Can We Improve Energy Efficiency of Secure Disk Systems without Modifying Security Mechanisms?

Xiaojun Ruan[†], Adam Manzanares[†], Shu Yin[†], Mais Nijim[‡], and Xiao Qin[†*]

[†] Department of Computer Science and Software Engineering
Auburn University, Auburn AL, 36849-5347
{ xzr0001, acm0008, szy0004, xqin}@auburn.edu

[‡] School of Computing
University of Southern Mississippi
Hattiesburg, MS 39406
mais.nijim@usm.edu

***Abstract*—Improving energy efficiency of security-aware storage systems is challenging, because security and energy efficiency are often two conflicting goals. The first step toward making the best tradeoffs between high security and energy efficiency is to profile encryption algorithms to decide if storage systems would be able to produce energy savings for security mechanisms. We are focused on encryption algorithms rather than other types of security services, because encryption algorithms are usually computation-intensive. In this study, we used the XySSL libraries and profiled operations of several test problems using Conky - a lightweight system monitor that is highly configurable. Using our profiling techniques we concluded that although 3DES is much slower than AES encryption, it more likely to save energy in security-aware storage systems using 3DES than AES. The CPU is the bottleneck in 3DES, allowing us to take advantage of dynamic power management schemes to conserve energy at the disk level. After profiling several hash functions, we noticed that the CPU is not the bottleneck for any of these functions, indicating that it is difficult to leverage the dynamic power management technique to conserve energy of a single disk where hash functions are implemented for integrity checking.**

I. INTRODUCTION

In the past decade, energy efficiency has become an ever increasing priority in computer science research [2]. Computers have traditionally been designed with performance metrics being the main focus of the design. Since our sources of energy to power computers are not limitless, it is imperative to design energy efficient computer architectures. Disk systems tend to be large consumers of energy consumption [4] and; therefore, we have designed an energy efficient parallel disk framework (see [3] for details of our disk framework). Apart from high energy efficiency, security mechanisms are equally important for disk systems to support a wide range of data-intensive applications that are security sensitive. Although previous research has focused on producing a relationship between energy efficiency and security strength for mobile devices (see, for example, [1]), it is challenging to make good tradeoffs between high security and energy efficiency for storage systems in general and for disk systems in particular.

The long-term goal of this research is to develop energy-efficient security mechanisms for disk systems without significantly degrading disk performance. Similar design goals can be found in the literature (see, for example, [4] and [5]). This study started off with having a goal of developing a matrix that would outline the tradeoffs between energy efficiency and security in the context of large-scale disk systems.

Generally speaking, there are two approaches to implementing energy-efficient security-aware disk systems. The first one is to improve the energy efficiency of security mechanisms in disk systems (see, for example, [1]); the second approach advocates for integrating conventional security services with energy-efficient disk architectures. The first approach makes an effort to implement energy-efficient security mechanisms in traditional disk systems, whereas the second one is focused on energy-efficient disk systems without modifying existing security mechanisms. In this study, we focus on the second general approach. Thus, we attempt to answer an intriguing question of whether it is possible to seamlessly integrate security services with energy-efficient disk systems without modifying the source code of security services.

To determine if it is possible to conserve energy consumption of existing security services using new energy-efficient disk systems, we will have to investigate I/O access patterns of encryption and integrity checking algorithms. Studying I/O access patterns of disk requests issued by the security services requires knowledge of encryption algorithms and hash functions. In this research, we first investigate the I/O characteristics of encryption algorithms and hash functions. Next, we apply these I/O characteristics within the energy-efficient buffer disk architecture or BUD (see [9] for detailed information concerning BUD) to investigate the possibility of leveraging BUD to reduce energy dissipation caused by the existing encryption algorithms and hash functions.

Very recently, we had designed and implemented software modules to separately handle read [17] and write [9] requests within the BUD architecture, which will be briefly described in the next Section. In addition, we had made some generalizations about the BUD architecture. One of our

* Corresponding Author. xqin@auburn.edu http://www.eng.auburn.edu/~xqin

978-0-7695-3741-2/09 $25.00 © 2009 IEEE
DOI 10.1109/NAS.2009.71

previous studies showed that the BUD architecture is extremely sensitive to hard disks' *Break Even Time*, which is defined as the size of an idle time required for a disk to energy efficiently transition from the active state to the standby state. In hard disks, for example, the break even time often exceeds 10 seconds. Such a large disk break even time indicates that the BUD architecture is maximally energy efficient in applications that are not disk intensive. Any software module that leaves an opportunity for these break even times to be met allows the BUD architecture to save energy. Hence, security modules that can take full advantage of BUD to conserve energy should not bottlenecked at disk I/O operations. In other words, disk requests issued by the security modules must be sufficiently sparse to produce noticeable energy savings.

