the algorithm but also memory requisites such as the code size and the random-access memory (RAM) requirements for the purpose of software implementations, and the gate-counts for hardware implementations. In this regard performance evaluation of all the candidates is being continuously carried out on software, hardware and embedded platforms before the announcement of the final result in 2012. Continuous research on optimization techniques and cryptanalysis are under way to help NIST select the right SHA-3 candidate.

3.1 Hardware Implementations:

Various hardware implementations targeting on high-speed performance and compactness of required area have been proposed for Skein. High-speed hardware implementation has been proposed by Stefan Tillich in [10] for all the three variants of Skein on 0.18 µm standard-cell implementation and two modern FPGA architectures: Xilinx Spartan 3 and Xilinx Virtex 5. 0.18 µm standard-cell library was used for ASIC synthesis which re-sulted in throughput of 0.882 Gb/s for small Skein-256 and 1.762 Gb/s for fast Skein-256. On Xilinx Spartan 3, it came out to be 0.669 Gb/s and on Xilinx Virtex 5, 1.751 Gb/s.

Skein-256-256 is also implemented using Verilog on standard-cell library of 0.35 µm [11]. It has a datapath of 64 bits including a temporary 64-bit register, a register file with a size accommodating 16 words, a control FSM, and an AMBA APB interface of 32 bits. The 64-bit adder has been implemented in a standard fashion with Verilog's built-in '+' operator so that synthesizer can allow greatest flexibility for optimization. The results revealed an area of 12,890 GEs and throughput of 19.8 Mbits/sec.

The work in [12] focuses on low budget a cryptographic solution that enables investigations for possible optimizations for area efficient implementations, and to neglect pure high-throughput considerations. It showed that Skein is the lowest area efficient candidate as well as least performance efficient as compared to JH and Grostl with areas 1621, 1018 and 1722 slices respectively and throughputs 3178, 5416 and 10276 Mbits/sec respectively.

A low-area coprocessor has been designed on a Field-Programmable Gate Array (FPGA) in [13] for light-weight implementation of Skein-512-512. The architecture is prototyped on a Xilinx Virtex-6 device and produced result with an area of 132 slices and throughput of 80 Mbits/s. This work concluded that Skein has an upper edge over other SHA-3 finalists in that the same coprocessor allows one to encrypt or hash a message.

3.2 Software Implementations:

Several Software implementations of Skein have been reported using C#, .Net and Java on Skein's official website. In C, Skein has been efficiently implemented on Intel Core 2 Duo 2GHz processor [14]. This implementation revealed a throughput of 4.9Mbytes per second. Skein and MD6 has been implemented on NVIDIA graphics card and on a single core machine using CUDA-C to exploit the parallelism feature of the hash function using Tree Hashing in [1]. The SHA-3 candidates of round-2 including Skein have been implemented on Cell Broadband Engine (Cell) and NVIDIA graphics Processing Unit (GPU) for performance estimates [15]. A throughput of 1.9 Gb/sec is reported for Skein-512 on SPE architecture and 22.1 Gb/sec for the same on GPU architecture. Skein has been implemented using Spark language that is the subset of Ada. The implementation is known as SPARKSkein. The results have been compared with that of the C on the same Core i7 860 processor 2.8 GHz machine running 64bit GNU/Linux [16].

An efficient algorithm for implementing parallelism has been proposed in [17]. This paper describes sequential and parallel algorithms for Skein cryptographic hash functions, and the analysis, testing and optimization. Three possibilities to address parallelism have been discussed that includes Lower-level node priority, Higher-level node priority and Priority to a fixed number of nodes of higher level and same level. However, the first of these possibilities has been implemented using JAVA. It is implemented using the tree-hash approach by utilizing multithreads. The testing platform used was Dell Latitude D830, Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20Ghz, 2GB RAM, L2 cache size 4MB with a Ubuntu 9.10 operating system. On the 64-bit processor, Skein-256 turned out to be slowest of all the Skein versions Skein-1024 and Skein-512. Also, the tests showed that the one thread per node strategy is not efficient though there is maximum utilization of the CPU. After optimization, for a 700MB file, results showed speeds of 27 seconds for JAVA sequential version and 20 seconds for Skein java parallel version. These results when compared to the C reference implementation of Skein showed that JAVA tree hash mode implementation was slower. On the other hand, their parallel implementation using thread pool is faster than the C implementation. Such implementations are useful for high-end applications which require efficiently tuned implementations on multi-core target processors.

To the best of our knowledge, there is no previous work implementing Skein on core-i5 and CN5860 OCTEON Plus.

4 TARGET PLATFORMS

4.1 The Intel core-i5 processor:

The Intel i5-2450M is a Dual-Core Laptop processor that uses a 64 bit instruction set. It has 4 threads and 2 cores. It has a maximum clock speed of 2.50 GHz. The processor has built-in over-clocking which increases the maximum clock speed to 3.10 GHz. The GPU model is HD Graphics 3000 with clock speed ranging from 650 MHz – 1300 MHz. The RAM installed on the system is 4 GB but is capable of accommodating 16 GB of RAM. Memory supported is DDR3-1066 and DDR3-1333. The cache size is 3MB. The processor provides 2 memory channels, and has a maximum memory bandwidth of 21.3 GB/s.

4.2 OCTEON Plus processor:

OCTEON Plus CN5860 processor belongs to the family of OCTEON Plus CN58XX. It has 16 cnMIPS cores, frequency of 800MHz and supports a maximum of 25.6B of Instructions per second. The cnMIPS cores use the MIPS64 v2 instruction set, supporting both 32-bit and 64-bit processing. On the OCTEON processor, cache only supports access to system memory (DRAM) and not I/Os. The purpose of cache is to help in enhancing the system performance by means of core local or chip local fast memory which can save a copy of data accessed recently. The cache hierarchy includes:

- L2 cache ranging from 256 KB to 2 MB shared by the cores and I/O subsystem
- L1 instruction cache, 32 KB per core
- L1 data cache 8 KB to 16 KB per core

