2015

Towards Designing Energy-Efficient Secure Hashes

Priyanka Dhoopa Harish
University of North Florida, n00931752@ospreys.unf.edu

TOWARDS DESIGNING ENERGY-EFFICIENT SECURE HASHES

by

Priyanka Dhoopa Harish

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computing and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December, 2015

The thesis "Towards Designing Energy-Efficient Secure Hashes" submitted by Priyanka Dhoopa Harish in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee: Date

_____ _____
Dr. Swapnoneel Roy
Thesis Advisor and Committee Chairperson

_____ _____
Dr. Karthikeyan Umapathy

_____ _____
Dr. Sandeep Reddivari

Accepted for the School of Computing:

_____ _____
Dr. Sherif Elfayoumy
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

_____ _____
Dr. Mark A. Tumeo
Dean of the College

Accepted for the University:

_____ _____
Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to express my special thanks to my thesis advisor, Dr. Swapnoneel Roy, for being such a good mentor. This thesis was a tremendous learning experience and exposed me to the research world. I would also like to thank my thesis committee members, Dr. Karthikeyan Umapathy and Dr. Sandeep Reddivari, for their valuable comments and guidance in conducting experiments that were helpful in completing this thesis. Dr. Asai Asaithambi has helped me at every stage of my presentation and provided valuable suggestions; thank you so much for helping me. I am extremely grateful to Dr. Sherif Elfayoumy for attending my presentation, providing insightful suggestions, and research ideas which broadened my analytical thought process; thank you, my learning experience was substantially enhanced by your guidance during the review process.

Furthermore, I want to express my sincere appreciation to Dr. Roger Eggen for his academic advice, Ms. Shawn Broderick for her administrative support and encouragement, and Mr. Jim Littleton for his editorial suggestions.

Finally, I would like to thank my parents and all my relatives for their continual support. Last but not least, I would like to thank my husband, Harsha, for his patience, guidance, and support during the hard times.

CONTENTS

FIGURES

TABLES

ALGORITHMS

ABSTRACT

In computer security, cryptographic algorithms and protocols are required to ensure security of data and applications. This research investigates techniques to reduce the energy consumed by cryptographic hash functions. The specific hash functions considered are Message Digest-2 (MD2), Message Digest-5 (MD5), Secure Hash Algorithm-1 (SHA-1) and Secure Hash Algorithm-2 (SHA-2).

The discussion around energy conservation in handheld devices like laptops and mobile devices is gaining momentum. Research has been done at the hardware and operating system levels to reduce the energy consumed by these devices. However, research on conserving energy at the application level is a new approach. This research is motivated by the energy consumed by anti-virus applications which use computationally intensive hash functions to ensure security. To reduce energy consumption by existing hash algorithms, the generic energy complexity model, designed by Roy et al. [Roy13], has been applied and tested. This model works by logically mapping the input across the eight available memory banks in the DDR3 architecture and accessing the data in parallel.

In order to reduce the energy consumed, the data access pattern of the hash functions has been studied and the energy complexity model has been applied to hash functions to redesign the existing algorithms. These experiments have shown a reduction in the total

energy consumed by hash functions with different degrees of parallelism of the input message, as the energy model predicted, thereby supporting the applicability of the energy model on the different hash functions chosen for the study.

The study also compared the energy consumption by the hash functions to identify the hash function suitable for use based on required security level. Finally, statistical analysis was performed to verify the difference in energy consumption between MD5 and SHA2.

Chapter 1
INTRODUCTION

Performance of a software system depends on many factors, for instance, resource utilization, security, and scalability which are important for large-scale web based banking systems [Edge07]. For cryptographic applications, security provided by the application is a primary factor in the measurement of performance. Cryptographic algorithms are complex by design to ensure that the security of an application is not compromised. The computationally intensive nature of these algorithms increases the execution times of the applications that use them, thus increasing energy consumption [Damasevicius12].

A cryptographic hash function, also known simply as a hash function, converts large and possibly variable-size input into small fixed-size data output. Hash functions are used in many software applications that require data integrity, such as digital signatures, video conferencing [Tulu03], control systems, and group communication over secure channels (secure messaging) [Cooley10]. In fact, security protocols and energy consumption are featured on Service Level Agreements (SLAs) for performance in modern technology solutions [Potlapally06].

Energy consumption of hashes has been studied to analyze which hash functions are best suited for mobile devices [Damasevicius12]. Damasevicius et al. found that hash functions are computationally expensive and have large energy footprints . Reducing the

energy consumed by the hash functions by improving their energy footprints while maintaining the same security level and employing the same cryptographic protocols would greatly reduce the energy consumed by the applications that employ these hash functions. This research focuses on the broader question of whether modifying hash functions algorithmically without affecting the level of security provided can reduce energy consumption.

Anti-virus applications use hash functions for different tasks, such as tagging scanned files, recognizing changes, and identifying malicious patterns [Waldin00]. To ascertain the impact of energy consumption of software that use hash functions extensively, this research tagged a power monitoring tool called Microsoft® Joulemeter [Goraczko11] to an anti-virus application to measure power consumed in real-time and calculate the total energy consumed for routine tasks performed by the anti-virus application. The results reported by the Joulemeter in our experiment indicated that up to 85% of CPU power[1] could be consumed by the anti-virus software during a system scan operation. High energy consumption of applications running on battery-powered devices or on large-scale systems make energy conservation an important concern as energy is regarded as a primary component of computing [Ricardo04].

[1] While Energy = Power * Time, in this thesis, the terms power and energy are used interchangeably to address energy consumption in hash functions.

1.1 Problem Statement

The energy required by a system can be reduced by using such techniques as horizontal scaling, which involves adding servers or by vertical scaling in which case, virtual machines are created [Ashkan13]. At the operating system level, multitasking can help in utilizing available CPU power efficiently through simultaneous use of multiple cores [Ricado04]. At the network and communication level, reducing the number of hops in the network and managing network traffic with efficient routing algorithms for efficient processing can help to conserve energy [Mahadevan09]. However, when the application layer is considered, limited work has been done on measures taken from the algorithmic perspective to reduce the energy consumption.

This research takes the approach of modifying applications from an algorithmic perspective by parallelizing input data access in order to improve energy consumption by hash functions. To achieve this, the energy complexity model designed by Roy et al. [Roy13] is considered, which is based on reading data in blocks from all the available memory banks, which reduces the total energy required to access the data.

This work specifically addresses the following problems:

1. Testing the applicability of the Energy Complexity Model on hash functions.
2. If the model is proved to be applicable, redesigning hash functions to decrease energy consumption without changing the hash function's security level.

1.2 Contribution of Study

This work makes a few key contributions:

Firstly, a detailed experimental evaluation of the energy consumption for various hash functions is presented. The hash functions considered are MD2, MD5, SHA-1, and SHA-2 hashes. Our work experimentally validates the applicability of the energy model proposed by Roy et al [Roy13] on these hash functions. A generic way to achieve any desired degree of memory parallelism for a given hash function is also presented. Software engineers can employ this approach to implement energy efficient cryptographic hash functions, as increased parallelism reduces running time along with energy consumed by the hash functions.

Secondly, the energy consumption pattern of specific anti-virus software applications which use hash functions extensively has been studied. During the study, we observed that a large share of a laptop's energy (i.e., battery power) was consumed by the anti-virus during a full system scan.

Another contribution of this work is a set of practical recommendations derived from the experimental study for energy-aware secure hash function design.

1.3 Organization of the Thesis

Chapter 2 contains a literary review of chosen cryptographic hash functions, their properties, and their application. Chapter 3 describes experiments to analyze the amount

of power consumed by the hash functions of an anti-virus application. The focus of Chapter 4 is to understand the construction of cryptographic hash functions, specifically MD4, MD5, SHA-1 and SHA-2. Chapter 5 focuses on the research methodology and techniques used to study optimization of energy consumption in hash functions. Chapter 6 describes the experiments and the results of the statistical analysis, Chapter 7 summarizes the contribution in this research, and Chapter 8 explores possible future work that can be done in the study of energy optimization.

Chapter 2

LITERATURE REVIEW

2.1 Cryptographic Hash Functions

A cryptographic hash function is designed to return fixed-size output data for large and possibly variable size input data. The output returned by hash functions is called *hash value*, *hash code*, *hash sum*, or simply a *hash*. One of the uses of hash functions is to detect accidental or intentional modification of data. For instance, comparing the hash value of data before and after transmission over a network can indicate if the data have been modified during transmission.

The cryptographic hash functions whose outputs are difficult to invert are called one-way hash functions [Oldwasser01], and the hash values are called message digests (MD). These types of hash functions are commonly used in cryptographic and error control services to provide message authentication and ensure message integrity [Damasevicius12].

2.1.1 Properties of Hash Functions

The properties of hash functions explained by Damasevicius et al. in [Damasevicius12] are as follows:

- Uniformity - There should be statistically significant differences between the hash values generated for uniformly distributed data and non-uniformly distributed data.
- Spread - Similar input data with slight changes should result in different hash values generated by the hash functions.
- Determinism - The hash value returned for a particular input should be the same every time it is input to the hash function.

Some of the required properties that need to be satisfied by hash functions are as follows [Stallings06]:

1. Pre-image resistance: For a given hash value, finding the input data corresponding to the hash value should be difficult. This property is related to the one-way hash function.
2. Second pre-image resistance: For a given input data message, finding another data message that generates the same hash value should be difficult.
3. Collision resistance: Finding two samples of data which map to the same hash code should be difficult. This pair leads to a cryptographic hash collision and a hash value that is at least twice as long as the second pre-image resistance is required to avoid it.

A one-way hash function is defined as a function for which generating the message content using only the hash value is computationally infeasible. Practical one-way hash functions are designed so that different data messages that might be input to the function

during the lifetime of the function do not generate the same hash value. Changing a single bit in the message changes the hash value from an avalanche effect due to iterative convolution steps and the change is statistically modeled to maintain a minimum Gaussian difference between hash values [Karras04].

2.1.2 Applications of Hash Functions

Hash functions have a wide range of applications in computing. Some uses of hash functions are memory management (hash tables), error detection and control to maintain data integrity, cryptographic applications for message authentication, and encryption. Hash functions are also used in computer graphics, computational geometry, searching, and sorting [Damasevicius12]. Hash function applications are not limited to digital signature and data integrity. Because of their one-way and randomness characteristics, they are used in proof key derivation and pseudo-random number generation [Damasevicius12].

Cryptography involves secure protocols that address a range of security issues. In the modern era, cryptography includes encryption and other aspects of security such as techniques and protocols to ensure authentication, non-repudiation, and integrity objectives. In today's world, cryptographic algorithms are measured by their computational hardness [Mishra15]. Information, Communication, and Technology (ICT) rely on cryptographic hash functions and protocols for protection from the dynamically changing threat scenarios [Mishra15].

Cloud computing involves sharing of network and storage resources in the computing environment [Neto11]. Here, cryptography ensures confidentiality using encryption primitives and data integrity, and authentication primitives ensure non-repudiation of the message/data. Some of the popular cryptographic algorithms that support the primitives mentioned above are AES, DES, SHA, RSA, and ECC.

2.1.3 Classes of Hash Functions

1. Checksum Algorithms: Cyclic redundancy check algorithms such as CRC-32 are used for error detection where the output hash code is used to identify accidental or intentional modification of data being transmitted [Damasevicius12].
2. Perceptual hashes: Perceptual hashes are digital fingerprints of media files which are used to check for modification of data content [Klinge14]. These are used to recognize copyrighted data. For instance, books or articles which hold copyrights can be hashed and stored. If the same article or book is uploaded, the hash code of the uploaded document may be compared with the original hash code in order to detect a plagiarized copy of copyrighted data.
3. Message Digest Algorithms: Message digest algorithms are cryptographic algorithms such as MD2, MD4, and MD5 which rely on an avalanche effect to produce drastically different hash values with slight changes in input data. These are used to check data integrity [Preneel94], and message digests are used in software version control systems like Git to uniquely identify files and versions. Message digests are also used to maintain a hash table data structure as it has O(1)

complexity to lookup keys stored in the data structure [Konheim10]. Another use of message digests is in verifying contents of a downloaded file [Perrin07].

2.2 Hashes in Anti-virus Applications

Security of files in computers has multiple facets. When retrieving files stored on a hard disk, data are checked for corruption or unintended changes [Waldin00]. In computer networks, a file's security can be compromised during transmission across a network. A potentially compromised file can be checked for changes in content by calculating and comparing its hash value with that of the original file. According to the patented work of Waldin et al. [Waldin00], every critical sector of the file maintains identification, hash values of each scanned sector, the date of update by an anti-virus module and a version number of the anti-virus module. Message digests are used to affix a digital signature to the critical sector before storing it or transmitting it over the network to recipient computers. The anti-virus module at the recipient side scans the critical sector of the received file, calculates the hash value of the data received, and compares both to verify whether the file has been modified during transmission [Waldin00].