To answer the fundamental question of whether we can improve energy efficiency of secure parallel disk systems without modifying security mechanisms in the disk systems, we choose the BUD disk architecture as target energy-efficient disk systems. A vital part of this study is to profile encryption algorithms and hash functions in the context of disk systems. We intend to figure out if I/O access patterns of the encryption algorithms and hash functions would allow the energy-efficient BUD disk architecture to conserve energy. The key I/O features we focused on were arrival patterns of disk request operations issued by the encryption algorithms and hash functions. We aimed to determine if the disk operations can yield sufficient idle periods for the BUD disk architecture to reduce disk energy consumption using the energy-efficient data management strategies we have previously studied.

The rest of the paper is organized as follows. Section 2 gives a summary of the related work for this research. Section 3 provides an overview of an energy-efficient disk architecture. Section 4 describes our test bed setup. Section 5 presents the experimental results and explanations of the trends in our results. Finally we end with a conclusion and some future work possibilities.

II. RELATED WORK

Chandramouli *et al*. investigated battery power-aware Encryption algorithms [1]. The main conclusions they reached was that the power consumption changes linearly with the number of rounds of several popular cryptographic algorithms. Their experimental test bed had a laptop connected to a power supply. The power supply was connected to a computer running the Lab VIEW software to graph changes in voltage and current from the power supply. These changes were graphed during the life of the encryption algorithms [1]. Chandramouli *et al*. paid attention on improving energy efficiency of security mechanisms in mobile devices systems [1]. Our research is radically different from theirs in the sense that we are focused on energy-efficient disk systems without modifying existing security mechanisms.

Potlapally *et. al* characterized the energy consumption of cryptographic algorithms and security protocols [2]. The work undertaken by Potlapally *et. al* [2] was very similar to the research project conducted by Chandramouli *et al*. [1]. Potlapally *et. al* used IPAQ PDA's instead of using a laptop to measure power consumption [2]. They connected the IPAQ to a power supply that was connected to a computer running the Lab VIEW software. This allowed them to measure the energy differences between various cryptographic algorithms. Potlapally *et. al* obtained an array of interesting results related to the SSL Protocol processing. For example, they determined that for small data sizes asymmetric algorithms dominated symmetric algorithms in terms of energy consumption. For large data sizes symmetric algorithms consume significantly more energy than asymmetric algorithms. Network protocol energy consumption, defined as non-cryptographic processing necessary to establish the SSL protocol, does not vary much for different data sizes. Their main observations are that asymmetric algorithms consume the most energy with hash algorithms having the smallest energy footprint. Potlapally *et. al* also stated that asymmetric algorithms energy consumption is dependent on the key size used. They also determined that the level of security and energy consumption can be tuned using key size and number of rounds [2].

Our research differs from the aforementioned research because we are focused on the energy impact of encryption algorithms and hash functions on disk systems rather than mobile devices. Furthermore, the goal of our work is to characterize the I/O behavior of encryption algorithms and hash functions in the context of parallel disk systems (e.g., the BUD architecture). We intended to find the existing security mechanisms that produce the most energy savings in an energy-efficient parallel disk system. Rather than implementing energy-efficient security mechanisms by updating the existing security services, we made the first step towards seamlessly integrating security services with energy-efficient disk systems without changing the source code of the existing security mechanisms. More importantly, our research is orthogonal to the above work in that energy-efficient security mechanisms can be incorporated into our energy-efficient parallel disk architecture to achieve both high security and energy efficiency for parallel disk systems.

III. OVERVIEW of THE BUD DISK SYSTEMS

Although a significant amount of energy can be saved if idle disks are turned to the standby mode, short idle periods (i.e., smaller than the disk break even time) prevents idle disks to be switched to standby to conserve energy. This problem can be solved by aggregating smaller idle periods into idle times that are larger than the disk break even time. We implemented an idle time aggregation process in the buffer-disk architecture or BUD (see Fig. 1 below) using buffer disks to temporally buffer disk requests while keeping data disks to standby as long as possible.

The buffer disk controller – a center piece in the BUD architecture – is responsible for the dynamic power management in both buffer and data disks. The two areas where our previous studies have focused is the buffer disk controller [3] and energy-efficient prefetching [17]. Moreover, we have extensively investigated a disk write buffer strategy

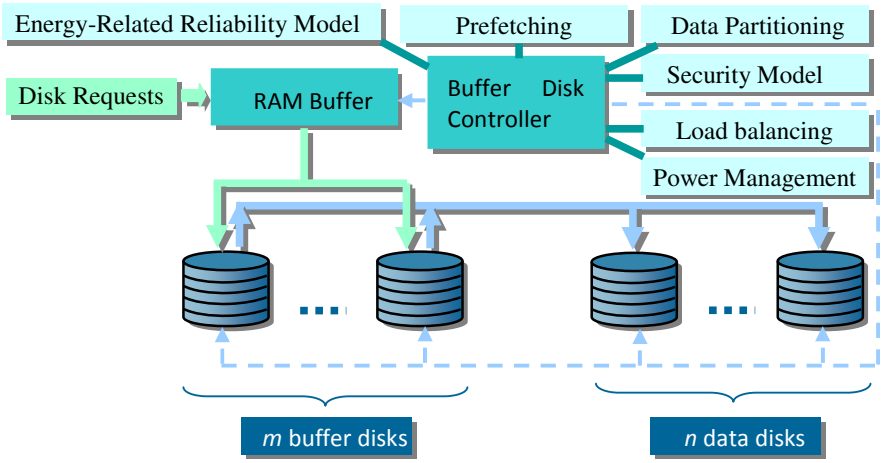

Fig. 1. The buffer-disk architecture or BUD for parallel disk systems.