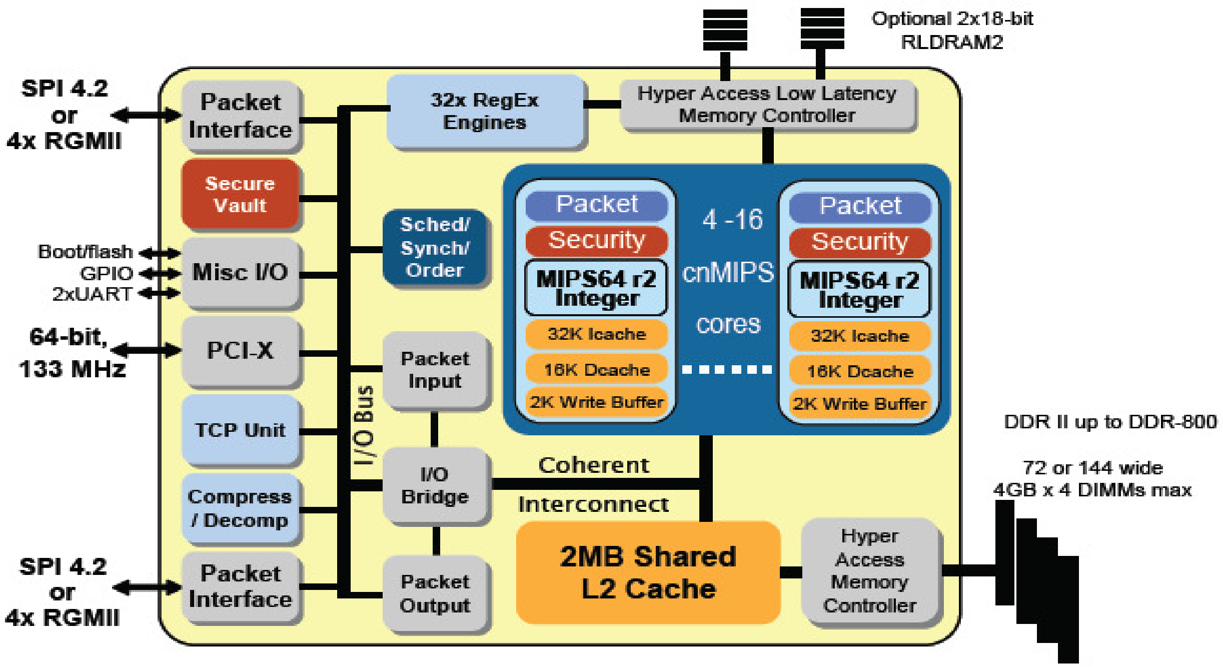


Figure .: OCTEON Plus Evaluation board Block diagram [4]

OCTEON processor is integrated with multiple hardware units that are used to reduce the load of cores thereby reducing overhead and complexity. In addition, these hardware units possess Dedicated DMA Engines to access memory. Cores and hardware units interconnect using high-speed connections. These connections operate at the same frequency as the cores. These connections are a collection of buses. The advantages of the hardware structure is that it enables an adaptable software structure design and at the same time allows for core grouping as required in order to increase performance. In addition, flexible software architecture, standard MIPS64 ISA and industry standard Toolchains helps in minimizing software development complexity. Modifications have been made in industry standard Toolchains (GCC, GDB) and operating systems (including SMP Linux) so that OCTEON's processor's multiple cores, special Cavium Networks specific instructions and hardware acceleration units can be utilized. C/C++ codes can be written easily while reusing the legacy software. Programs written for MIPS64 and MIPS32 ISA are inherently supported.

A Cavium Networks Software Development Kit (SDK) is provided. The SDK comprises of GNU C/C++ compiler and other development tools, C language Application Programmer's Interfaces (APIs) to the hardware units, a simple executive that can execute code on the cores without any operating system and Cavium Networks SMP Linux. The Simple Executive provides a Hardware Abstraction Layer (HAL) in the form of an Application Programming Interface (API) to the underlying hardware units. The CPU registers are accessed via this thin layer of API. It offers convenience for the process of block initialization. In addition, the hardware units can be accessed by Simple Executive API.

Simple Executive functions and macros allow building standalone applications or they can be run via drivers or application running on an operating system. For instance, as soon as the booting of Linux takes place, a Cavium Networks Ethernet driver may be initiated. In order to configure the OCTEON hardware, this driver makes use of the Simple Executive API. Using Linux, Simple Executive User Mode applications can also be executed supporting both 32 bit and 64 bit modes. A number of choices are available when it comes to runtime environment. Cavium Networks supply three such environments: Simple Executive standalone mode, hardware simulator and Linux as demonstrated in figure 4.2.

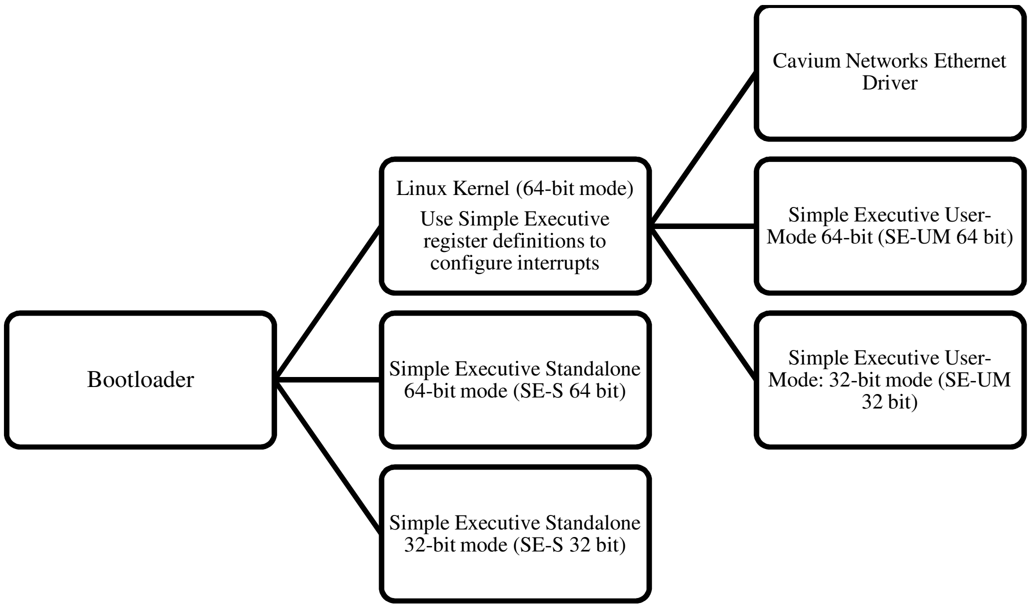

Cavium Networks Ethernet Driver
Linux Kernel (64-bit mode) Use Simple Executive register definitions to configure interrupts
Simple Executive User-Mode 64-bit (SE-UM 64 bit)
Bootloader
Simple Executive Standalone 64-bit mode (SE-S 64 bit)
Simple Executive User-Mode: 32-bit mode (SE-UM 32 bit)
Simple Executive Standalone 32-bit mode (SE-S 32 bit)