When a file is scanned for the first time, a hash value is calculated and stored in a hash table maintained by the anti-virus software in an internal database. Since hash functions are computationally intensive, it is optimal to compute the hash value of the entire file only if there is a mismatch with the stored hash value. However, critical sections of files are scanned to ensure that unintentional changes to files are detected [Waldin00].

2.3 Energy Consumption in Hash Functions

Damasevicius et al. [Damasevicius12] have shown that hash functions are computationally intensive and are used to ensure security in applications running on battery-powered mobile devices. Energy efficiency and quality characteristics of several hash functions were analyzed in their research by performing avalanche and chi-square tests on the energy usage in Java-enabled smart phones.

Damasevicius12 et al used the percentage of drain in battery caused by running Java applications. The capacity of the battery of the mobile devices (smart phones) was used to determine the energy consumed. The experiment was performed on seventeen existing hash functions; graphs were generated to analyze which hash function consumed the least energy. These hash functions were both cryptographic and non-cryptographic hash functions. With the obtained results, the researchers identified and proposed energy-efficient hash function SV1 for cryptographic applications and CRC-16 for non-cryptographic applications for mobile devices. However, their research did not study the energy drawn by hash functions from an algorithmic perspective.

2.4 Energy Complexity Model for Algorithms

A new approach to design algorithms considering energy complexity during the early stages of the design has been proposed in the research by Roy et al. [Roy13]. The authors found that the energy consumed by an algorithm depends on the type of data being read, the level of parallelism in data access and effective usage of the data read by

the algorithm to reduce the number of I/O requests. Based on the observed results, the primary methodology proposed to design energy aware algorithms is the utilization of parallel memory banks in the RAM. By arranging data in memory, and reading data in parallel, algorithms which use a single data sequence have shown reduced energy consumption. This is of particular interest as hash functions use single data sequence input and can utilize data from multiple read operations.

Chapter 3

MOTIVATION

3.1 Energy Consumption by Anti-Virus Programs

With the increase in malicious software and programs in computing, the demand on anti-virus applications has increased dramatically [Massengale08]. In order to support the secure transfer of data and the maintenance of security of internal data, an anti-virus application runs at all times and listens to specific open ports. As anti-virus software uses hash functions [Waldin00]. The research in this thesis began by analyzing the instantaneous power consumed by the anti-virus software for some routine tasks performed on a laptop. During the experimentation, the main anti-virus process was observed to be idle most of the time but spawned new processes to perform tasks. Occasionally, the anti-virus processes accounted for 85% of the total power drawn by the CPU from the battery. A process for a task such as a full system scan was typically executed for half hour, and the energy consumed by the entire process was proportionally high. In order to reduce the energy consumed by anti-virus software, redesigning key components of algorithms for energy efficiency was necessary.

Some of the scenarios considered for experiments are based on [Sobol94] and [Harish14]:

1. File system scan.
2. External hard disk drive scan.

3. Virus definition update.
4. Attachment download from an email program with automatic virus scan enabled.
5. Third party software and operating system files scan to detect vulnerabilities.

3.2 Experimental Setup

The test setup for the experiments consisted of a laptop with an Intel i5-3427U processor with 256 KB L2 cache and 3 MB L3 cache, 4 GB DDR3 1333 MHz RAM running Windows 7 SP1, Kaspersky anti-virus software version 14.0.0.4651.

A power measurement tool developed by the Microsoft® Research group called Joulemeter [Goraczko11] measured the instantaneous power consumed by the CPU. Using this tool, a user can tag any CPU process and measure the real-time power it consumes. When run on battery power, Joulemeter tracks CPU utilization and screen brightness and subsequently estimates the power usage of the tagged processes. Using power and time taken by the application, energy consumed can then be calculated. Joulemeter is designed for Windows XP and Windows 7.

The metrics logged were power consumed by the tagged processes, power consumed by the CPU, and the total power consumed by the laptop including auxiliary boards and chips, memory controllers, and graphics cards. These metrics measured by Joulemeter were logged for a period of time. Graphs were plotted to visually analyze the instantaneous power needed to perform various tasks initiated manually to test various scenarios.

3.3 Experimental Results

Full System Scan: A full system scan is a scheduled event in most computers. It is a periodic scan, scheduled to run daily or weekly based on the system settings. Figure 1 shows the power consumed during a full-system scan with the graph's results averaged over five iterations of the experiment.

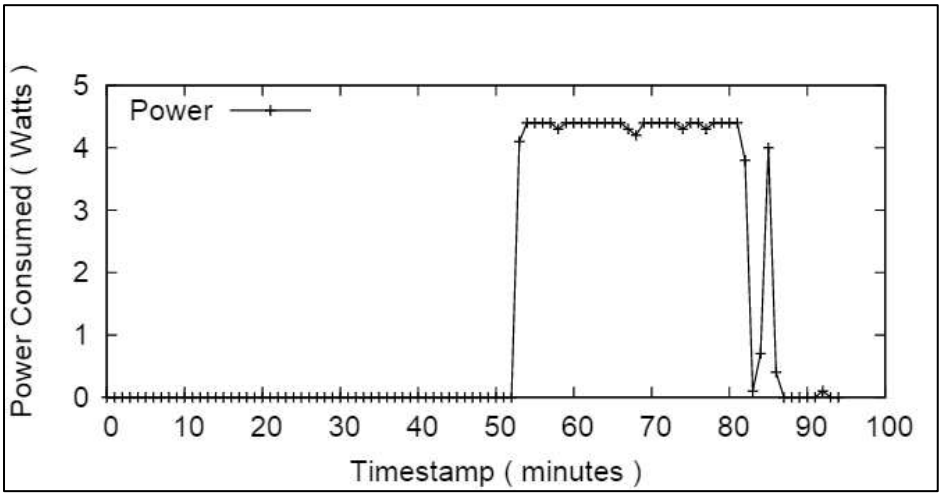

Figure 1: Power Consumed by Anti-virus Process during a Full System Scan.

Figure 2 shows the ratio of power consumed by the anti-virus software and the total power consumed by the CPU during the full-system scan. The ratio remains high, between 0.6 and 0.8 during most of the scan. Results are averaged over five iterations of the experiment.

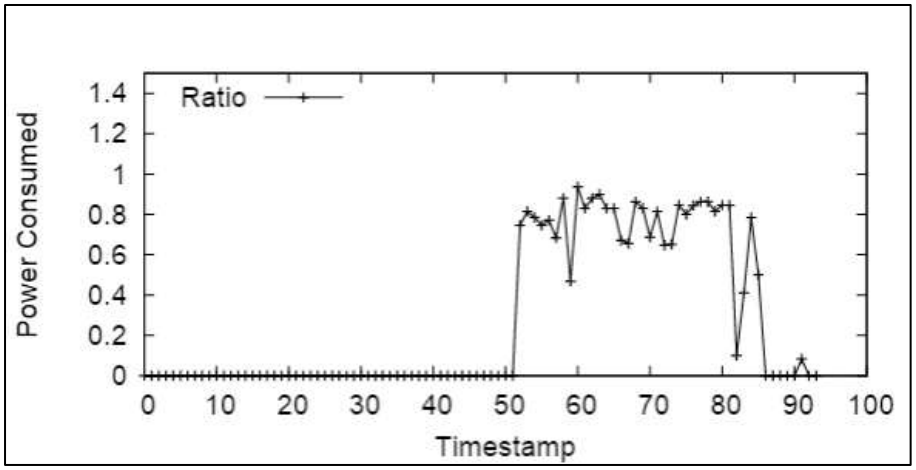

Figure 2: Ratio of Power Consumption by Anti-Virus Process and Whole CPU

Virus Definition Update: The virus definition update process is triggered manually over the course of a day. Typically, this process automatically runs multiple times a day based on settings. It is designed to receive RSS or Atom feeds from virus definition provider web services [Yensen05]. Figure 3 shows the instantaneous power consumed by the process to update virus definitions.

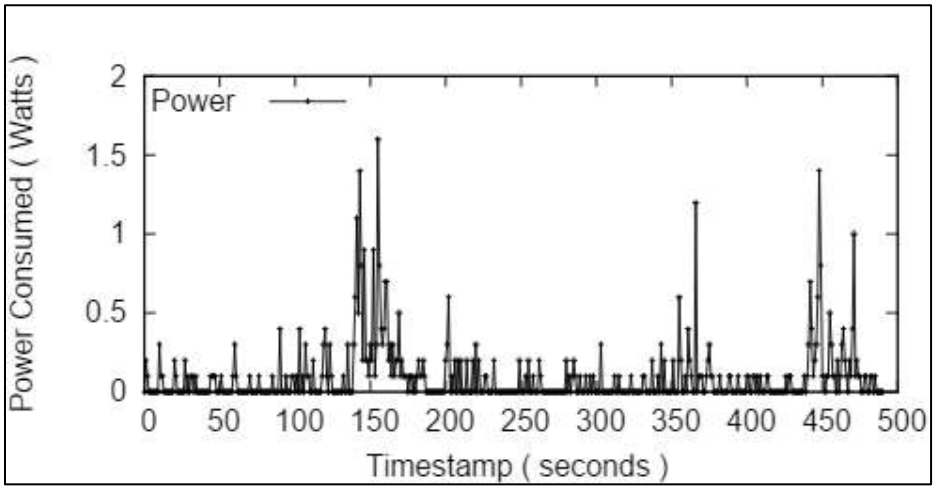

Figure 3: Power Consumed by an Anti-Virus Process during Virus Definition Updates

Email Attachment Download: Figure 4 shows the power consumed when an attachment is downloaded from Microsoft® Outlook®. The anti-virus software had internet security enabled for this experiment so that it automatically scanned any downloaded file. Large attachments were downloaded from emails during this experiment.

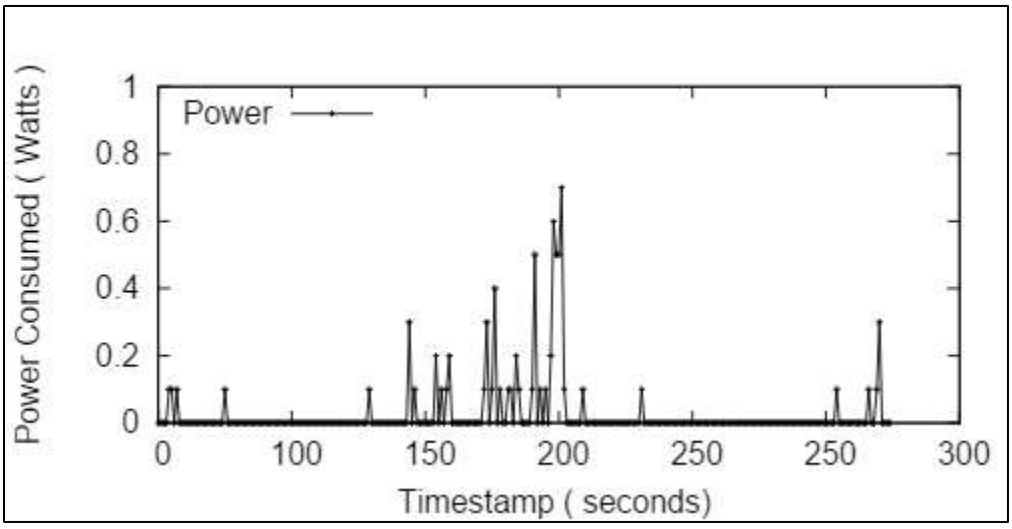

Figure 4: Power Consumed by an Anti-Virus Process while Downloading Outlook Attachments

External Disk Scan: Figure 5 plots the power consumed when an external hard disk drive was scanned. A solid state flash drive was used for this experiment. The flash drive contained video files, each of 7 GB size, and these files were scanned while the data resided on the external drive. The process's run-time was lower than in previous experiments as the anti-virus software stored a hash key while storing the data on the external hard drive. So it only needed to recompute the hash and compare the keys to determine if data changed. In this experiment, data in the files remained the same and, hence, the anti-virus software did not rescan the entire files.