to improve parallel I/O energy efficiency [9]. There has been work on load balancing the buffer disks and controlling writes to the data disks [18]. For example, buffering write requests in the buffer disks and writing out data when certain criteria are met [9][18]. To further improve energy efficiency of parallel disk systems, we have developed energy-efficient data partitioning schemes, data placement strategies, and data movement algorithms [19][20]. It is worth noting that the BUD architecture embraces a security component, which not only provides security services but also measures security overhead imposed by the integrated security mechanisms. The detailed information concerning the BUD architecture along with a set of energy-efficient data management strategies can be found in [3][9][17].

IV. EXPERIMENTAL SETUP

In the BUD architecture prefetching and data buffering are closely related to total capacity of buffer disks as well as the arrival rate of disk requests. The buffer disks can, literally speaking, prefetch and buffer data aggressively until the disk capacity is reached. When the buffer disks become full, either a portion of buffered data must be moved to the standby data disks or part of prefetched data has to be evicted from the buffer disks. The time taken for the buffer disks to reach their capacity largely depends on request arrival rates, data size, and storage capacity. Since the total buffer disk storage space is managed by with the BUD controller, this study was not meant to address the issue of buffer disk capacity. Thus, let us focus on access patterns (e.g., disk request arrival rates and I/O processing time) to explore the possibilities of achieving high energy efficiency in secure parallel disk systems without modifying security mechanisms.

To capture access patterns of disk requests issued by confidentiality and integrity services, we have to profile encryption algorithms and hash functions. We chose to use a Linux computer because of the open source nature of Linux and availability of free software like GkrellM, Conky and XySSL. The first software monitor used in our experiments is GkrellM. Although GkrellM is capable of providing disk, memory, and CPU usage statistics, we were unable to find a quick method to produce the output in a text file. Hence, we moved to using Conky, which is a lightweight system monitor that is highly configurable and supports text output. After making use of the XySSL libraries to evaluate an array of security mechanisms, we generated disk trace files for further analysis.

Our created a testbed using one Linux PC. The following table outlines the important properties of the testbed.

Table 1 System Parameters of the Testbed

CPU Speed	Pentium 4 2.4 GHZ
Memory	512 MB
Operating System	Ubuntu 7.10
USB 1.1	12 Mb/s
HD Bus	IDE

The CPU speed of the testbed is high enough that the CPU will probably not be the performance bottleneck for the evaluated security mechanisms. The bandwidth of the memory component in our testbed is relatively low for a Pentium 4 computer. We chose to connect a flash drive to the testbed by a USB port, because the USB interface was used to emulate a network interface card in our experiments. A hard drive is connected to a SCSI Bus with the bandwidth of at least 5MB/s.

The Linux operating system was used to perform our experiments, because it is probably a bit easier to find the required software needed to profile encryption algorithms and hash functions. The Linux-based software tools used in our experiments allow us to make any changes to the software to monitor and study access patterns of security services.

Once we have our OS chosen, we are in a position to identify software that allow us to test our encryption algorithms and hash functions. An ideal software tool should include the implementation of a wide range of popular

encryption algorithms and hash functions. Apart from a security software tool, we have to choose a software tool enabling us to monitor the CPU, memory, and read/write performance of the hard drive and flash drive connected to the USB port. After we found software to fulfill the above requirements, we were ready to profile encryption algorithms and hash functions.

The first software tool we chose to use is XySSL, which implements a set of well-known encryption algorithms accompanied by testing programs. In this study, we pay particular attention to two encryption algorithms - 3DES [12] and AES [14]. 3DES – slow in software - was developed in response to the weakness of DES. 3DES is a strong encryption algorithm, but it is typically slower than AES. 3DES uses a 64 bit block size to encrypt data [6]; it uses a 192 bit key that is split into three different keys. In the 3DES algorithm, these three 64 bit keys are required to encrypt and decrypt data using DES. Each of these 64 bit keys uses 8 bits for parity checking leaving 3DES with an effective key strength of 168 bits.

AES is the current standard for data encryption widely adopted by the US Government [6]. AES is typically employed with a 192-bit encryption key. Block size in AES is 128-bit in length, meaning that AES encrypts twice as much data as 3DES at each call of the function implementing the encryption. AES is not only fast in software but also requiring a small amount of memory. In addition to encryption algorithms, several hash functions were implemented in XySSL. To this end, we decided to evaluate MD5 [10] and SHA (1-2) [11]. MD5 is an algorithm that produces a hash output value of 128 bits. SHA-1 generates 160-bit hash values, whereas SHA-2 can provide a hash bit varying in size between 224 and 512 bits. We also evaluated RSA verification implemented in XySSL. Please note that all of the aforementioned encryption algorithms and hash functions, except for 3DES, are coupled with testing programs in XySSL. Thus, we had to implement a testing program for 3DES based on the AES testing program.