Figure .: Different runtime environment choices

4.2.1 Simple Executive

Simple Executive is used to provide API to hardware units. It can either be run as a user mode (SE-UM) application on an operating system for example on Linux or it can be run as standalone (SE-S). Different application startup code (main()) when Simple Executive is run as a user mode application.

4.2.2 SMP Linux

In SMP (Symmetric Multi Processing), Linux may be run on or more cores. There are two choices for file system; it can either be the tiny root embedded file system (embedded_rootfs) or the large Debian file system. The root file system acts as a RAM disk when Linux is run with the embedded root file system (embedded_rootfs). In such a case, the ELF file is either downloaded from a host, or stored in on- board flash. When no device is attached to the OCTEON processor for storing root file system to download OCTEON, the embedded root file system is used. Usually, the embedded root file system contains least number of files required. A small utility called "BusyBox" instead of the normal Linux utilities in order to save space. The utility can be customized to add functions according to the requirement of the application thus reducing the size of the executable file and saving space.

The Simple Executive is different from Linux in a sense that when it is run in standalone mode, it offers minimum overhead along with the greatest potential for scaling. Simple Executive when run as user mode application may perform significantly slower than simple Executive Standalone.

5 DESIGN AND IMPLEMENTATION

5.1 Design Methodology

The main goal of this work was to design a static source code for Skein-256 for parallel implementation that would hash portions of a large message simultaneously. Utilization of multiple cores was the key objective when designing the architecture.

5.1.1 Sequential processing of Skein

In order to process an input message sequentially, it is divided into equal blocks of 256 bits $M_o \dots M_{n-1}$ where n is the number of blocks. UBI function is then called for each message block. For every message block, Threefish function is called that executes the number of rounds required for Skein-256 that is, $N_r = 72$.

The 256 bit message block is further divided into four blocks of 64 bits and is stored into unsigned 64 bit arrays. This is because the maximum size that could be achieved in an array location is 64 bit and because of the MIX operations that needs to operate on 64 bit blocks of data. In each round from $N_r = 0$ to 71, a key scheduler function is called that takes input the initial chaining variable IV and tweak T and returns eighteen subkeys for skein-256. The first subkey is the same as the initial chaining variable IV that is predefined. These subkeys are added to the intermediate outputs after every four rounds.

A mix function and permutation function constitutes each round. Mix function performs the operations of addition, left-shift rotate and XOR. Number of bits left shift-rotated depends on the rotation constants that are different for each round so a check has been implemented here. The output of mix function forwarded for permutation. The hash output is xored with original input message and the result is fed as the initial variable IV for the next message block M_1. Subsequent tweaks for the rest of the message blocks are calculated using the tweak calculator function.

The drawback with sequential processing is that when one message block is being processed all other successive blocks have to wait for their turn. This puts a limitation on the usage of multiple cores because of data dependency nature of Threefish where the processing of subsequent message blocks depends on the output hash value of the previous message blocks.

5.1.2 Parallel processing of Skein

To address parallelism, a Merkle tree structure is exploited. Hash tree structure can be traversed in different ways [17]. In this work, tree is traversed from top to bottom and left to right. This method is referred to as leftmost node priority in [17].

The algorithm design is based on a static hash tree model for two fixed input message. Figure 5.2 shows a hash tree model for input message size of 1024 bytes. As the figure depicts, the maximum tree height and the number of nodes at each level are constant for basic implementation. The first level starts with 16 nodes and at every successive level the number of nodes gets halved of the previous level. The maximum tree height is 5.

The original input message is again divided into 256 bit blocks M_0 – M_{n-1}. A group of these blocks are fed to each node simultaneously. A single thread per node is initiated at every level. That means, at the first level, 16 threads are initiated essentially. Each thread invokes the sequential process for the message blocks passed as input arguments. The output of two consecutive nodes is concatenated to form a new message which is again fed to the higher level node. At each level, the priority starts from the leftmost node. Same sequential process approach as mentioned above is extended towards parallelism with the addition of multithreading. Value of leaf size encoding depends on the size of the input message and thus increases with larger input message given a fixed number of cores.

A thread function is invoked that takes the input message as input argument. The task of this function is to divide the input message into 256 bit blocks and pass them to each thread created in the first level of the tree. Since maximum number of cores in OCTEON processor is 16, 16 threads are created where each thread must take at least two blocks as input argument. Each thread executes sequential processing for each 256 bit block. At every level, the number of threads are halved the previous number of threads (for our case) until one is reached. This is the root node or the last level of the tree which requires no thread creation. For the root node, the sequential process is called in a normal way.

After the sequential process finishes, threads join the parent thread. For the successive levels of tree hashing except for the last level (root node), each node waits for all its child nodes (2 in our case) to finish processing before beginning its own process. This is a constraint to achieve maximum parallelism because even if the child nodes of other parent nodes have finished processing, parent nodes still have to wait for the child nodes of predecessor parent nodes when traversing from left to right.