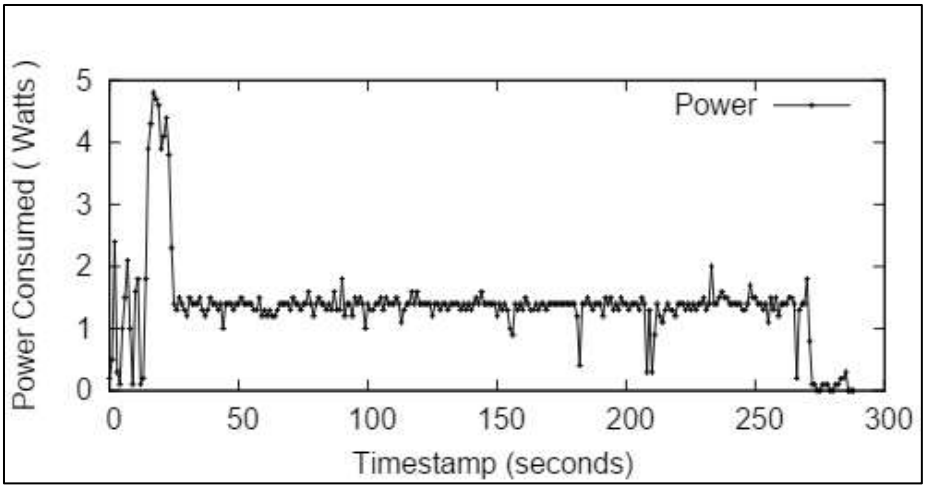

Figure 5: Power Consumed by an Anti-Virus Process while Scanning External SSD Drive

Executable File Scan: In the final experiment, large executable files residing on the hard drive were scanned. Figure 6 shows the power consumed when the file scanned was a Microsoft® Office 2010 Installer, which is an MSI file of 1.6 GB size. The power consumed was similar to that consumed while doing a system scan, because individual components within packages like MSI have hash codes that are computed and compared. This is similar to the case where different types of files in the hard disk were scanned during a system scan.

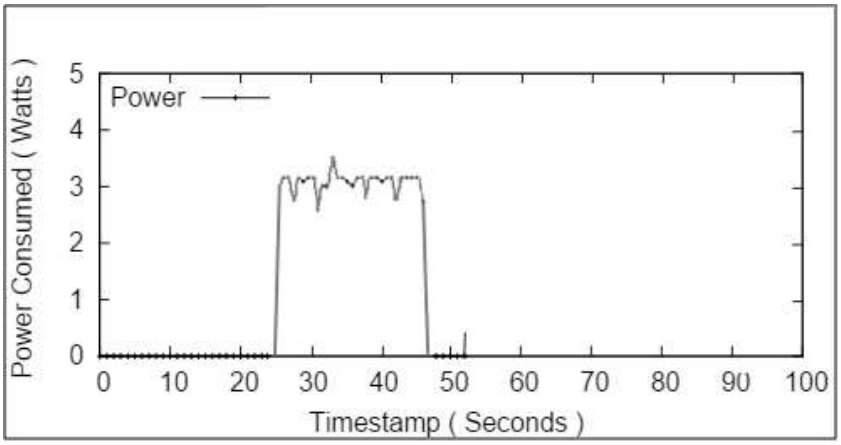

Figure 6: Power Consumed by an Anti-Virus Process while Scanning a Large EXE File

3.4 Inference Based on Experimental Results

Most anti-virus software incorporate computationally expensive hash functions [Amamra12] which consume a high proportion of available power in a device, as seen in Figure 1. This level of power consumption might form a security gap in which an attacker can exploit the power-consumption trait of anti-virus software to perform battery-draining attacks by continuously sending data that triggers the anti-virus software in the target machine [Harish14]. In this way, the device is forced to perform computationally intensive tasks which would drain the battery at a quicker rate. This is called a battery draining attack and is effective on battery-powered devices such as smart phones, PDAs, remote control modules, and laptops [Martin04]. To counter or reduce the effects of these types of attacks, existing hash functions could be redesigned to reduce power consumption and the impact of attacks on applications that use computationally expensive cryptographic hash functions for security. Intrusion detection systems are examples of such applications.

In the experiment to update virus definitions (see Figure 3), the size of the update package received can determine the total energy consumed. As this process runs occasionally and many scheduled runs have no updates, the total energy consumed by this process is negligible compared to other anti-virus' processes.

When scanning files downloaded from an email program, every sector of the file is scanned for suspicious patterns, and data in critical sectors is compared to hash values computed on the downloaded files to ensure that all vulnerabilities are checked when

external data enters the system, particularly when the use of digital signatures exclusively is not sufficient [Waldin00]. The run time of this process is higher and the energy consumed is significant, especially during web browsing, as many web resources are downloaded and so require checking by the anti-virus program in the background.

The experiment involving the scanning of external hard disk revealed the optimizations incorporated in the anti-virus software by the software provider. Metadata about each file indicates if the file was modified since the last scan, and in certain conditions it indicates whether checking critical sectors of files is sufficient. This conserves energy and reduces scan process time. The I/O operations to read data from disk are primarily the reason for energy consumed by this process [Xun13]. Calculating hash values for modified files caused the spikes seen in Figure 5, and optimizing the hash functions would reduce the energy consumed by the entire process.

Chapter 4

ALGORITHMS OF HASH FUNCTIONS

In the previous chapter, hash functions in anti-virus applications were shown to consume a large amount of power. This chapter describes the construction of a generic hash function and applications of MD2, MD5, SHA-1, and SHA-2. The applications in this section will help us analyze the areas where design of energy-efficient hash functions can help conserve energy.

4.1 Merkle-Damgård Construction

Merkel-Damgård construction, shown in Figure 7, is a cryptographic algorithm that uses collision resistant one-way compression functions to build collision resistant cryptographic hash functions [Damgård90]. Hash functions MD5, SHA-1, and SHA-2 were designed using this construction.

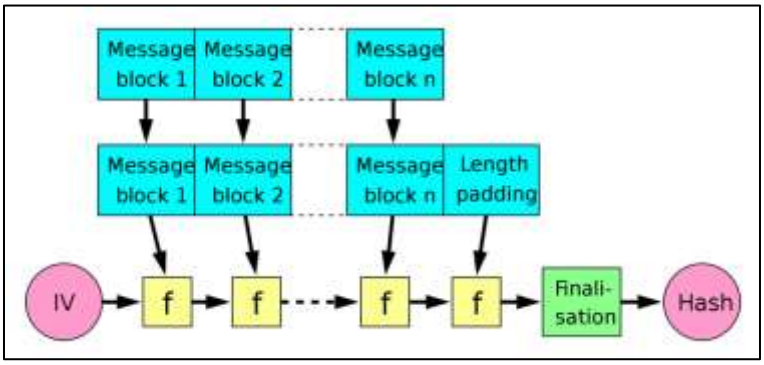

Figure 7: Merkle-Damgård Construction

Merkel-Damgård construction consists of the following steps:

1) Message padding

    i) As shown in Figure 7, for arbitrary sized data, Merkel-Damgård construction applies MD-compliant padding functions to create an output. The output size is a multiple of a fixed number.

    ii) The output from step 1 is divided into blocks of fixed size again, and a compression function is applied to each block sequentially. The result of each block is combined with the results obtained from the compression of the previous block.

2) Length padding: The length of the input message should be equal to a predetermined length to ensure secure construction, and so padding bits are added to the input message to achieve this.

3) *IV* is an initial vector with a fixed value that is fed to the one-way compression function *f*, which is applied to each message block from step 2 to produce an intermediate result that will be fed as input to the next compression function. The final block is padded with zeroes as required. Further, a process called *Finalization* is applied to the result to harden the hash function. This step involves adding additional rounds of the compression function to convolute the message, thereby making the output more one-way after each iteration.

Table 1 lists the hash functions that are considered in this research. For any arbitrary size of input, the size of output produced is listed in the table. The table also lists the

known types of attacks and the complexity at which hash functions can be vulnerable to the attacks.

Algorithm	Output size (bits)	Attacks
MD2 [Kaliski92]	128	Preimage and collision attack [Rogier10]
MD5 [Rivest92A]	128	2^{64} complexity (collision found) [Dobbertin96]
SHA-1 [Eastlake01]	160	2^{80} complexity, theoretical attack found in 2^{61} [Schneier05]
SHA-2 [Sklavos03]	256	No vulnerability found at the time of writing of this thesis.

Table 1: Hash Functions – Outputs and Known Attacks.

4.2 Message Digest 2 (MD2)

Message Digest 2 (MD2) is a cryptographic hash function developed by Ronald Rivest [Kaliski92]. It is used in public key infrastructure as part of certificates generated with MD2 and Rivest-Shamir-Adleman (RSA) algorithm.

For any length input, MD2 generates a hash value of 128 bits in length, which is represented as a 32-digit hexadecimal number. MD2 was found to be vulnerable to different types of attacks, such as preimage and collision attacks, and therefore is no longer considered secure [Rogier10]. This research focuses on analyzing the energy consumed by MD2 and reducing the energy consumed by MD2 as the same technique can be extended to advanced cryptographic algorithms. Cryptographic algorithms used in applications at the time of this research, e.g., MD5 [Rivest92A], SHA-1 [Eastlake01], and SHA-2 [Sklavos03], are algorithmically similar to MD2.

4.3 Message Digest 5 (MD5)

MD5 is a widely used message digest algorithm which produces a 128-bit hash code, represented as a 32-digit hexadecimal string. As MD5 is a one-way algorithm, it is used to ensure data integrity while transferring messages over a network.

Ronald Rivest designed MD5 in 1992 [Rivest92A]. This algorithm was an enhanced version of a previous hash function named MD4 [Rivest92B]. Additional processing was added to the algorithm of MD4 to make it more secure while designing MD5. In 1996, flaws were found in MD5 algorithm, and use of Secure Hash Algorithm-1 (SHA-1) was recommended [Dobbertin96]. MD5 was tested for vulnerabilities in 2005, 2006, and 2007, and as a result MD5 is no longer considered safe for use in digital signatures or SSL certificates for digital security [Stevens12].

4.3.1 Applications of MD5

A typical application of MD5 is to generate a digital fingerprint for a piece of a message to prevent tampering. For instance, when a file transferred across a network is modified by human intervention, network error, or an intruder, hash values generated using MD5 can be used to detect the change. For instance, the file is sent with its hash code to the receiver. At the receiving end, the hash code of the received file is generated using the MD5 function. A mismatch in the hash code generated and received at the receiving end indicates that the message is not the original message because of the one-way property of MD5.

In encryption and decryption technology, a user's password is encrypted using MD5, and the hash value obtained is stored in the file system [Wang13]. When the user logs in using his or her credentials, the user's password is transformed to its hash value using MD5 and compared to the corresponding hash value stored in the file system, thus authenticating the legitimate user while also preventing system administrators' access to user passwords.

Yan et al. have researched the application of MD5 in control systems [Yan12]. Information security and safety are crucial in industrial control systems. Traditional systems were built with safety related systems like Emergency System Shutdown to reduce risk of hazards during system operation to an acceptable level. However, the issues of hacking and malicious attacks were not addressed and led to system failures.

Industrial control system security is an important factor in modern plant automation as the control system is usually a subsystem of a larger network [Yan12].

The safety-related system in plant automation control systems is critical because malfunctioning of the safety system may lead to disasters such as a blast, human injury, or a gas leak. It is required that the communication between peers across a network be guaranteed and failsafe in the safety system. The authors describe earlier systems where safety-related modules used sequence numbers and timestamps to ensure validity of messages received, a security key to ensure communication of the message between the right roles, and CRC to ensure the message is intact [Yan12]. However, by recalculating the CRC after data are modified and changing the sequence number and timestamps, attacks on the control system can be carried out. In addition, disclosure of the key constitutes a security compromise. To overcome all these problems, hash functions such as MD5 and SHA-1 have been used for secure communication, and these functions are faster compared to cryptographic algorithms such as AES, DES, and TDES, which were used in earlier systems. The authors in [Yan12] chose MD5 for its speed compared to other hash functions. In their approach, Hash Message Authentication Code (HMAC) replaces CRC to address the errors such as repetition, deletion, insertion, disorder, natural data corruption, delay, masquerade, and intentional data modification. HMAC is intended to replace CRC but the downside is that the message code produced by MD5 is 128 bits long (16 bytes) and the CRC is 16 bits or 32 bits long. The resulting longer error detection code requires higher bandwidth and reduces some of the real time data transfer ability of the system.