To monitor our testbed computer, we take full advantage of GKrellM - a software monitor that is capable of monitoring the CPU, memory, and disk I/O subsystems. GKrellM seems to be a promising software tool. The only problem with GKrellM, however, is the lack of well defined text output. There is no straightforward way to produce text output. Hence, we would have to dig into the source code of GKrellM and implement a module to deal with text output. Alternatively, we would have to find another system monitor that has some kinds of text output. We chose the second approach due to time constraints and began our search for a system monitor with text output. Fortunately, Conky - a lightweight system monitor that is highly configurable - is able to yield a text output file. There is a listing of all the variables that Conky keeps track of. If we need to output of these variables to a text file, we simply specify the variable in a configuration file. The configuration file allows us to format output files, making it possible for us to put our desired variables in the csv format readable by Microsoft Excel. In our experiments, the variables that are kept track of include events of CPU, memory, and disk read/writes in the testbed. Since we substituted a USB-based flash drive for a networked disk system, all data reads are initiated from the USB drive. We were able to control the testing programs in XySSL to guarantee all disk writes are physically issued to the hard disk in our testbed. In doing so, we are capable of separately monitoring I/O access patterns of the flash drive and hard drive where the encryption algorithms and hash functions are evaluated.

V. EXPERIMENTAL RESULTS AND ANALYSIS

Now we are positioned to present experiment results and analysis in the this sections. We evaluated three different types of system events, namely, CPU usage, flash drive read bandwidth, and hard disk write bandwidth. Recall that we extensively investigated three hash-function algorithms, one RSA signature verification algorithm, and two block and stream encryption algorithms in our experiment.

In our experiment, we use a flash drive to emulate source data retrieved from a remote disk system connected to the testbed through a network. The size of the source data set is 900Mbyte; the data was organized and stored in 78 files residing in the USB-based flash drive.

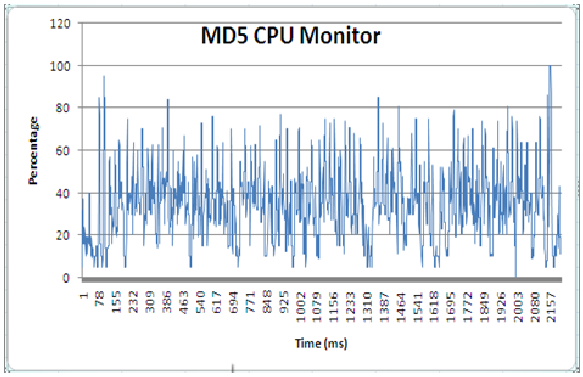

Fig. 2. CPU usage (measured in percentage) of the testbed when MD5 is evaluated.

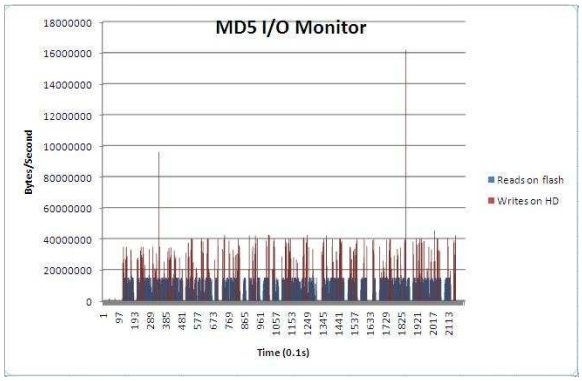

Fig. 3. Read/write bandwidth of the testbed when MD5 is evaluated.

MD5 used to be one of the most popular hash functions for data integrity checking services. Although MD5 is no longer considered as the most secure algorithm for integrity checking [15], it is still a good representative hash function to be considered in this research. Hence, we started our experiment by exhibiting the workload of CPU and I/O of the testbed when MD5 is running to ensure that a file has not been tampered with. Fig. 2 shows the CPU usage when the testbed is using MD5 for integrity checking. Since the MD5 hash function is not CPU-intensive, we observed from Fig. 2 that the CPU usage is almost always lower than 60% and very rarely exceeds 80%. Hence, we concluded that CPU is not the performance bottleneck of MD5.