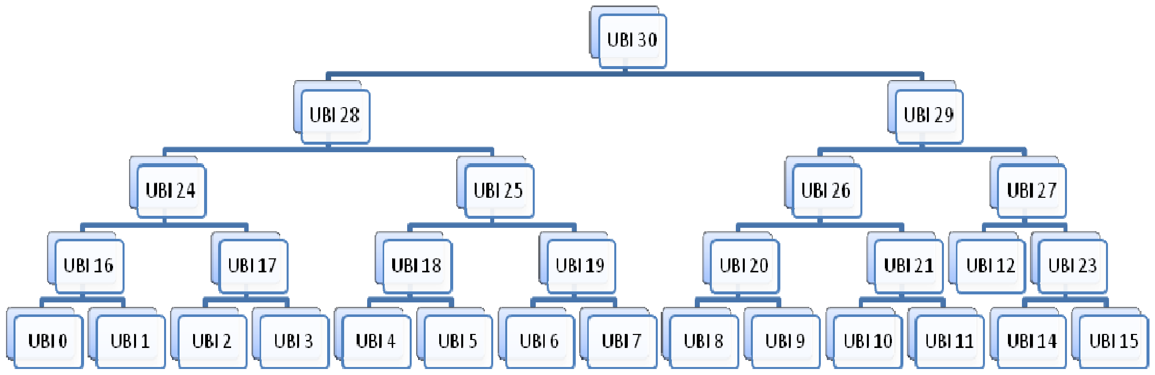


Figure .: Hash-Tree model for 16 core parallel processing

5.2 Implementation:

The implementation on the two unexplored platforms is not for optimization but only for demonstrative purposes and testing the performance of Skein.

5.2.1 Intel Core - i5:

The set up of Intel Core i5 serves two purposes: For the implementation and analysis of Skein-256 and second, to serve as a cross- development host system for OCTEON embedded platform which is called as the development target. Linux operating system Fedora 17 is used with kernel 2.6. For compiling, GNU-GCC compiler is used. Twenty samples for both sequential implementation and parallel implementation have been taken.

5.2.2 OCTEON evaluation board CN5860:

For the implementation on OCTEON evaluation board CN5860, an i386 or x86_64 machine is required that will serve the purpose for the cross-development platform. This machine is referred to as development host whereas OCTEON evaluation board is referred to as the Target host. The development host used for this research is the same Intel core i5 machine with OS 64 bit Fedora 17. As explained in Chapter 4, Cavium Network provides different runtime environment choices. Here, SMP Linux Runtime environment is used. Therefore, the following steps for the implementation are according to the SMP Linux configuration and the sequential and parallel programs designed above are cross-compiled using the Linux development tools. The implementation requires:

1. Installing OCTEON-SDK
2. Build OCTEON Linux
3. Copying it to compact flash and copying the program to CF
4. Setting up the EVB
5. Booting the board and downloading the program

1. Installation of SDK:

The sequential and parallel Skein codes are built on the development host and then down-loaded to the development target connected to the development host through serial cable. For this, Software Development Kit (SDK) that comes with the OCTEON evaluation board is installed on the development host. The SDK has two rpm packages: the base SDK (OCTEON-SDK-*.i386.rpm) (OCTEON-SDK-2.3.0-427.i386) and OCTEON Linux (OCTEON-LINUX-*.i386.rpm). The base SDK package includes:

- GNU based tool chain with linker, compiler and generic libraries.
- OCTEON simulation software with performance measurement utilities
- A Cavium Networks Simle Executive software for easy application development.
- Example applications

The OCTEON Linux package contains OCTEON Linux that is ported to the OCTEON processor. OCTEON_SDK is installed by running, as root, the command:

rpm -i /media/OCTEON-SDK/.rpm*

The SDK is by default installed in the directory: /usr/local/Cavium_Networks/OCTEON-SDK which is created during installation. This directory is referred to as $OCTEON-SDK. OCTEON-SDK is an environment variable and $OCTEON-SDK refers to the value of the environment variable. The working directory is changed to $OCTEON-SDK and a script 'env-setup' is executed that sets essential environment variables. The following variables are set by the env-setup script:

- OCTEON-SDK
- The PATH
- OCTEON_MODEL
- OCTEON_CPPFLAGS_GLOBAL_ADD

The env-setup script requires the value of a single argument: OCTEON_MODEL. The value of this argument is chosen from a list of values available in a file $OCTEON-SDK/octeon-models.txt. The OCTEON Evaluation board model used in this work is CN5860 so the value of OCTEON_MODEL from the text file is specified as OCTEON_CN58XX. The env-setup script is sourced (source env-setup) to modify the environment variables of the current shell (usually bash). As a result, all shells started from this shell inherit the shell's environment variables.

2. Build OCTEON Linux kernel:

OCTEON Linux kernel is built for the Target development on the host development. An ELF file (vmlinux.64) is created and stored in on-board flash. Since embedded systems do not have built-in memory like hard disks, linux is booted from this ELF file. Everytime the development target is powered off, the copy of the file system in the memory is deleted. Thus, vmlinux.64 is booted from the onboard flash whenever the target development is powered on again. Following command is used to build OCTEON Linux kernel:

```
$ cd $(OCTEON-SDK)/linux
 $ make -s clean
 $ make -s kernel
```

Successful built shows the following output:

```
Preparing...
##################################################
OCTEON-LINUX
##################################################
The Linux Kernel has been successfully installed under the direc-
tory
/usr/local/Cavium_Networks/OCTEON-SDK/linux
Please    refer    to    file:///usr/local/Cavium_Networks/OCTEON-
SDK/docs/html/linux.html
on how to use Linux on the OCTEON.
```

This build takes about 20 minutes. The Makefile creates an ELF file at *$OCTEON-SDK/linux/kernel_2.6/linux/vmlinux.64* which can be run on the OCTEON processor. This ELF file contains the Linux kernel and a filesystem which runs in memory only. This filesystem is the embedded root filesystem (*embedded_rootfs*).

3. Copying vmlinux.64, file system and the source code to Compact Flash:

A 1GB compact flash that comes with the evaluation board is organized as two PC style partitions. The first partition is used for the kernel image **vmlinux.64**. This partition is used by the bootloader, so it uses the FAT16 file system (vfat under Linux). The second partition is an EXT3 file system containing the embedded root filesystem.