MD5 is used in the field of telemedicine for video conferencing between patients and physicians [Tulu03]. Several applications are dedicated to this service and run on dedicated ISDN lines as bandwidth, security, and privacy are amongst the primary concerns in telemedicine [Tulu03]. Session Initiation Protocol (SIP), designed for multimedia communications over IP-based networks, is the signaling protocol used as it has a built-in security mechanism. In order to support multiple video conferencing users simultaneously, applications require secure authentication and communication mechanisms. CGUsipClient is a SIP-based video conferencing telemedicine application that provides authentication and communication channel guarantee using MD5 hashing [Tulu03].

4.4 Secure Hash Algorithms (SHA)

In 1993, the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA) proposed the first algorithm in a series of cryptographic hash algorithms and named it Secure Hash Algorithm-0 (SHA-0). In 1995, SHA-128 algorithm was proposed as the United States Federal Information Processing (FIPS) PUB 180-1 standard with a message hash value of 160-bits (20 bytes) length [Eastlake01]. SHA-1 has since become the DSA Digital Signature Standard. In 2003, FIPS PUB I80-1 expanded the SHA series, proposing SHA-256, SHA-384, and SHA-512 algorithms [Sklavos03], which produce hash codes of length 256-bits, 384-bits, and 512-bits, respectively. SHA-128 and SHA-256 are commonly known as SHA-1 and SHA-2.

The SHA series of algorithms were designed by departments of the United States federal government as the government requires secure hash algorithms to protect classified and sensitive information [Wang13]. SHA-1 and SHA-2 are algorithmically similar and their construction is based on the design of message digests MD4 and MD5. They are most commonly used in the transport layer of computer communication, SSL-based applications, and revision control systems like Git [Eastlake01].

4.4.1 Applications of SHA-1

SHA-1 is used in cloud computing where users may store sensitive data [Mishra15]. Security of the data stored on the cloud is an important factor and challenging due to the shared nature of the network and access control of data stored on the cloud [Carlson14].

SHA algorithms have also been used in the design of secure communication systems, a novel application based on Multilayer Perceptron (MLP) Neural Network (NN). This system is used for evaluation and production of digital fingerprints [Karras04]. SHA-1 was chosen to compute the digital fingerprint due to its one-way property.

Digital fingerprint functions use mathematical algorithms to generate a string of digits representing the content of the electronic message [Karras04]. This fixed length string is called a digital fingerprint and is generated using one-way hash functions such as MD5, SHA-1, or SHA-2. If an intruder tampers with the message, the digital fingerprint of the message changes. At the receiving end, the fingerprint is recomputed and compared with

the fingerprint of the original message that is transmitted with the message to ensure that the message content was not altered.

4.4.2 Applications of SHA-2

Network applications using SHA-2 Group-Oriented Communication form an important class of communication applications. These applications are used for different modes of communication like text and voice chats, video broadcasting, and other content distribution applications. A secure way of communication between group members and management of secure keys used by members when they join the group and leave the group is handled within these applications [Cooley10]. GROK is one such general purpose cryptographic building block that ensures secure communication between group members. In GROK secure group communication application, confidentiality, integrity, and authenticity of each message destined to the group are implemented. The application uses the Advanced Encryption Standard (AES-256) cipher block chaining mode and the Secure Hash Algorithm-256 (SHA-2) to ensure that the security measures in GROK are failsafe. GROK uses hashes for configuration agreement between two communication endpoints. It is important for endpoints to agree on which algorithms and configuration settings will underlie their security. In order to achieve efficient security in the target communication environment, GROK undergoes multi-phase cryptographic parameter and configuration agreement. Each message includes a global configuration identifier defined as the prefix of a configuration value, e.g., Prefix (SHA2 (cipher ID, message)). Here the prefixing step uses the hash value generated by SHA-2 along with a cipher ID that is established during the configuration agreement by the communicating endpoints.

Ravilla et al. [Ravilla15] consider the security perspective of secure routing in mobile ad-hoc networks (MANET). The authors state that constructing the Message Authentication Code (MAC), which is a digital fingerprint of the message generated using SHA-2, provides faster execution in software than symmetric block ciphers like Data Encryption Standard (DES).

Attacks on ad-hoc networks are classified as either passive or active attacks [Hongmei02]. Passive attacks involve eavesdropping, and the security of data transmitted over the network may be compromised. In active attacks, the data could be altered intentionally at a malicious node in the network or data might change accidently due to errors in the network such as radio signal interference or hardware glitches. Confidentiality, availability, authenticity, and non-repudiation are important features in the protocols used to provide security in ad-hoc networks.

According to Hongmei et. al. in [Hongmei02], an implementation of HMAC-SHA2 fits well in the security protocol. However, a downside noticed was increased processing time. One of the reasons cited for the increase in time is the complexity involved in the SHA-2 hash algorithm. Since the original SHA-2 algorithm is used by the authors, the energy consumed will also be high as the application runs for a longer time.

Chapter 5

RESEARCH METHODOLOGY

5.1 Energy Complexity Model

To redesign hash functions so that they consume less energy, an asymptotic energy complexity model designed by Roy et al. [Roy13] was considered. The model assumes that memory is divided into *P* memory banks and stores data in blocks of size *B*. The model is inspired by the DDR3 RAM architecture in which *P* memory banks with blocks of size *B* can be accessed in parallel as shown in Figure 8.

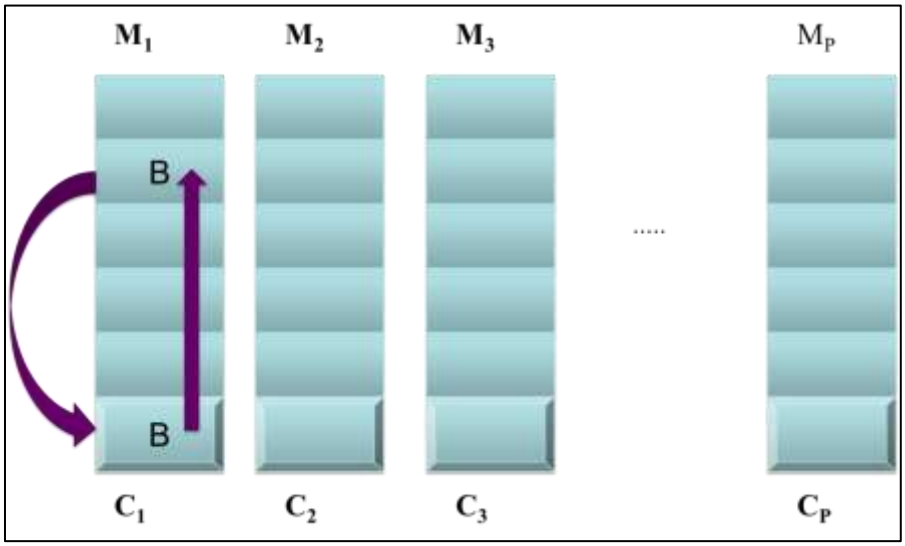

Figure 8: Energy Complexity Model [Roy13]

The next section describes parallel access of memory blocks, where the input message was arranged across the banks to support desired level of parallelism. This reduces the

number of I/O requests to access the entire set of data. For instance, reading data with a single stream might read data that cannot be used in the current iteration cycle of the algorithm. Arranging the data in blocks ensures that the data needed in the immediate cycle of processing is read from the memory. Due to the efficient use of memory, the time required to read all data is reduced. Thus, the energy consumption of a hash algorithm can be reduced and has been derived as follows in [Roy13].

$$E = T + (P * B) / I$$

where *E* is energy consumption*; T* is total time taken*; P* is number memory banks*; B* is size of each memory block*; I* is number of parallel input/output operations made by the algorithm.

5.1.1 P-way Parallelism for Hash Functions

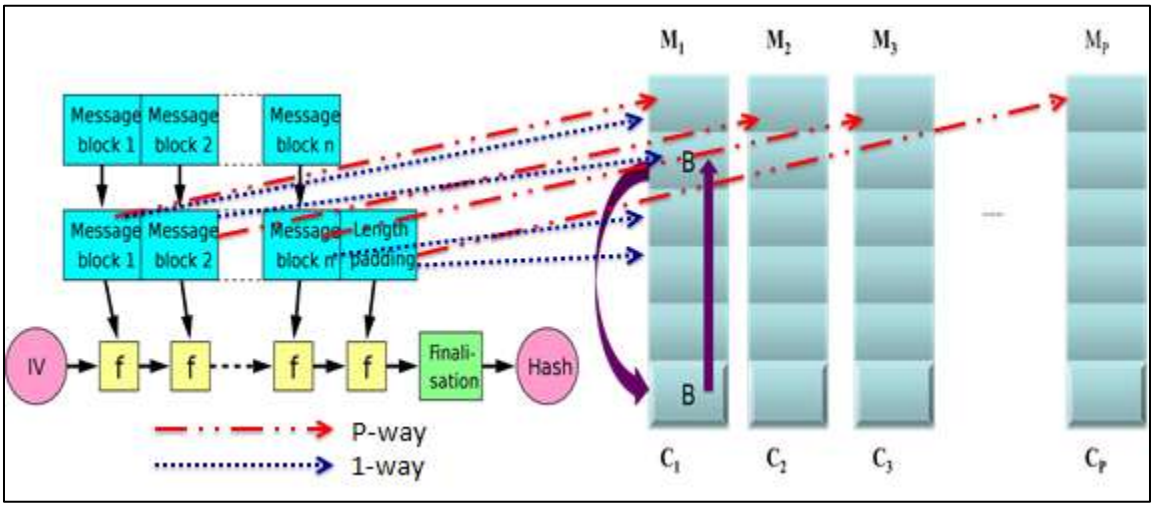

Figure 9: Parallelization of Hash Function's Input Blocks

In Figure 9, Merkle-Damagard construction [Damgård90] (see Figure 7) and the energy complexity model **(**see Figure 8) [Roy13] are combined to achieve the desired level of

parallelism of input blocks 1, 2, 4, and 8-way. For instance, for 1-way parallelism, all the message blocks in step 2 of Merkle-Damagrad construction are mapped first to memory bank M_1. On the other hand, for *p*-way (i.e., 2, 4, and 8-way) the message blocks are laid across *p* memory banks M_1 through M_p, so that the message blocks can be accessed in parallel.

Further, this section describes the technique used to parallelize the available input. For hash functions, the input data (the message) can be considered to be in vector form. In order to perform this experiment, controlled access to the input was used to lay the data across the memory banks, thereby allowing usage of all data read in parallel.

Given an arbitrary input data in vector form, a logical mapping of the data has to be created so that the data can be accessed in a *p*-way parallel fashion. In today's technology and architecture, *p* can range from 1 to 8 as there are eight memory banks in DDR3 architecture [Roy13].

```
Input:  Page table vector V, jump amount j.
factor = 0;
for (i = 0; i< N/B – 1; i++)
     if i > 1 and ((i*j) (mod N/B) = 0) then
          factor = factor + 1;
     end
     V_i = (i * j +factor) (mod N/B);
end
```

Algorithm 1: Function to Create Ordering of Blocks

The first step is to ensure that the blocks are allocated in a specified *p*-way. The mapping function converts the input vector *V* of size *N* into a matrix *M* of dimension *N/B * B*.

Each block of size B is further divided into strides of size *s*, which results in *B/s* logical strides of size *s* in the matrix *M*. In the mapping function, depending on the specified *p*-way, *P*B* blocks are logically assigned to the memory banks in each iteration. To utilize spatial locality, all strides of a block are placed in the same bank. Finally, the mapping function contains the logic to map M_{ij} to the memory banks, which represents the j^{th} stride in the i^{th} block.

In the second step, a page table of size *N/B* is created, as shown in Figure 10. This maintains the order of the *B* blocks in the *P* memory banks. This ordering defines the way to access the blocks from the memory in *p*-way. *p*-way represents the parallel access method of the memory blocks. When *P* = 1, it means 1-way sequential access where all the blocks are present in a single bank and the page table is populated consecutively. For *P* = 4, input data is stripped across four banks such that four blocks of data can be accessed in parallel. This is illustrated in Figure 8 where the page table is populated in jumps of four in a round-robin fashion.