Fig. 3 shows the read/write bandwidth of the disk subsystem in the testbed. It is observed from Fig. 3 that 15MB/Sec. is the maximum I/O read bandwidth achieved by the testbed MD5 is evaluated. Such a maximum read bandwidth is apparently the upper bound of the bandwidth exhibited by the USB-based flash drive. A second observation drawn from Fig. 3 is that the average write bandwidth of the disk subsystem is 32MB/Sec., which rarely reach the maximum write bandwidth. More interestingly, the flash drive experiences a limited number of idle periods during the execution of MD5. Compared with read access patterns, write access patterns contain more and longer idle times. Fig. 3 reveals that data read bandwidth becomes the performance bottleneck of the disk system where MD5 is applied for data integrity checking. More importantly, Fig. 3 suggests that without modifying MD5, we can apply the BUD architecture to reduce energy consumption of disk systems integrated with MD5 for data integrity checking. BUD can conserve energy for MD5, because many small idle periods in disks can be grouped to form large idle periods, which in turn allow disks to be switched to the standby mode to save energy.

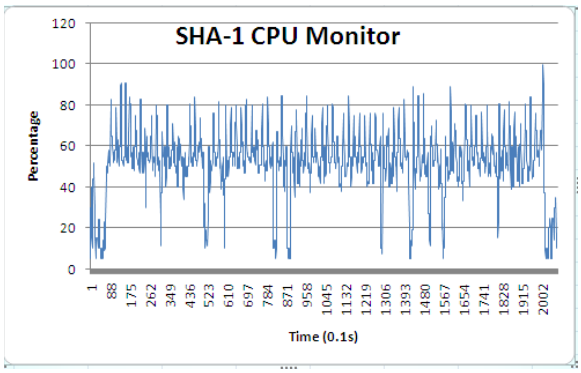

Fig. 4. CPU usage (measured in terms of percentage) of the testbed when SHA-1 is evaluated.

SHA-1is a second hash function evaluated in our testbed. The access patterns of SHA-1 are very similar to those of MD5. SHA-1 is more secure than MD5 (see [16] for detailed comparisons), because SHA-1 is more complicated in implementation than MD5. Fig. 4 indicates that the average CPU usage of the testbed when SHA-1 is evaluated is higher than that of the same testbed when MD5 is running.

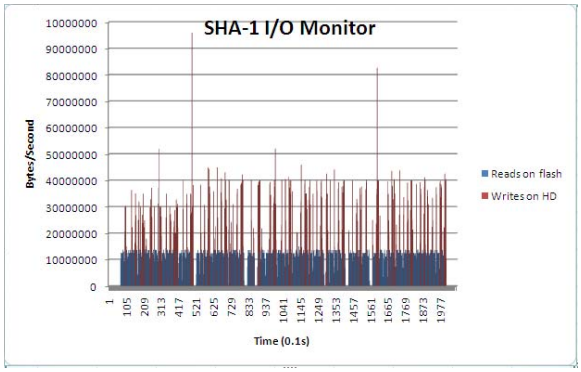

Fig. 5. Read/write bandwidth of the testbed when SHA-1 is evaluated.

Fig. 5 shows that in almost all the cases, the testbed keeps reading data from the flash drive at the highest bandwidth. There is only very small number of gaps among reading events, meaning that idle periods rarely occur in SHA-1. Unlike read requests, disk write operations (see Fig. 5) in SHA-1 demonstrate more idle periods. This is mainly because with respect to SHA-1, reading data from the flash drive is the performance bottleneck of the testbed system. Like Fig. 3, Fig. 5 indicates that the SHA-1 can be seamlessly integrated with the BUD architecture to achieve both high energy efficiency and data integrity. Such integration can be straightforwardly realized by employing a software module of the existing SHA-1 service on top of BUD without even changing the source code of SHA-1.

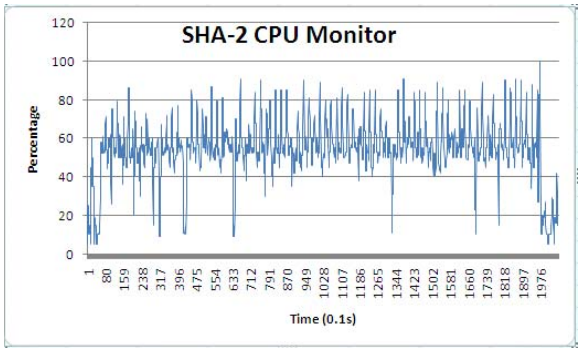

Fig. 6. CPU usage (measured in terms of percentage) of the testbed when SHA-2 is evaluated.

SHA-2 is comprised of a group of hash functions. Without loss of generality, in this experiment let us evaluate SHA-256, which can be considered as one of the most secure hash functions in this study. Fig. 6 shows that when SHA-2 is evaluated, the CPU usage of the testbed system is slightly higher than that of the same testbed with SHA-1. Fig. 7 indicates that disk I/O characteristics of SHA-2 are very close to those of SHA-1. In other words, data read bandwidth of the

flash drive is the performance bottleneck of SHA-2. An implication of this result is that without changing the source code of SHA-2, it is possible to leverage the BUD architecture to improve energy efficiency of disk systems where SHA-2 is employed.