Partition	Size	File system	Target Mount	Host Mount	Description
0	67MB	vfat	/dev/sda1	/mnt/cf1	Kernel and files for the Bootloader vmlinux.64
1	946MB	ext3	/ dev/sda2	/mnt/cf2	Embedded root filesystem

Table .: Layout of 1 GB compact flash

Mount points are created for the compact flash in /mnt:

```
$ mkdir -p /mnt/cf1
$ mkdir -p /mnt/cf2
```

Both partitions are mounted on /run/media/aisha. The Linux kernel is copied to the compact flash:

```
$ mount /dev/sdb1 /mnt/cf1
$mips64-octeon-linux-gnu-strip  -o  /mnt/cf1/vmlinux.64 kernel_2.6/linux/vmlinux.64
$ umount /mnt/cf1
```

Three tool directories are created that are commonly used in building, running, and debugging applications:

```
$OCTEON-SDK/tools/bin
$ OCTEON-SDK /host/bin
$ OCTEON-SDK /linux/kernel_2.6
```

Two sets of GNU cross development tools are present in the tools/bin directory. These tools are run on the development host to build binary files for the development target. These tools include files and utilities that build software for the COTEON processor. The two sets of tools are:

1) Simple Executive Development Tools: The mipsisa64-octeon-elf-* tools are used to build Simple Executive applications.
2) Linux Development Tools: The mips64-octeon-linux-gnu-* tools are used to build the Linux kernel and Linux User-mode applications.

The latter is used to cross-compile the sequential and parallel SKEIN using the GCC utility:

```
/usr/local/Cavium_Networks/OCTEON-SDK/tools/bin/mips64-gnu-gcc -o outputfilename inputfilename.gcc
```

A binary with the name of outputfilename is created in the working directory. For the parallel skein, the thread library is also linked during the cross-compilation as follows:

```
/usr/local/Cavium_Networks/OCTEON-SDK/tools/bin/mips64-gnu-gcc -o outputfilename inputfilename.gcc  -lpthread
```

These binaries are copied to the CF using the commands:

```
$mount   /dev/sdb2   /mnt/cf2

$cp outputfilename  /mnt/cf2

$unmount /mnt/cf2
```

4. Connection to the EBT5860 Hardware:

The OCTEON 5800 target development evaluation board consists of:

1. A NULL modem serial cable to attach to the Linux development system, a USB Flash reader/writer, and a Compact Flash card.
2. The EVB-5800-NIC4, power supply and cables

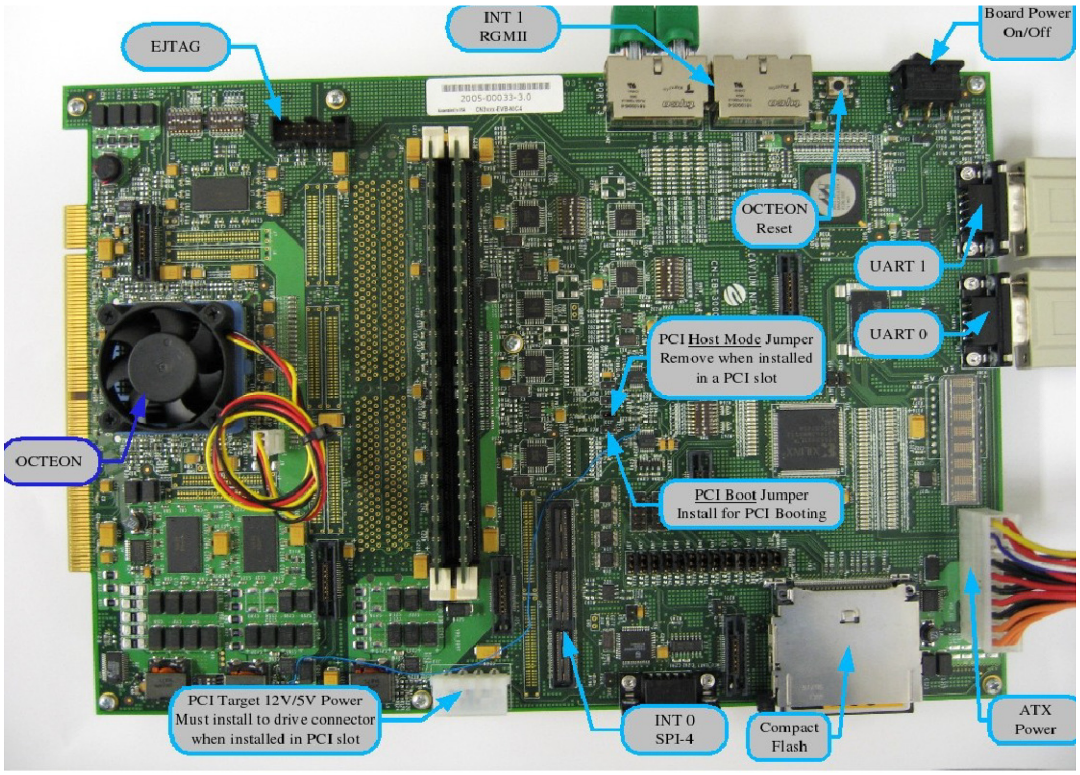

EJTAG
INT 1 RGMII
Board Power On/Off
OCTEON Reset
UART 1
UART 0
PCI Host Mode Jumper Remove when installed in a PCI slot
OCTEON
PCI Boot Jumper Install for PCI Booting
PCI Target 12V/5V Power Must install to drive connector when installed in PCI slot
INT 0 SPI-4
Compact Flash
ATX Power

Figure .: The CN5860 OCTEON Evaluation board [4]