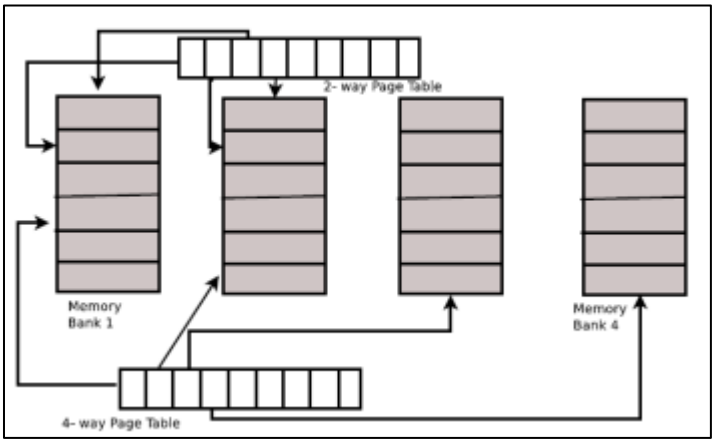

Figure 10: Memory Layout for Achieving P = 4-way Parallelism

5.2 Optimizing the Hash Algorithms

The memory layout in Figure 10 shows mapping of the logical data blocks to physical data blocks, which is done by the mapping function. The mapping function requires additional data structures to place data in blocks which introduces an overhead. This overhead may be counterproductive as the energy required for mapping the memory layout might overshadow the energy saved by redesigning the algorithm. In order to overcome this problem, the following code optimization techniques have been considered.

Optimization techniques:

1. Bit shift operation: Most of the input data to hash functions are in powers of 2, so all the multiplication, modulus, and division operations are replaced by bit shift operation.
2. Register variables: The variables that need multiple computations are stored in register variables where access time and latency are lowest and less energy is consumed for each access [Tiwari94].
3. Minimizing the call to the map function: Since calling the map function for each memory access incurs additional overhead, the map function is made inline to optimize the code. Also, the strides of a data block are placed consecutively in the memory, thereby utilizing the spatial principle of locality to reduce overhead by enabling only one call for each iteration of data access.

5.3 Engineering the Algorithm

In a hash function, the sequence of accessing input data is crucial to obtain the correct hash value output. To obtain the desired level of parallelism using the memory layout scheme designed in this research, while maintaining correctness of the hash function, the following techniques were considered:

1. Analyze the common access pattern of the input data to lay the data blocks in the memory blocks.
2. Create the vector with the desired access sequence using the mapping function, which helps in parallel access.
3. If there are iterations involved in accessing the data, in each iteration a multiple of *P*B* or sub-sequences of size *P*B* are accessed in the order from step 2.
4. A large vector is created by adding the sub-sequences in arbitrary order for each iteration.
5. A large vector of most the likely sequence is created if there are no sub-sequences of data due to multiple iterations of data access.
6. The data from the large vector is used for hash functions that require multiple input vectors for multiple iterations.

The steps mentioned above lead to providing a controlled access to the input data so that it can be placed in stripes across the available memory banks. The placement of data is a critical requirement in this research as it allows *p*-way parallel data access without affecting correctness of the algorithm.

Chapter 6

EXPERIMENTATION AND RESULTS

6.1 Experimental Setup

Experimentation began with MD2, which was amongst the initial hash functions developed for security-related applications [Kaliski92]. The primary goal of the experiments was to record any variation in energy consumption in MD2 after incorporating the energy complexity model and redesigning its algorithm to use the DDR3 memory architecture. Varying the degree of parallelism of data access from 1, 2, 4 to 8-way allows the analysis of energy consumption of MD2. The results of these tests yielded a better understanding of the hash functions and laid the foundation to redesign and test advanced hash functions such as MD5 [Rivest92A], SHA-1 [Eastlake01], and SHA-2 [Sklavos03], which are being used in many applications at the time of this research [Karras04].

6.2 Hardware Setup

An Apple® Macbook© laptop was used to perform the experiments described below. The system configuration was as follows:

- Processor: Intel i5- 520M with 2 cores running 2 hyperthreads each, with 2.4 GHz processor clock. RAM: 4 GB DDR3 1066 MHz memory with 8 memory banks

- Operation system type: Mac OS© X Lion 10.7.3
- Cache size: 256 KB L2 and 6 MB L3
- Disk size: 499.25 GB solid state drive.

The total power consumed by the application running the redesigned hash functions is measured by Apple® laptop's Hardware Monitor (HM) tool [Softwar15]. The power measured includes RAM, background leakage, and CPU power. Here, the HM tool directly reads Apple's System Management Controller (SMC). Laptop's software processes which are not associated with the experiments were terminated prior to the measurement. The measurement reported by the HM tool can be considered accurate because it is an auxiliary processor, which is independent of the main computer, and sensor readings therefore do not include the energy consumed by the monitoring infrastructure.

6.3 Comparison of Energy Consumption by Varying *P*-way Parallelism

The experiment performed entailed parallelizing input data to MD2 in 1, 2, 4 to 8-way. The 16 bytes of blocks of data in the block processing step of MD2 algorithm are spread across *P* memory banks based on the selected *P*-way. The power consumption is measured for 1, 2, 4, to 8-way parallelism of the input of MD2 hash algorithm. The energy consumed was calculated with the power measured and time taken to execute the entire algorithm. The tests were repeated 50 times and the mean values of measurements were recorded.

Figure 11 displays the results obtained by varying the parallelism from 1, 2, 4 to 8-way while keeping the input message constant. The graph shows the redesigned algorithm for MD2 consuming less energy as parallelism is changed from 1, 2, 4 to 8-way. The energy consumption by MD2 was improved by 1.55% with 8-way compared to 1-way.

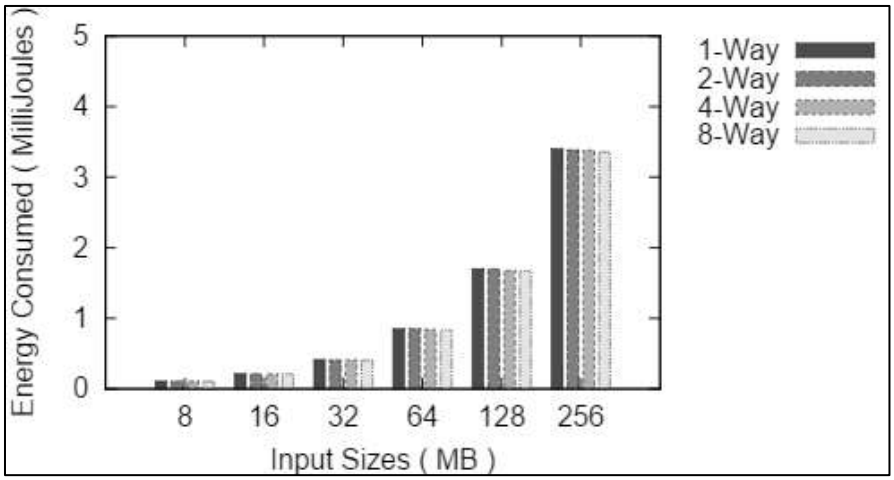

Figure 11: Energy Consumption of Redesigned MD2 with P-way Parallelism

Similar experiments were conducted on the MD5 hash function by varying the *p*-way parallelism. The algorithm was redesigned to utilize the spatial locality of data, and data is read in blocks. Figure 12 shows energy consumption from the experiments on MD5. The difference in energy consumed is more pronounced in the case of large input data sizes in MD5 (5.11%) compared to the difference in case of MD2 (1.55%).

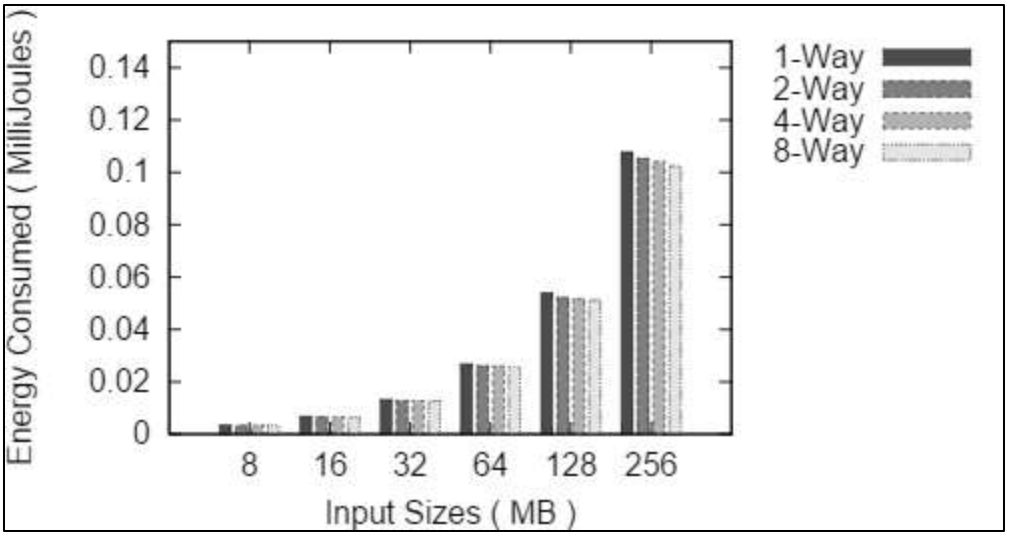

Figure 12: Energy Consumption of Redesigned MD5 with P-way Parallelism

Experimentation on SHA-1 and SHA-2 show similar trends with reduction in energy consumption complying with the energy complexity model used to redesign these hash functions. SHA-1 shows 3.56% decrease while SHA-2 shows 2.26% decrease in energy consumption. The results of experiments on SHA -1 and SHA-2 are shown in Figure 13 and Figure 14 respectively.

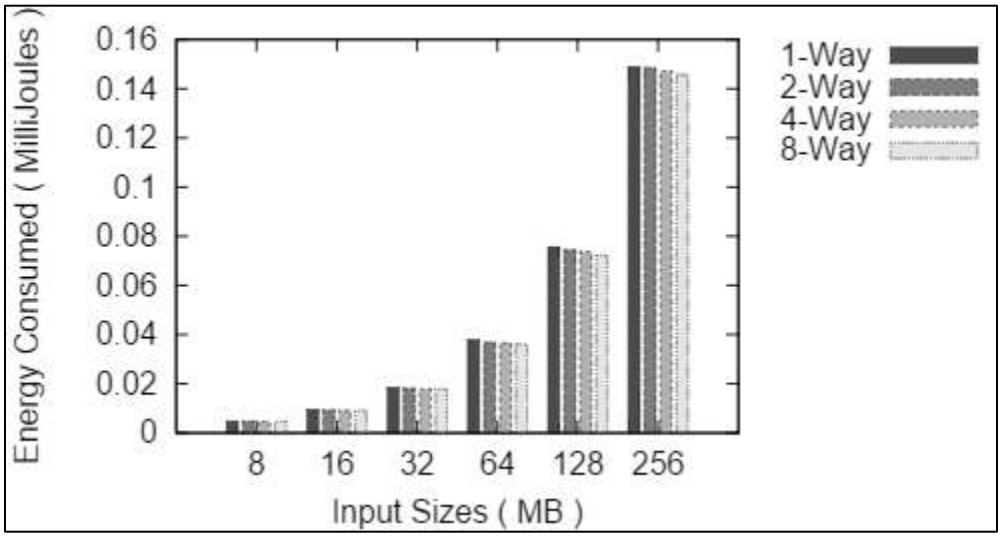

Figure 13: Energy Consumption of Redesigned SHA-1 with P-way Parallelism

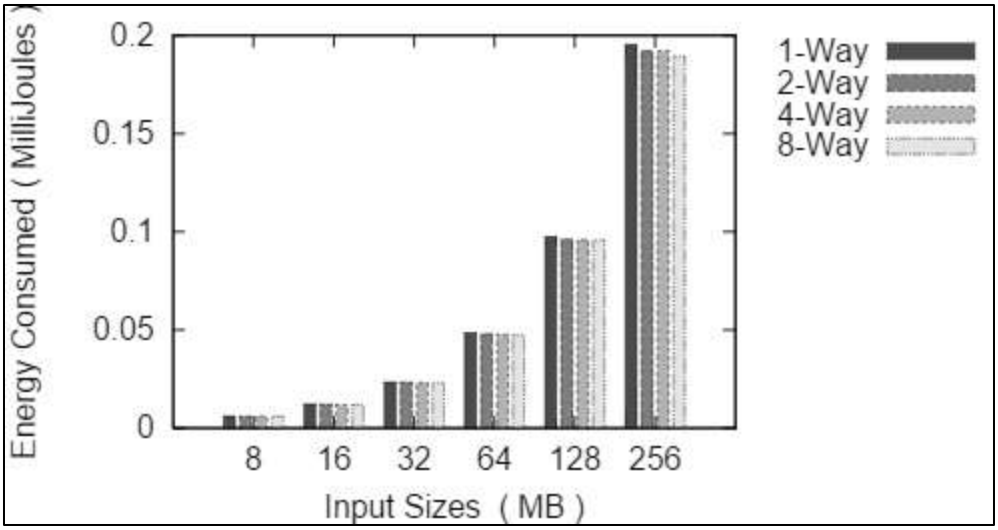

Figure 14: Energy Consumption of Redesigned SHA-2 with P-way Parallelism

From the results, 8-way parallelism using eight memory banks was found to consume the lesser energy than 1-way parallelism, as shown in Table 2. The reason for this is that 8-way parallelism provides complete parallel data access from the available memory banks and reduces the I/O requests while 1-way parallelism requires a higher number of I/O requests, taking more time and consuming more energy compared to 8-way parallelism. According to the energy complexity model considered for the experiment, 8-way parallelism should consume less energy when compared to 1, 2 and 4-way data access patterns in the hash functions. Thus, the experimental observations and theoretical expectations are in agreement.