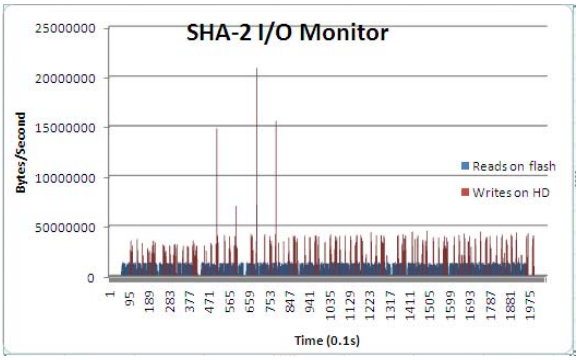

Fig. 7. Read/write bandwidth of the testbed when SHA-2 is evaluated.

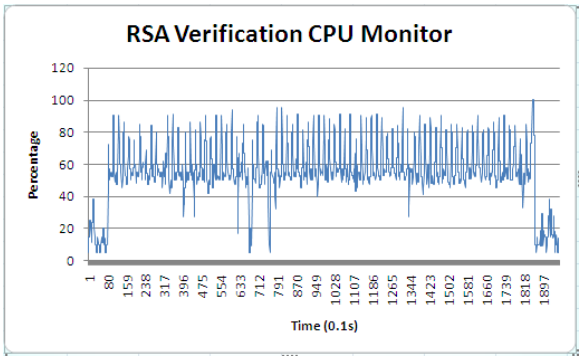

Fig. 8. CPU usage (measured in terms of percentage) of the testbed when RSA Verification is evaluated.

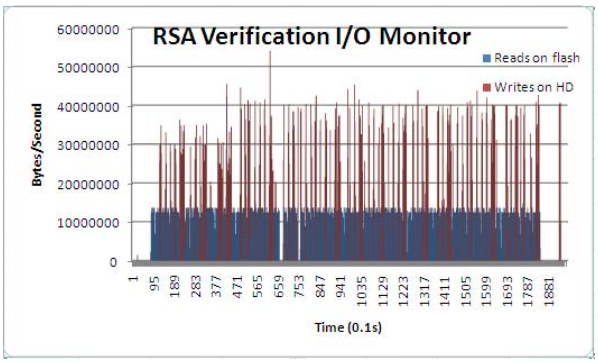

Fig. 9. Read/write bandwidth of the testbed when RSA Verification is evaluated.

RSA - widely adopted as a public-key encryption scheme. - has signature generation and verification functions. In this experiment, we evaluate RSA's signature and verification functions in our testbed. Before we started the test, we generated signature files for each file residing in the USB-based flash drive. In this case, each input data file is coupled with a corresponding signature file.

We observed from Figs. 8 and 9 that RSA processes the input data set faster than the other three hash functions, suggesting that the CPU and I/O load imposed by RSA verification function is lower than those of the above studied hash functions.

Since the CPU load in RSA is reduced compared with the other three hash functions, Fig. 9 confirms that the read bandwidth of the flash drive is the performance bottleneck of the testbed. Due to the fact that very few idle periods exist in the flash drive, it is unlikely to leverage the BUD architecture to reduce energy dissipation of reads in RSA. Fortunately, the results obtained from Fig. 9 indicate that executing RSA in the BUD disk architecture can provide energy savings for writes issued to the hard drive. The energy savings become possible for writes in RSA, because there are a large number of small idle periods that can be aggregated by BUD to form larger idle periods.

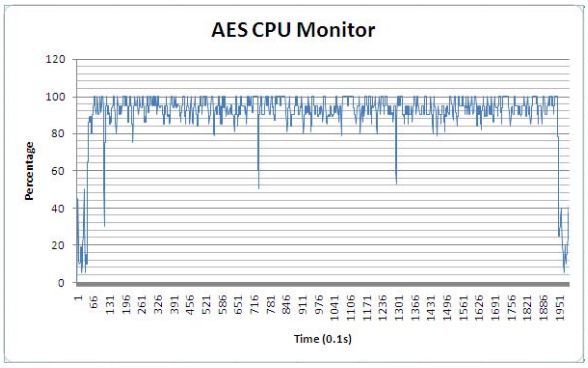

Fig. 10. CPU usage (measured in terms of percentage) of the testbed when AES is evaluated.

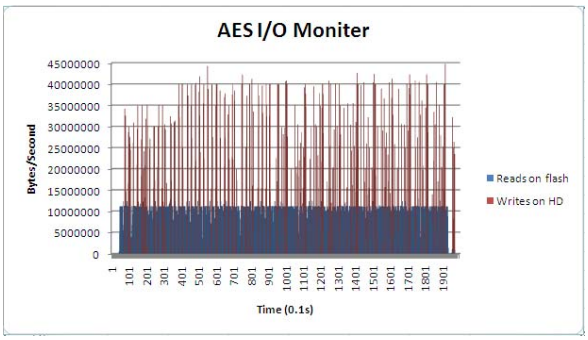

Fig. 11. Read/write bandwidth of the testbed when AES is evaluated.