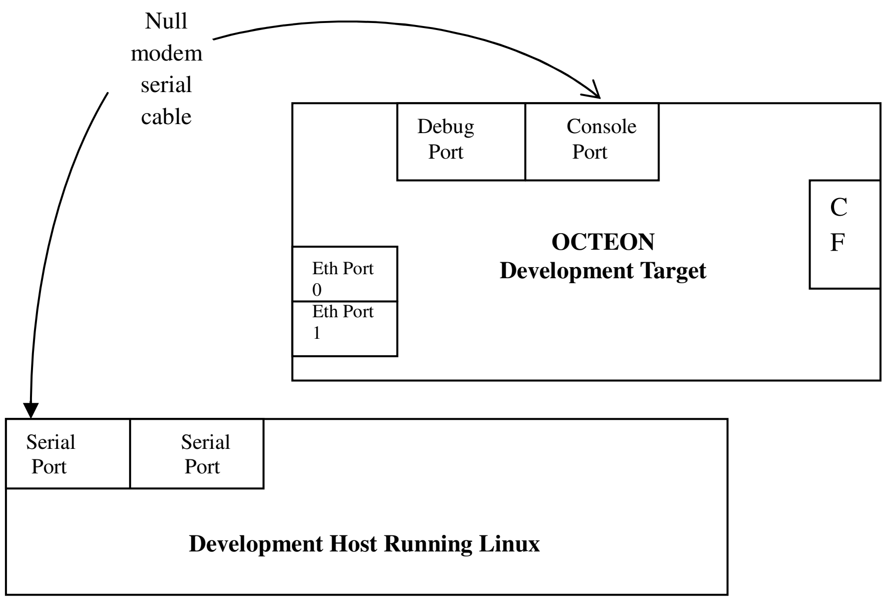


Figure .: Hardware configuration –Connections to Development target

The console output for the target development is directed to UART0 on the target evaluation board and is viewed on the terminal emulator, HyperTerminal. Linux connects to the first serial port on the device /dev/ttyS0.

5. Booting from Onboard Flash

When the evaluation board is booted from the vmlinux.64 in the on-board compact flash, Core 0 starts execution at the reset vector 0xBFC00000 (the location of the bootloader code in onboard flash). The bootloader (U-Boot) then carries out the following steps:

1. Initializes the UART
2. Configures the DRAM controller to allow physical memory to be used
3. Relocates itself from the onboard flash to DRAM, and continues executing from DRAM.
4. Executes the default command, if present

```
.U-Boot 1.1.1 (U-boot build #: 217) (SDK version: 1.8.1-290) (Build time: Dec
9
2008 - 19:22:32)

EBT5800 board revision major:2, minor:0, serial #: 2009-2.0-
00512
OCTEON CN5860-NSP pass 2.3, Core clock: 600 MHz, DDR clock: 399 MHz (798 Mhz data
rate)
DRAM:  2048 MB
Flash:  8 MB
Clearing DRAM........ done
BIST check passed.
Net:   octeth0, octeth1, octeth2, octeth3
 Bus 0 (CF Card): OK

  ide 0: Model: CF 1GB Firm: 20071116 Ser#:
TSS20031090724031238
         Type: Removable Hard Disk
         Capacity: 967.6 MB = 0.9 GB (1981728 x 512)
```

The ELF file is downloaded to the specified Reserved Download Block address using the following command. Bootloaders built with SDK 1.7 and higher allow the specified address to be 0. When the address is 0, the default Reserved Download Block address is selected by the bootloader.