Algorithm (Redesigned, with 8-way parallelism)	Energy Consumed in 1-way parallelism (MicroJoules)	Energy Consumed in 8-way parallelism (MicroJoules)	Percentage Difference in Energy consumption (1-way vs. 8-way)
MD2	288.55	283.96	1.55%
MD5	908.81	866.3	5.11%
SHA1	624.21	602.75	3.56%
SHA2	164.41	160.78	2.26%

Table 2: Improvement in Energy Consumption with 8-way Parallelism

With increase in the input size from 128 MB to 256 MB, the energy consumption was further reduced, as shown in Table 3. The energy saved increases when the size of the input is relatively large (128MB and 256MB) rather than relatively small (8MB, 32MB, 64MB) because the smaller data will be stored in the processor cache (6 MB L3 and 256 KB L2 in the test setup), thereby nullifying the effect of memory parallelism in the DDR3 architecture (RAM) on energy consumption.

Algorithm (Redesigned, with 8-way parallelism)	Percentage reduction in Energy Consumption for 8 MB input size	Percentage reduction in Energy Consumption for 256 MB input size
MD2	1.23%	1.55%
MD5	3.39%	5.11%
SHA1	3.27%	3.56%
SHA2	1.86%	2.26%

Table 3: Comparison of Energy Comsumption Decrease with Increase in Input Size for 8-way Parallelism

6.4 Comparison of Energy Consumption by Hash Functions

Additional tests and analysis was performed for this research to determine the recommended hash function for applications. The tests for this experiment were run on the redesigned versions of the hash functions with 8-way parallelism. The tests were conducted 50 times on each of the redesigned hash functions and the mean value was recorded. The result obtained is shown in Figure 15. MD2 consumed more energy when compared to MD5, SHA-1, and SHA-2. Since MD2 has been proved to be compromised, usage of MD2 is not recommended for applications run on battery-driven handheld devices.

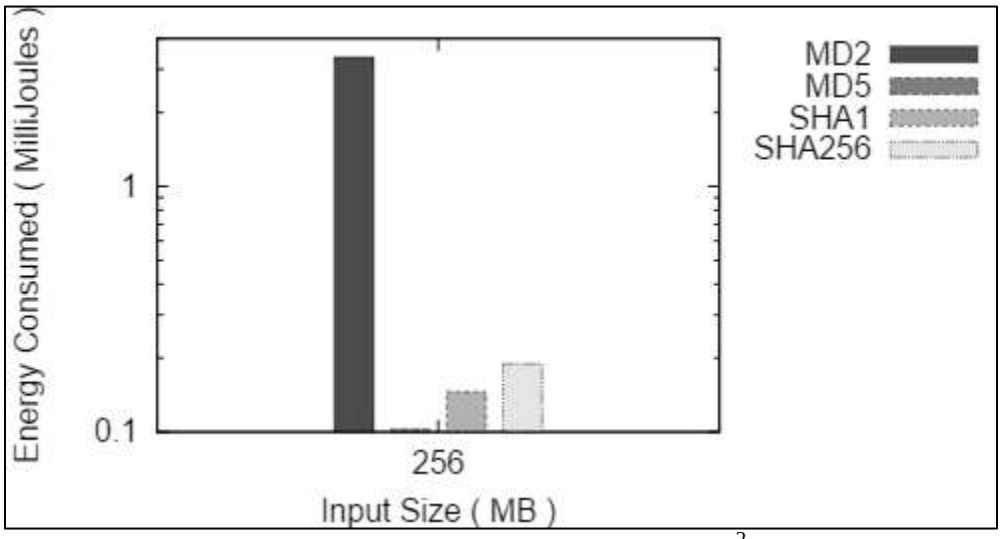

Figure 15: Energy Consumed by Redesigned Hash Functions[2] with 8-way Parallelism

On the other hand, MD5 consumed the less energy amongst the tested hash functions. Although it is compromised [Dobbertin96], MD5 finds use in applications due to its

[2] In Figure 15, SHA1 is same as SHA-1 and SHA256 is SHA-2.

execution speed. However, advanced cryptographic hash functions SHA-1 and SHA-2 have been designed with additional complexity for better security and are recommended for use over MD5. Higher energy consumed by SHA-1 and SHA-2 can be attributed to additional iterations to further convolute output which were added to increase function complexity.

Considering the mean value of energy consumed by each hash function for 4-way parallelism, the result still showed that MD2 consumed more energy while MD5 consumed lesser energy compared to other hash functions, as shown in Figure 16.

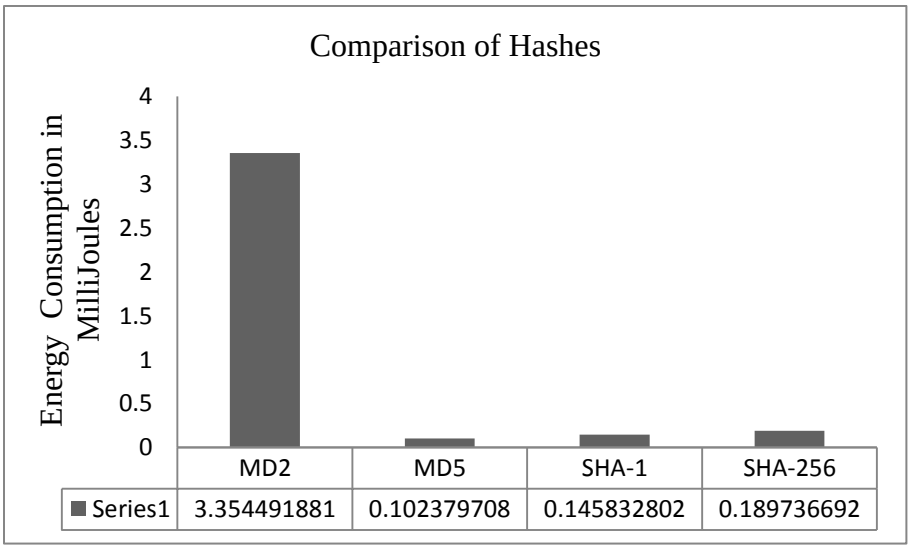

Figure 16: Energy Consumed by Redesigned Hash Functions with 4-way parallelism

6.5 Statistical Analysis

Redesigned MD5 and SHA-2 were selected for further statistical analysis. While MD5 algorithm consumes the least energy amongst tested algorithms, SHA-2 is used for data

integrity in the latest applications at the time of this research, as it is not vulnerable to pre-image or collision attack [stackExchange12]. For the statistical analysis, energy in MilliJoules is measured for 50 runs of the redesigned MD5 and SHA-2 algorithms with 8-way parallelism. Each test runs the algorithm with different inputs of the same size to avoid caching of results in order to make results independent for statistical analysis.

The *t*-test [Sawilowsky92] was initially considered for statistical analysis, which requires two samples of data having uniform distribution. For example, the *t*-test can be used to analyze students' performance in tests between a group that received training and a group that did not receive training as it can be assumed that the results are independent and the two samples have equal variance when the data is normalized. However, for the data samples collected from MD5 and SHA-2 tests, it is not possible to make an assumption that the samples are uniformly distributed. Thus, the Mann-Whitney U-Test non-parametric test [McKnight10] was selected for the statistical analysis.

6.5.1 Mann-Whitney U Test

In Table 4, it can be observed that the variance of range (maximum – minimum) is lower than the mean value. Thus, the standard deviation shows higher spread as it is nearly half the value of the variance. This shows that the data is not uniformly distributed and it cannot be used to perform a *t*-test.

Descriptive Statistics						
Energy Consumed	Number of experiment runs	Mean	Std. Deviation	Minimum	Maximum	Variance
	100	4963251.27	1496198.189	3445280	6520338	3075058

Table 4: Analysis of Variance of Energy Consumption of Redesigned MD5 and SHA-2

The statistical analysis tool used for this experiment was SPSS software [Morgan05]. SPSS was used to generate the rank table shown in Table 5. Higher mean value in this table indicates higher energy consumption. From the results, it is evident that SHA-2 consumes higher energy than MD5 and the difference is statistically significant.

Ranks				
	Group	N	Mean Rank	Sum of Ranks
Energy Consumed	MD5	50	25.50	1275.00
	SHA-2	50	75.50	3775.00
	Total	100		

Table 5: Mann-Whitney Test Ranks for Redesigned Hash Functions, 8-way Parallelism

Mann-Whitney U-Test was performed to compare the energy consumption in MD5 and SHA-2 because variance was unequal and the dependent variables were ordinal. The total number of samples, *N*, was 100. As shown in Table 6, U is .000, *r* is -0.8(Z/sqrt (N)) and Asymp. Sig. (2-tailed), also called *p* value, is .000, indicating statistically

significant differences between MD5 and SHA-2 energy consumption. These results of the U-Test indicate that energy consumed by MD5 was significantly lower than SHA-2.

Test Statistics[3]	
	Energy Consumed
Mann-Whitney U	.000
Wilcoxon W	1275.000
Z	-8.617
Asymp. Sig. (2-tailed)	.000

Table 6: Grouping Variables by Hash Function MD5 and SHA-2

6.5.2 Recommendations on Hash Usage

From the above experiments and results, we can observe that MD5 is the most efficient in terms of energy consumption. However, MD5 has been found to be vulnerable to preimage and collision attacks, whereas no vulnerabilities in SHA-2 are currently known (See Table 1). To analyze if MD5 can be used to reduce energy consumption without compromising security of an application, the information presented in Table 7 may be useful.

[3] Grouping Variable: Group

Hash	Vulnerabilities (Collision Complexity)	Time Required at 10^9 comparisons/s	Time Required at 10^{13} comparisons/s
MD5	2^{64}	$2^{64} \approx 584.5$ years	6 months
SHA-1	2^{80}	$2^{80} \approx 3.83 * 10^7$ years	38000 years
SHA-2	None	_	_

Table 7: Time Required for Collision Attack [Harish14]

From the time required by attackers to perform collision attacks shown in Table 5, it is evident that, an adversary would be unlikely for a supercomputer to perform a collision attack on an application in reasonable time. Thus, if the user is mostly sure of the strength of the adversary, MD5 may be chosen over SHA-2 to conserve energy.

Chapter 7

CONCLUSION

In this research, the energy consumed by an anti-virus application that uses hash functions was measured. The results obtained show that the instantaneous power and total energy consumed by the anti-virus software is high due to the use of computationally intensive cryptographic hash algorithms. For this reason, hash functions were chosen to test the applicability of the energy complexity model. The hash function MD2 was considered first to understand the data access pattern and test the applicability of the energy complexity model. On incorporating the energy complexity model, the amount of energy consumed by MD2 was reduced by 1.55% with 8-way parallelism. Thus, it was found that the energy complexity model is applicable to hash functions.

The energy complexity model works based on the specified level of parallelism for DDR3 based memory. The memory layout of the complexity model is based on DDR3 architecture which has eight memory banks from which data can be accessed in parallel. A page table is used to create logical mapping of data blocks from given input to physical location in the memory banks so that blocks can be accessed from the memory based on specified *P*-way (1, 2, 4 and 8). As the level of parallelism is increased from 1-way to 8-way (full parallelism), the results show decrease in the energy consumption, by 5.11% in MD5, 3.56% in SHA1, and 2.26% in SHA-2.

The experiment is extended by further incorporation of the energy complexity model on MD5, SHA-1 and SHA-2. Input of sizes 8MB, 16MB, 32MB, 64MB, 128MB, and 256MB are considered for the experiment and results are recorded by varying the level of parallelism from 1, 2, and 4 to 8- way for each size. The results show a decrease in energy consumption when tests change parallelism from 1-way to 8-way. Also, the improvement is significant for larger data of sizes 128MB and 256MB as the cache negates the effects of parallelism on smaller inputs. MD2 showed 1.23% decrease for 8MB and 1.55% decrease for 256MB (Table 3).