Next, let us study the access patterns of AES and 3DES - two block/stream encryption algorithms. The AES (Advanced Encryption Standard) is a block cipher standard published by the US government in November 2001. After input data is retrieved from the flash drive in the testbed, AES encrypts the input data with 256-bit keys and stores cipher text on local hard drive in the testbed. Results plotted in Figs. 10 and 11

show that the CPU and I/O load caused by AES are very well balanced. For example, the CPU usage, read/write bandwidth of the testbed running AES tend to reach their upper bounds. When it comes to reads, there is almost no idle period found during the course AES's execution. Although AES does not issue writes as intensively as reads, idle periods in the hard drive are smaller than those of cases for MD5, SHA-1/SHA-2, and RSA.

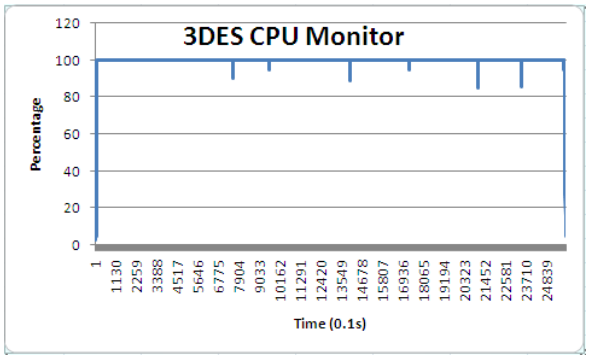

Fig. 12. CPU usage (measured in terms of percentage) of the testbed when 3DES is evaluated.

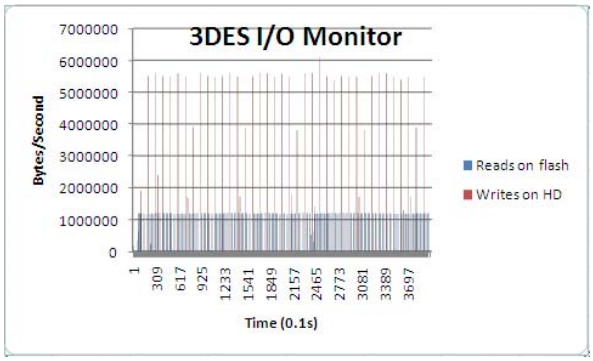

Fig. 13. Read/write bandwidth of the testbed when 3DES is evaluated.

3DES, CPU-intensive in nature, is very slow in software. For example, it took approximately 190 Sec. for AES to encrypt all the 78 files in the flash drive; 3DES spent more than 25,000 Sec. in processing the same set of files. Fig. 12 clearly shows that the CPU load is extremely high and CPU becomes the performance bottleneck in our testbed running 3DES. Results depicted in Fig. 13 suggest that the BUD disk architecture can be seamlessly integrated with 3DES to cluster small disk idle periods together in large idle time frames, which enable disks to be operated in the standby mode for long time intervals to save energy.

VI. CONCLUSIONS AND FUTURE WORK

Achieving both high energy efficiency and security in disk systems is challenging, because energy efficiency and data security are often two conflicting goals. There are two general approaches to improving security and energy-efficiency in disk systems. The first approach is to modify existing security mechanisms to enhance energy efficiency of security services in disk systems. In contrast, the second approach is to seamlessly integrate security services with energy-efficient disk systems without modifying security mechanisms. In this research, we took the first step toward the second approach by answering an intriguing question of whether we can improve energy efficiency of security mechanisms in disk systems without changing the source code of security services.

Target energy-efficient disk systems considered in this study is the buffer disk architecture or BUD (see [9] for detailed information regarding BUD). The BUD disk architecture aims at aggregating many small idle periods in disks into a few large idle intervals so that disks can be turned into the standby mode and kept in standby as long as possible to conserve energy. In this research, we first built a testbed containing a disk subsystem, USB-based flash drive, Linux operating systems, and six encryption modules and hash functions. Next, we captured CPU usage and I/O access patterns of a disk system where the encryption modules and hash functions were tested and evaluated. Finally, we analyzed the possibility of leveraging the BUD disk architecture to reduce energy consumption incurred by the existing encryption algorithms and hash functions in the context of disk systems.

Table 2 Summary of the CPU usage, read/write load of the testbed running the six encryption algorithms and hash functions. The two rightmost columns show possibilities of employing the BUD architecture to save energy for secure disk systems without modifying the security mechanisms. (L: Low, M: Medium, H: High, VH: Very High EH: Extremely High)

	CPU Load	Read Load	Write Load	Save Energy for Reads?	Save Energy for Writes?
MD5	M	H	M	Unlikely	Yes
SHA1	M	VH	M	Unlikely	Yes
SHA2	M	VH	M	Unlikely	Yes
RSA	M	VH	M	No	Yes
AES	VH	VH	M	No	Yes
3DES	EH	M	L	Yes	Yes

Table 2 summarizes the CPU utilization and read/write bandwidth of the testbed running the six security services. Table 2 shows that except for 3DES, the CPU is not a performance bottleneck for the other five security services. Among the six evaluated security services, 3DES is the only one that can make use of BUD to improve energy efficiency for disk reads and writes. This is mainly because 3DES is very slow in software, leaving many small idle periods in the hard disks and flash drive. Although we are unable to conclude that 3DES is the most energy efficient security service on BUD, it is certain that 3DES is a good representative security mechanism that can benefit a whole lot from BUD.