Octeon ebt5800# fatload ide 0 $(loadaddr) vmlinux.64

After the ELF file is downloaded, the bootloader relocates it to a physical location of its choice (creating the in-memory image).

```
Octeon ebt5800# bootoctlinux $(fileaddr)
ELF file is 64 bit
Attempting to allocate memory for ELF segment: addr: 0xffffffff80100000 (adjusted to:
0x0000000000100000), size 0x2045750
Allocated memory for ELF segment: addr: 0xffffffff80100000, size
0x2045750
Processing PHDR 0
  Loading 1fc4c00 bytes at ffffffff80100000
  Clearing 80b50 bytes at ffffffff820c4
## Loading Linux kernel with entry point: 0xffffffff80105c70
...
Bootloader: Done loading app on coremask: 0x1
Linux version 2.6.32.27-Cavium-Octeon (root@ash) (gcc version 4.3.3 (Cavium
Inc.
 Version: 2_3_0 build 116) ) #2 SMP Tue Jul 3 02:49:46 PKT
2012
CVMSEG size: 2 cache lines (256 bytes)
Cavium Inc. SDK-2.3
bootconsole [early0] enabled
CPU revision is: 000d030b (Cavium Octeon+)
Checking for the multiply/shift bug... no.
Checking for the daddiu bug... no.
Determined physical RAM map:
 memory: 0000000001171000 @ 0000000000f5f000 (usable after in-
it)
 memory: 000000000dc00000 @ 0000000002200000 (usable)
 memory: 0000000011400000 @ 0000000020000000 (usable)
Wasting 220360 bytes for tracking 3935 unused pages
Initrd not found or empty - disabling initrd
CVMX_GMXX_INF_MODE (block_id = 4) not supported on this
```

```
chip
Using internal Device Tree.
Placing 0MB software IO TLB between a800000002c8e000 -
a800000002cce000
software IO TLB at phys 0x2c8e000 - 0x2cce000
Zone PFN ranges:
  DMA32    0x00000f5f -
  Normal   0x000f0000 -> 0x000f0000
Movable zone start PFN for each node
early_node_map[3] active PFN ranges
    0: 0x00000f5f -> 0x000020d0
    0: 0x00002200 -> 0x0000fe00
    0: 0x00020000 -> 0x00031400
Cavium Hotplug: Available coremask 0x0
PERCPU: Embedded 10 pages/cpu @a800000002cdf000 s11264 r8192 d21504
u65536
pcpu-alloc: s11264 r8192 d21504 u65536 alloc=16*4096
pcpu-alloc: [0] 0
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
128736
Kernel command line:  bootoctlinux console=ttyS0,115
PID hash table entries: 2048 (order: 2, 16384 bytes)
Dentry cache hash table entries: 65536 (order: 7, 524288
bytes)
Inode-cache hash table entries: 32768 (order: 6, 262144
bytes)
Primary instruction cache 32kB, virtually tagged, 4 way, 64 sets, linesize 128 bytes.
Primary data cache 16kB, 64-way, 2 sets, linesize 128 bytes.
Secondary unified cache 2048kB, 8-way, 2048 sets, linesize 128
bytes.
Memory: 495492k/525764k available (5950k kernel code, 29796k reserved, 8763k data, 17860k
init, 0k highmem)
Hierarchical RCU implementation.
NR_IRQS:453
Calibrating delay loop (skipped) preset value.. 1200.00 BogoMIPS
(lpj=6000000)
Security Framework initialized
Mount-cache hash table entries: 256
Checking for the daddi bug... no.
Brought up 1 CPUs
NET: Registered protocol family 16
```

```
Not in host mode, PCI Controller not initialized
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
Switching to clocksource OCTEON_CVMCOUNT
NET: Registered protocol family 2
IP route cache hash table entries: 4096 (order: 3, 32768
bytes)
TCP established hash table entries: 16384 (order: 6, 262144
bytes)
TCP bind hash table entries: 16384 (order: 6, 262144 bytes)
TCP: Hash tables configured (established 16384 bind 16384)
TCP reno registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
/proc/octeon_perf: Octeon performance counter interface
loaded
octeon_wdt: Initial granularity 5 Sec.
octeon_gpio 1070000000800.gpio-controller: probed
HugeTLB registered 2 MB page size, pre-allocated 0 pages
JFFS2 version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
msgmni has been set to 968
alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
Serial: 8250/16550 driver, 6 ports, IRQ sharing disabled
brd: module loaded
loop: module loaded
pata_octeon_cf 1d000000.compact-flash: version 2.2 8 bit.
scsi0 : pata_octeon_cf
ata1: PATA max PIO6 cmd 900000001d000800 ctl
900000001d00080e
slram: not enough parameters.
mdio-octeon: probed
mdio-octeon 1180000001800.mdio: Version 1.0
```

```
Intel(R) PRO/1000 Network Driver - version 7.3.21-k5-NAPI
Copyright (c) 1999-2006 Intel Corporation.
e1000e: Intel(R) PRO/1000 Network Driver - 1.0.2-k2
e1000e: Copyright (c) 1999-2008 Intel Corporation.
sky2 driver version 1.25
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
usbcore: registered new interface driver libusual
i2c /dev entries driver
i2c-octeon 1180000001000.i2c: version 2.0
rtc-ds1307: probe of 0-0068 failed with error -5
md: linear personality registered for level -1
md: raid0 personality registered for level 0
md: raid1 personality registered for level 1
md: raid10 personality registered for level 10
md: multipath personality registered for level -4
md: faulty personality registered for level -5
device-mapper: ioctl: 4.15.0-ioctl (2009-04-01) initialised: dm-
devel@redhat.com

oprofile: using mips/octeon performance monitoring.
TCP cubic registered
NET: Registered protocol family 17
L2 lock: TLB refill 256 bytes
L2 lock: General exception 128 bytes
L2 lock: low-level interrupt 128 bytes
L2 lock: interrupt 640 bytes
L2 lock: memcpy 1152 bytes
1180000000800.serial: ttyS0 at MMIO 0x1180000000800 (irq = 125) is a OC-
TEON
console [ttyS0] enabled, bootconsole disabled
console [ttyS0] enabled, bootconsole disabled
1180000000c00.serial: ttyS1 at MMIO 0x1180000000c00 (irq = 126) is a OC-
TEON
Bootbus flash: Setting flash for 8MB flash at 0x1f400000
phys_mapped_flash: Found 1 x16 devices at 0x0 in 8-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
phys_mapped_flash: Swapping erase regions for broken CFI ta-
```

```
ble.
number of CFI chips:
cfi_cmdset_0002: Disabling erase-suspend-program due to code broken-
ness.
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
ata1.00: CFA: CF 1GB, 20071116, max MWDMA4
ata1.00: 1981728 sectors, multi 0: LBA
ata1.00: configured for PIO6
ata1.00: configured for PIO6
ata1: EH complete
scsi 0:0:0:0: Direct-Access     ATA      CF 1GB           2007 PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 1981728 512-byte logical blocks: (1.01 GB/967 MiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DP
O or FUA
 sda: sda1 sda2
sd 0:0:0:0: [sda] Attached SCSI removable disk
Freeing unused kernel memory: 17860k freed
/sbin/rc starting
Mounting file systems
Setting up loopback
Starting syslogd
Starting telnetd
/sbin/rc complete
Jan  1 00:00:02 (none) syslog.info syslogd started: BusyBox v1.18.4
Jan  1 00:00:02 (none) daemon.info init: starting pid 827, tty '': '-/bin/cav_sh
 /bin/sh'

BusyBox v1.18.4 (2012-07-03 02:31:49 PKT) built-in shell (ash)
Enter 'help' for a list of built-in commands.
~ # NET: Registered protocol family 10
```

6 PERFORMANCE RESULTS

This chapter provides details on performance analysis of C implementation of Skein-256 on Intel core-i5 and CN5860 OCTEON processor. The details of the performance provided below are for the purposes of illustration only. This work is not meant to optimize the method for practical use; the sole purpose is to demonstrate the performance of the hash algorithm on two unexplored platforms.

Table 6.1 shows performance results of Skein-256 sequential processing in terms of throughput for two fixed files sizes: 256 bytes and 1024 bytes. On core-i5, a 256 byte file takes 666.55 µsec and results in a throughput of 384.06 KB. When the file size increases to 1024 bytes, processing time also increases but results in a relatively high throughput of 588.84KB. As the file size is increased by a factor of 4, the processing time has increased only by a factor of 2.6 thereby resulting in a relatively high throughput. Same files when processed on OCTEON platform, shows more latency in processing as compared to the core-i5 processing times. However, as the file size is increased by a factor of 4, processing time shows increase by a factor of 4.5 that results in approximately same throughput as that for 256 byte file. The marked difference in the processing times of the two platforms for the same file sizes is due to their clock frequencies. OCTEON offers 800MHz whereas core-i5 has 2.5GHz.

Table .: Sequential Processing of Skein-256 on two platforms

Platform	File Size (bytes)	Execution time (µsec)	Throughput (Kbyte/s)
CORE-i5	256	666.55	384.067212
	1024	1739	588.844163
OCTEON	256	924.55	276.891461
	1024	4339.9	235.950137

Table 6.2 tabulate results of parallel processing of Skein-256 on the two platforms depending on the file sizes and the utilization of the number of cores. The methodology used here is similar to the approach in [7] referred to as the lower-level node priority. An advantage of this approach is that it provides maximum parallelism in theory and is reliant on the algorithm parameters which not only affect the tree structure in Skein Hash tree mode but also the node sizes. Hence, results here depicted are for fixed tree-structure with Y_F=Y_L= 1. When the same files are passed to core-i5 to run on all the cores available, it results in relatively high processing times to output the hash value and thus results in a low throughput as compared to the sequential processing of the same file. As the file size increases by a multiple of 4, processing time increases by a factor of 4.2. On OCTEON processor, it takes even more time to process 256 byte file as compared to that on core-i5. However, since there is the availability of maximum 16 cores, 1024 byte is run on all the cores. Not only the file size is increased but also the number of cores when increased by a factor of 4 resulted in reduced processing times thus giving an increased input compared to the results on core-i5. This difference is due to the limitation of number of cores in the latter that puts subsequent message blocks in pipeline whereas in the former, all message blocks are available with vacant cores and thus, no thread has to wait in pipeline for its turn.