Some of the applications like video conferencing in telemedicine, control systems, group communication, digital signature and data integrity use hash functions for the secure transfer of data across a communication network. By incorporating the energy complexity model in these cryptographic hash functions, it is expected that the energy consumed by the applications will be reduced.

Additionally, a comparison of energy consumption among hash functions showed that MD2 is the most expensive in terms of energy consumption. On the other hand, MD5 consumes the least energy compared to SHA-1 and SHA-2. However, SHA-1 and SHA-2 have additional processing steps for better security, and an adversary would be unlikely to have a supercomputer to perform a collision attack on application in reasonable time. Thus, if the user is mostly sure of the strength of the adversary, MD5 may be chosen over SHA-2 to conserve energy.

Chapter 8

FUTURE WORK

The current research focused on reducing energy consumption based on the energy complexity model designed by Roy et al. [Roy13]. With green computing and energy conservation becoming important indicators of performance of applications [Zhu04], current research can be further extended to other possible approached to improve the energy consumption of cryptographic hash algorithms. As Figure 17 shows, the energy consumed by the existing SHA-2 algorithm with no parallelism is not much worse than the energy consumed by the algorithm in which energy complexity model is incorporated with 8-way parallelism. Thus, there is still room for improvement with other techniques to reduce energy consumption. Also, this work can be extended by writing scalable programs, by developing multi-threaded applications with multiprocessor servers.

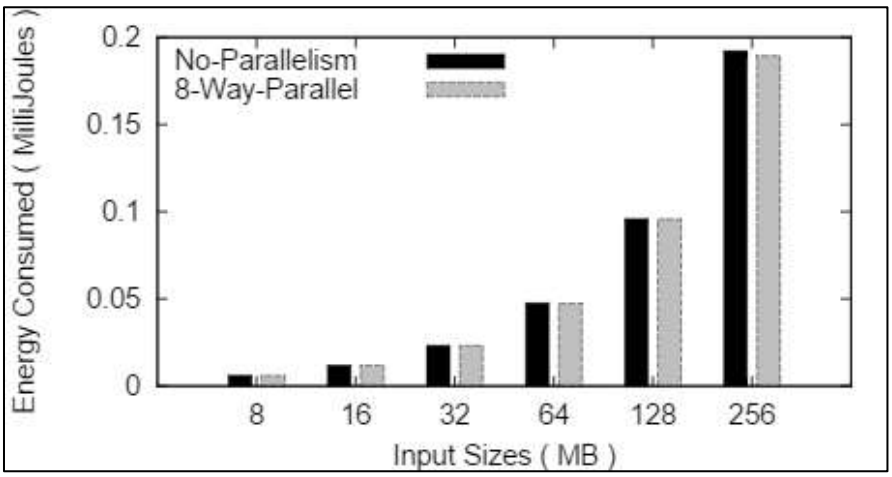

Figure 17: SHA-2 Energy Consumption Comparison between No Parallelism vs. 8-way Parallelism

This research conducted experimentation on an Apple© Macbook® laptop equipped with an Intel i5 processor and running Mac OS X Lion 10.7.3. Thus, the experiment can be extended to other operating systems, like Windows, Linux and operating systems developed for mobile devices, to compare the energy optimization across various processor and OS architectures.

The intent of this research was to study data access patterns in cryptographic hash functions and test the applicability of the generic energy complexity model to analyze variation in energy consumption by the cryptographic protocols. Based on the results obtained in this research, it is evident that researchers can make use of the complexity model to redesign more advanced hash algorithms like HMAC and SHA-3/Keccak.

REFERENCES

Print Publications:

[Amamra12] Amamra, A., Talhi, C. and Robert, J. "Smartphone malware detection: From a survey towards taxonomy," in Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on , vol., no., pp.79-86, 16-18 October 2012

[Aycock06] Aycock, J., deGraaf, R. and Jacobson, M., Jr. "Anti-disassembly using Cryptographic Hash Functions". Journal in Computer Virology, vol. 2, no. 1 pp. 79-85, August 2006

[Bianchini04] Bianchini, R. and Rajamony, R. "Power and Energy Management for Server Systems", Computer, vol.37, no. 11, pp. 68-74, November 2004.

[Carlson14] Carlson, F. R. "Security Analysis of Cloud Computing", eprint arXiv:1404.6849, April 2014

[Cooley10] Cooley, J., Khazan, R., Fuller, B. W. and Pickard, G. E. GROK "GROK: A Practical System for Securing Group Communications," in Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on , vol., no., pp.100-107, 15-17 July 2010

[Damasevicius12] Damasevicius, R., Ziberkas, G., Stuikys, V. and Toldinas, J. "Energy Consumption of Hash Functions."Elektronika ir Elektrotechnika, vol. 18, no. 10, pp. 81-84, 2012.

[Damgård90] Damgård, I. "A design principle for hash functions." In Advances in Cryptology—CRYPTO'89 Proceedings, pp. 416-427. Springer New York, 1990.

[Dobbertin96] Dobbertin, H. "The status of MD5 after a recent attack." CryptoBytes 2, no. 2 1996.

[Eastlake01] Eastlake 3rd, D. and Jones, P. "US secure hash algorithm 1 (SHA1)". No. RFC 3174. 2001.

[Edge07] Edge, K., Raines, R., Grimaila, M., Baldwin, R., Bennington, R. and Reuter, C. "The use of attack and protection trees to analyze security for an online banking system." In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, pp. 144b-144b. IEEE, 2007.

[Erdogan07] Erdogan, O. and Cao, P. "Hash-AV: fast virus signature scanning by cache-resident filters." International Journal of Security and Networks 2, no. 1-2, pp. 50-59, 2007

[Goldwasser99] Goldwasser, S. and Bellare, "Lecture notes on cryptography."Summer course “Cryptography and computer security” at MIT 1999, pp.1996, 1999.

[Goraczko11] Goraczko, M., Kansal, A., Liu, J. and Zhao, F. "Joulemeter: Computational energy measurement and optimization." 2011.

[Harish14] Harish, P. D. and Roy, S. "Towards Designing Greener Secured Hash Functions," in Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom), IEEE , vol., no., pp.618-621, 1-3 Sept. 2014

[Hodges01] Hodges, V. and O'donnell, S. "Method and system for providing automated updating and upgrading of antivirus applications using a computer network." U.S. Patent 6,269,456, issued July 31, 2001

[Hongmei02] Hongmei, D., Li, W. and Agrawal, D. P. "Routing security in wireless ad hoc networks," in Communications Magazine, IEEE , vol.40, no.10, pp.70-75, Oct 2002

[Kaliski92] Kaliski, B. The MD2 message-digest algorithm, 1992.

[Karras04] Karras, D. A. "A combined genetic optimization and multilayer perceptron methodology for efficient digital fingerprint modeling and evaluation in secure communications," in Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on , vol.3, no., pp.2319-2324 vol.3, 25-29 July 2004

[Knudsen10] Knudsen, L., Mathiassen, J., Muller, F. and Thomsen, S. "Cryptanalysis of MD2." Journal of cryptology 23, no. 1, pp. 72-90, 2010.

[Konheim10] Konheim, A. G. "Hashing in Computer Science: Fifty Years of Slicing and Dicing." John Wiley & Sons, 2010.

[Kouznetsova08] Kouznetsova, S. "A networking lab facility on the cheap: turning obstacles into opportunities." Journal of Computing Sciences in Colleges 23, no. 4, pp.180-185, 2008.

[Mahadevan09] Mahadevan, P., Sharma, P., Banerjee, S. and Ranganathan, P. "Energy Aware Network Operations," in INFOCOM Workshops 2009, IEEE , vol., no., pp.1-6, 19-25 April 2009

[Massengale08] Massengale, Rick L. "On demand dynamic educational lab configuration and use." Journal of Computing Sciences in Colleges 23, no. 4, pp. 66-72, 2008.

[Martin04] Martin, T., Hsiao, M., Ha, D. and Krishnaswami, J. "Denial-of-service attacks on battery-powered mobile computers," in Pervasive Computing and Communications,

2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on , vol., no., pp.309-318, 14-17 March 2004.

[McKnight10] McKnight, P. E. and Najab, J. "Mann-Whitney U Test." Corsini Encyclopedia of Psychology, 2010.

[Mishra15] Mishra, M, and Bhatele, M. Improved Cloud Security Approach with Threshold Cryptography. ISSN (ONLINE) . International journal of Master of Engineering Research and Technology, vol 2, 2015.

[Morgan12] Morgan, G. A., Leech, N. L., Gloeckner, G. W. and Barrett, K. C. "IBM SPSS for introductory statistics: Use and interpretation." Routledge, 2012.

[Neto11] Neto, P. "Demystifying cloud computing." In Proceeding of Doctoral Symposium on Informatics Engineering. 2011

[Paya13] Paya, A. and Marinescu, D. C. "Energy-aware Application Scaling on a Cloud." arXiv preprint arXiv:1307.3306, 2013.

[Potlapally06] Potlapally, N. R., Ravi, S., Raghunathan, A. and Jha, N. K. "A study of the energy consumption characteristics of cryptographic algorithms and security protocols," in Mobile Computing, IEEE Transactions on , vol.5, no.2, pp.128-143, Feb. 2006.

[Preneel94] Preneel, B. "Cryptographic hash functions." European Transactions on Telecommunications 5, no. 4, pp.431-448, 1994.

[Ravilla15] Ravilla, D. and Putta, C. S. R. "Implementation of HMAC-SHA256 algorithm for hybrid routing protocols in MANETs," in Electronic Design, Computer Networks & Automated Verification (EDCAV), 2015 International Conference on , vol., no., pp.154-159, 29-30 Jan. 2015

[Rivest92A] Rivest, R. The MD5 message-digest algorithm 1992.

[Rivest92B] Rivest, R. The MD4 message-digest algorithm 1992.

[Roy13] Roy, S., Rudra, A. and Verma, A. "An energy complexity model for algorithms." In Proceedings of the 4th conference on Innovations in Theoretical Computer Science, pp. 283-304. ACM, 2013

[Rogier97] Rogier, N., and P. Chauvaud. "MD2 is not secure without the checksum byte." Designs, Codes and Cryptography 12, no. 3, pp: 245-251, 1997.

[Schneier05] Schneier, B. "Schneier on Security: Cryptanalysis of SHA-1." February 18, 2005.

[Sawilowsky92] Sawilowsky, S. S. and Blair, R. C. "A more realistic look at the robustness and type II error properties of the t test to departures from population normality." Psychological bulletin 111, no. 2, pp. 352, 1992.

[Sklavos03] Sklavos, N. and Koufopavlou, O. "On the hardware implementations of the SHA-2 (256, 384, 512) hash functions," in Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on , vol.5, no., pp.V-153-V-156 vol.5, 25-28 May 2003

[Sobol94] Sobol, M. I. "Anti-Virus Software." Information Systems Security 3, no. 2, pp. 24-29, 1994.

[Stallings95] Stallings, W. "Network and internetwork security: principles and practice." vol. 1. Englewood Cliffs: Prentice Hall, New Jarsey, January 1995.

[Stevens12] Stevens, M., Lenstra, A. K. and De Weger, B. "Chosen-prefix collisions for MD5 and applications." International Journal of Applied Cryptography 2, no. 4, pp. 322-359, 2012.

[Tiwari94] Tiwari, V., Malik, S. and Wolfe, A. "Compilation techniques for low energy: an overview," in Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium , vol., no., pp.38-39, 10-12 Oct. 1994.

[Tulu03] Tulu, B., Chatterjee, S., Abhichandani, T. and Haiqing, L. "Secured video conferencing desktop client for telemedicine," in Enterprise Networking and Computing in Healthcare Industry, 2003. Healthcom 2003. Proceedings. 5th International Workshop on , vol., no., pp.61-65, 6-7 June 2003.

[Waldin00] Waldin, R. and Nachenberg, C. "Antivirus accelerator for computer networks." U.S. Patent 6,094,731, issued July 25, 2000.

[Wang05] Wang, X., Yao, A. C. and Yao, F. "Cryptanalysis on SHA-1." In Cryptographic Hash Workshop hosted by NIST. 2005.

[Wang13] Wang, Z. and Cao, L. "Implementation and comparison of two hash algorithms." In 2013 International Conference on Computational and Information Sciences, pp. 721-725. IEEE, 2013.

[Xun13] Xun, L. and Chong, F. T. "A case for energy-aware security mechanisms." In Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on, pp. 1541-1546. IEEE, 2013.

[Yan12] Yan, S., Tianran, W., Aidong, X., Kai, W. and Zhijia, Y. "Communication security problem and solution in safety-related system." InSystems and Informatics (ICSAI), 2012 International Conference on, pp. 524-528. IEEE, 2012.

[Yensen05] Yensen, J. "Leveraging RSS feeds to support current awareness."Computers Informatics Nursing 23, no. 3, pp.164-167, 2005.

[Zhu04] Zhu, Q., David, F. M., Devaraj, C. F., Li, Z., Zhou, Y. and Cao, P. "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," in Software, IEE Proceedings- , vol., no., pp.118-118, 14-18 Feb. 2004

Electronic Sources:

[Klinge15] Klinge, Evan, and Starkweather David. "pHash: The open source perceptual hash library." http://www.phash.org/ , 2015, last accessed 2015.

[Perrin07] Perrin, Chad. "Use Md5 Hashes to Verify Software Download." http://www.techrepublic.com/blog/it-security/use-md5-hashes-to-verify-software-downloads/, 2007, last accessed 2015.

[Software15] Software-Systeme. "M.B. Hardware Monitor." http://www.bresink.com/osx/HardwareMonitor.html, 2015, last accessed 2015.

[stackExchange12] stackExchange. "Sha-256 Vs Any 256 Bits of Sha-512, Which Is More Secure?" http://crypto.stackexchange.com/questions/3153/sha-256-vs-any-256-bits-of-sha-512-which-is-more-secure, 2012, last accessed 2015.

APPENDIX A

A.1 Sample Test Run for MD2

```
$ ./a.out 32768 0 1 16 1 8 256 ARG0: ./a.out  N: 32768   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256 e0753ba48ea3c4eb0a9adb2f1a37c64b Execution
time: 17807 nsec
$ ./a.out 32768 0 1 16 2 8 256 ARG0: ./a.out  N: 32768   mode: 0,  iterations 1
realStride 16 way 2  NUMBANKS 8 blockSize 256 e0753ba48ea3c4eb0a9adb2f1a37c64b Execution
time: 17848 nsec
$ ./a.out 32768 0 1 16 4 8 256 ARG0: ./a.out  N: 32768   mode: 0,  iterations 1
realStride 16 way 4  NUMBANKS 8 blockSize 256 e0753ba48ea3c4eb0a9adb2f1a37c64b Execution
time: 18292 nsec
$ ./a.out 32768 0 1 16 8 8 256 ARG0: ./a.out  N: 32768   mode: 0,  iterations 1
realStride 16 way 8  NUMBANKS 8 blockSize 256 e0753ba48ea3c4eb0a9adb2f1a37c64b Execution
time: 18319 nsec $
$ ./a.out 65536 0 1 16 1 8 256 ARG0: ./a.out  N: 65536   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256 f10370836e97170b76c8f6e8000db8be Execution
time: 31956 nsec
$ ./a.out 65536 0 1 16 2 8 256 ARG0: ./a.out  N: 65536   mode: 0,  iterations 1
realStride 16 way 2  NUMBANKS 8 blockSize 256 f10370836e97170b76c8f6e8000db8be Execution
time: 32673 nsec
$ ./a.out 65536 0 1 16 4 8 256 ARG0: ./a.out  N: 65536   mode: 0,  iterations 1
realStride 16 way 4  NUMBANKS 8 blockSize 256 f10370836e97170b76c8f6e8000db8be Execution
time: 33155 nsec
$ ./a.out 65536 0 1 16 8 8 256 ARG0: ./a.out  N: 65536   mode: 0,  iterations 1
realStride 16 way 8  NUMBANKS 8 blockSize 256 f10370836e97170b76c8f6e8000db8be Execution
time: 34847 nsec
```

A.2 Sample Test Runs of MD5

```
$ ./a.out 8192 0 1 16 1 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 1  NUMBANKS 8 blockSize 256 5433359A 66CCBEC7 DF9BA0BC F98F8374 474E414C 585F5F00
5F4350 20435058  9a353354c7becc66bca09bdf74838ff9 Execution time: 221 nsec  $
$ ./a.out 8192 0 1 16 2 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 2  NUMBANKS 8 blockSize 256 5433359A 66CCBEC7 DF9BA0BC F98F8374 474E414C 585F5F00
5F4350 20435058  9a353354c7becc66bca09bdf74838ff9 Execution time: 218 nsec
$ ./a.out 8192 0 1 16 4 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 4  NUMBANKS 8 blockSize 256 5433359A 66CCBEC7 DF9BA0BC F98F8374 474E414C 585F5F00
5F4350 20435058  9a353354c7becc66bca09bdf74838ff9 Execution time: 220 nsec
$ ./a.out 8192 0 1 16 8 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 8  NUMBANKS 8 blockSize 256 5433359A 66CCBEC7 DF9BA0BC F98F8374 474E414C 585F5F00
5F4350 20435058  9a353354c7becc66bca09bdf74838ff9 Execution time: 264 nsec
$ ./a.out 16384 0 1 16 1 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256 9395E346 393F006C CBE3D040 268C64A6
474E414C 585F5F00 5F4350 20435058  46e395936c03f3940d0e3cba6648c26 Execution time: 439
nsec
$ ./a.out 16384 0 1 16 1 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256
9395E346 393F006C CBE3D040 268C64A6 474E414C 585F5F00 5F4350 20435058
46e395936c03f3940d0e3cba6648c26 Execution time: 440 nsec
```

```
$ ./a.out 16384 0 1 16 2 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 2  NUMBANKS 8 blockSize 256 9395E346 393F006C CBE3D040 268C64A6
474E414C 585F5F00 5F4350 20435058  46e395936c03f3940d0e3cba6648c26 Execution time: 443
nsec
$ ./a.out 16384 0 1 16 4 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 4  NUMBANKS 8 blockSize 256 9395E346 393F006C CBE3D040 268C64A6
474E414C 585F5F00 5F4350 20435058  46e395936c03f3940d0e3cba6648c26 Execution time: 446
nsec
```

A.3 Sample Test Runs for SHA-1

```
$ ./a.out 8192 0 1 16 1 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 1  NUMBANKS 8 blockSize 256 C344215A A786078D 77869B38 1F028352 71EDC018 5A827999
6ED9EBA1 8F1BBCDC  c344215aa78678d77869b381f2835271edc018 Execution time: 309 nsec SHA1
tests:  DONE
$ ./a.out 8192 0 1 16 2 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 2  NUMBANKS 8 blockSize 256 C344215A A786078D 77869B38 1F028352 71EDC018 5A827999
6ED9EBA1 8F1BBCDC  c344215aa78678d77869b381f2835271edc018 Execution time: 311 nsec SHA1
tests:  DONE
$ ./a.out 8192 0 1 16 4 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 4  NUMBANKS 8 blockSize 256 C344215A A786078D 77869B38 1F028352 71EDC018 5A827999
6ED9EBA1 8F1BBCDC  c344215aa78678d77869b381f2835271edc018 Execution time: 311 nsec SHA1
tests:  DONE
 $ ./a.out 8192 0 1 16 8 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1
realStride 16 way 8  NUMBANKS 8 blockSize 256 C344215A A786078D 77869B38 1F028352
71EDC018 5A827999 6ED9EBA1 8F1BBCDC  c344215aa78678d77869b381f2835271edc018 Execution
time: 406 nsec SHA1 tests:  DONE
$ ./a.out 16384 0 1 16 1 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256 D5776504 EDAFE273 6128A53D FABD84F6
5F9FDEAB 5A827999 6ED9EBA1 8F1BBCDC  d577654edafe2736128a53dfabd84f65f9fdeab Execution
time: 615 nsec SHA1 tests:  DONE
$ ./a.out 16384 0 1 16 2 8 256 ARG0: ./a.out
 N: 16384   mode: 0,  iterations 1  realStride 16 way 2  NUMBANKS 8 blockSize 256
D5776504 EDAFE273 6128A53D FABD84F6 5F9FDEAB 5A827999 6ED9EBA1 8F1BBCDC
d577654edafe2736128a53dfabd84f65f9fdeab Execution time: 619 nsec SHA1 tests:  DONE    $
./a.out 16384 0 1 16 4 8 256  ARG0: ./a.out  N: 16384   mode: 0,  iterations 1  realStride
16 way 4  NUMBANKS 8 blockSize 256 D5776504 EDAFE273 6128A53D FABD84F6 5F9FDEAB 5A827999
6ED9EBA1 8F1BBCDC  d577654edafe2736128a53dfabd84f65f9fdeab Execution time: 620 nsec SHA1
tests:  DONE
```

A.4 Sample Test Runs for SHA-256

```
$ ./a.out 8192 0 1 16 1 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 1  NUMBANKS 8 blockSize 256 D455EB5A 6D4865CC 61FE154 A9B90BF6 C938C4E4 A5755593
D7670CC8 F7382B3D  d455eb5a6d4865cc61fe154a9b9bf6c938c4e4a5755593d767cc8f7382b3dExecution
time: 403 nsec
$ ./a.out 8192 0 1 16 2 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 2  NUMBANKS 8 blockSize 256 D455EB5A 6D4865CC 61FE154 A9B90BF6 C938C4E4 A5755593
D7670CC8 F7382B3D  d455eb5a6d4865cc61fe154a9b9bf6c938c4e4a5755593d767cc8f7382b3dExecution
time: 404 nsec
$ ./a.out 8192 0 1 16 4 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 4  NUMBANKS 8 blockSize 256 D455EB5A 6D4865CC 61FE154 A9B90BF6 C938C4E4 A5755593
D7670CC8 F7382B3D  d455eb5a6d4865cc61fe154a9b9bf6c938c4e4a5755593d767cc8f7382b3dExecution
time: 405 nsec
$ ./a.out 8192 0 1 16 8 8 256 ARG0: ./a.out  N: 8192   mode: 0,  iterations 1  realStride
16 way 8  NUMBANKS 8 blockSize 256 D455EB5A 6D4865CC 61FE154 A9B90BF6 C938C4E4 A5755593
```

```
D7670CC8 F7382B3D  d455eb5a6d4865cc61fe154a9b9bf6c938c4e4a5755593d767cc8f7382b3dExecution
time: 407 nsec
$ ./a.out 16384 0 1 16 1 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 1  NUMBANKS 8 blockSize 256 4F789305 FBAA59D0 B14DF108 FD94532C
6BE30265 AA634035 38ADDB9C 8A88A742
4f78935fbaa59d0b14df18fd94532c6be3265aa63403538addb9c8a88a742Execution time: 804 nsec
$ ./a.out 16384 0 1 16 2 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 2  NUMBANKS 8 blockSize 256
4F789305 FBAA59D0 B14DF108 FD94532C 6BE30265 AA634035 38ADDB9C 8A88A742
4f78935fbaa59d0b14df18fd94532c6be3265aa63403538addb9c8a88a742Execution time: 805 nsec
$ ./a.out 16384 0 1 16 4 8 256 ARG0: ./a.out  N: 16384   mode: 0,  iterations 1
realStride 16 way 4  NUMBANKS 8 blockSize 256 4F789305 FBAA59D0 B14DF108 FD94532C
6BE30265 AA634035 38ADDB9C 8A88A742
4f78935fbaa59d0b14df18fd94532c6be3265aa63403538addb9c8a88a742Execution time: 808 nsec
```

VITA

Priyanka Dhoopa Harish is currently working as a Software Development Engineer at Deutsche Bank, North Carolina. She expects to receive her Master of Science in Computing and Information Sciences degree in the fall 2015. Priyanka was working as a graduate research assistant under the guidance of Dr. Swapnoneel Roy at UNF from January 2014 - August 2015 and also worked as graduate teaching assistant during spring 2015. The research work performed by her has resulted in the conference publications in the field of Computer Security and Energy Aware Computing. She has participated in IEEE conferences like ACM SIGCOMM-2014 held at Chicago, GREENCOM-2014 held at Taipei and also she was awarded with a travel grant to participate in the workshop, N2WOMEN at ACM SIGCOMM. Priyanka has worked for 4 years as a Software Engineer at IBM Software Labs, where she was responsible for developing the testing framework for database application tools using technologies such as Java J2EE, Perl, C and IBM DB2.

Publications from this Thesis

1. Harish, Priyanka D., and Roy, Swapnoneel (2014). Energy Oriented Vulnerability Analysis on Authentication Protocols for CPS, *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS).*

2. Harish, Priyanka D., and Roy, Swapnoneel (2014). Towards Designing Greener Secured Hash Functions. *IEEE International Conference on Green Computing and Communications (GreenCom).*

3. Harish, Priyanka D., Talluri, Raghu S., and Roy, Swapnoneel. Towards Designing Energy Optimal Secured Hashes and Symmetric Key Encryption Protocols. *Communicated to Computing Journal,* Springer.