Table .: Parallel processing of Skein-256

Platform	File size (bytes)	Number of cores	Execution time (µsec)	Throughput (Kbyte/s)
CORE-i5	256	4	1221.5	209.578387
	1024	4	5126.8	199.734727
OCTEON	256	4	1859.85	137.645509
	1024	16	3637.6	281.504289

In Table 6.3, the results have been compared to the work in [14] and [17]. In [14], only the C sequential implementation of Skein-512 is discussed which results in a throughput of 4.9 Mbytes/sec on 2 GHz Core 2 Duo processor. In [17], java implementations of both sequential and parallel processing of Skein-256, Skein-512 and Skein-1024 have been discussed. However, for comparison, results of only Skein-256 have been used in Table 6.3. There is a marked difference between results in [17] and results from this work due to a number of reasons. A major difference is of the file size used. Also, in [7], skein is implemented using java.

Table .: Performance comparison in terms of throughput

Platform	File size	Implementation	Throughput	
			Sequential processing	Lower node priority
[17]Skein-256	700 MB	Java	4 MB/s	26.5 KB/s
[14]Skein-512	512 bits	C	4.9 MB/s	---
Skein-256 on Core-i5	256 bytes	C	384 KB/s	209 KB/s
	1024 bytes	C	588 KB/s	199 KB/s
Skein-256 on OCTEON	256 bytes	C	276 KB/s	136 KB/s
	1024 bytes	C	235 KB/s	281 KB/s

7 CONCLUSION AND FUTURE WORK

Presently, C implementation of only sequential Skein is available in open literature. This work not only implements sequential skein-256 using C but also provides a basic parallel implementation of Skein-256 using C on two unexplored multiprocessor platforms (core-i5 and OCTEON). The design approach used is for fixed input parameters for illustrative purposes only. The results are reported in terms of throughput (KB/s) and compared with each other. From the results obtained, it is concluded that performance can be improved using large file sizes to utilize the true potential of multiprocessing.

Future work includes designing a generic algorithm for parallel processing of larger input files on core-i5 and OCTEON processor. Moreover, the algorithm should be optimized to provide variable tree structures based on the different values of Y_L and Y_F.

Bibliography

[1] G. v. Laszewski, A. Fitzgerald and A. Schorr, "Benchmarking of SHA-3 Candidates on a GPU using CUDA," Rochester Institute of Technology, 2009.

[2] X. Wang, D. Feng and H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPPEMD," in *Cryptology ePrint Archive*, 2004.

[3] "National Institute of Standards and Technology (NIST): Cryptographic Hash Algorithm Competition," [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[4] J. Curtis, "OCTEON Programmer's Guide," July 2010. [Online]. Available: http://university.caviumnetworks.com/downloads/Mini_version_of_Prog_Guide_EDU_July_2010.pdf.

[5] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas and J. Walker, "The Skein Hash Function Family Version 1.3," 2010.

[6] "Description of Known Answer Test (KAT) and Monte Carlo Test (MCT) for SHA-3 Candidate Algorithm Submissions," 20 February 2008. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/documents/SHA3-KATMCT1.pdf.

[7] N. At, J. Beuchat and I. San, "Compact Implementation of Threefish and Skein on FPGA," in *5th International Conference on New Technologies, Mobility and Security (NTMS)*, Istanbul, 2012.

[8] A. Schorr, "Performance Analysis of a Scalable Hardware FPGA," Rochester Institute of Technology, Rochester, New York, 2010.

[9] M. S. Turan, R. Perlner, L. E. Bassham, W. Burr, D. Chang, S.-J. Chang, M. J. Dworkin, J. M. Kelsey, S. Paul and R. Peralta, "Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition," National Institute of Standards and Technology, 2011.

[10] S. Tillich, "Hardware Implementation of the SHA-3 Candidate Skein," *IACR Cryptology ePrint Archive,* p. 159, 2009.

[11] S. Tillich, M. Feldhofer, W. Issovits, T. Kern, H. Kureck, M. Mühlberghuber, G. Neubauer, A. Reiter, A. Köfler and M. Mayrhofer, "Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein," *International Association for Cryptologic Research (IACR), Cryptology ePrint Archive,* 2009.

[12] B. Jungk, "Compact implementations of Grostl, JH and Skein for FPGAs," in *CRYPT II Hash Workshop*, 2011.

[13] A. Nuray, J. -L. Beuchat and S. Ismail, "Compact Implementation of Threefish and Skein on FPGAs," in *IACR Cryptology ePrint Archive*, 2012.

[14] K. Latif, M. Tariq, A. Aziz and A. Mahboob, "Efficient Software Implementation of Secure Hash Algorithm (Sha-3) Candidate-Skein," *International Journal of Academic Research,* pp. 313-317, 2011.

[15] J. W. Bos and D. Stefan, "Performance Analysis of the SHA-3 Candidates on Exotic Multi-Core Architectures," in *CHES'10 Proceedings of the 12th international conference on Cryptographic hardware and embedded systems*, Berlin, 2010.

[16] R. Chapman and E. Botcazou, "SPARKSkein: A Formal and Fast Reference Implementation of Skein," in *Formal Methods, Foundations and Applications; 14th Brazilian Symposium, SBMF 2011, São Paulo, Brazil, September 26-30, 2011, Revised Selected Papers*, Springer Berlin Heidelberg, 2011, pp. 16-27.

[17] K. Atighehchi, A. Enache, T. Muntean and G. Risterucci, "An Efficient Parallel Algorithm for Skein Hash Functions," *International Association for Cryptologic Research (IACR) Cryptology ePrint Archive,* 2010.