Furthermore, BUD can be employed to reduce energy consumption of writes issued by the MD5, SHA-1, SHA-2, RSA, AES modules (see the rightmost column in Table 2). It is unlikely to minimize energy consumption of reads for MD5, SHA-1, SHA-2 using BUD. Even worse, it is almost impossible for BUD to reduce energy consumption of reads for RSA and AES. Now we are positioned to conclude that the BUD disk architecture can provide an ideal energy-efficient data storage platform for security mechanisms, which have high CPU usage and issue sparse disk requests.

As part of future work, we will take full advantage from the generated I/O traces to drive our BUD disk simulator, which allows us to quantitatively study how BUD can produce energy savings for security mechanisms. Currently, we are in a process of building a cluster storage system based on the BUD architecture to investigate if BUD can be applied to clusters to conserve energy. For a large cluster storage system with sufficiently large CPU and disk capacities, it is intriguing to evaluate the energy efficiency of a variety of hash functions and encryption algorithms on the BUD-based cluster storage system. For the I/O-intensive security services, we will implement a dynamic resource manager in BUD to keep parallel data disks active to achieve high aggregated read bandwidth while allocating a limited number of CPUs to perform integrity checking. In doing so, BUD will be capable of conserving energy dissipation in processors in the cluster storage system. When it comes to CPU-intensive security services, the dynamic resource manager will save energy by keeping all CPUs active to carry out encryption while force a small number of buffer disks to buffer data.

ACKNOWLEDGMENTS

This work was made possible partially thanks to NSF awards CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), and an Intel gift (number 2005-04-070) as well as an Auburn University startup grant.

REFERENCES

[1] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi, and R. N. Uma, "Battery Power-Aware Encryption," *ACM Trans. Information and System Security,* pp. 162-180, May 2006.

[2] N. Potlapally, N., S. Ravi, A. Raghunathan, and N. Jha, 2006. "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols," *IEEE Trans. Mobile Computing*, pp. 128-143, Feb. 2006.

[3] Z. –L. Zong, M. Briggs, N. O'Connor, and X. Qin, "An Energy-Efficient Framework for Large-Scale Parallel Storage Systems," *Proc. 21st Int'l Symp. Parallel and Distributed Processing,* March 2007.

[4] L. Lu and P. Varman, "DiskGroup: Energy Efficient disk Layout for RAID1 Systems," *IEEE Int'l Conf. Networking, Architecture, and Storage*, pp. 233-242, Jul. 2007.

[5] B. Mao, D. Feng, H. Jiang, S. Wu, J. Chen, and L. Zeng, "GRAID: A Green RAID Storage Architecture with Improved Energy Efficiency and Reliability", *Proc. IEEE/ACM Int'l Symp. Modelling, Analysis and Simulation of Computer and Telecommunication Sys.*, pp.1-8, Sept. 2008.

[6] C. Parikh and P. Patel, "Performance Evaluation of AES algorithm on Various Development Platforms," *IEEE Int'l Symp. Consumer Electronics*, pp. 1-6, Jun. 2007.

[7] P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen, "Configurable Hardware Implementation of Triple-DES Encryption Algorithm for Wireless Local Area Network," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal, pp. 1221-1224*, May. 2001.

[8] A. Nadeem and Y. Javed, "A Performance Comparison of Data Encryption Algorithms," *Proc. Int'l Conf. Information and Comm. Tech.*, pp. 84-89, Aug. 2005.

[9] X.–J. Ruan, A. Manzanares, K. Bellam, X. Qin, and Z. –L. Zong, "DARAW: A New Write Buffer to Improve Parallel I/O Energy-Efficiency," *Proc. 24th Annual ACM Symp. Applied Comp.*, Mar. 2009.

[10] R. L. Rivest, The MD5 Message-Digest Algorithm, *RFC 1321, MIT Laboratory for Comp. Sci. and RSA Data Security, Inc.*, Apr. 1992.

[11] Federal Information Processing Standards. Secure Hash Standard. *FIPS PUB 180-2*. Aug. 2002.

[12] "Data Encryption Standard," Federal Information Processing Standards (FIPS) Publication 46-7, National Institute of Standards and Technology (NIST), USA, 1999.

[13] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[14] National Institute of Standard and Technology. Advanced Encryption Standard FIPS 197 [S]. Nov. 2001.

[15] X.–Y. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," *EUROCRYPT 2005*, pp. 19-25, May. 2005.

[16] X.–Y. Wang, Y. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," *Proc. Crypto*, pp. 17-36, 2005.

[17] A. Manzanares, K. Bellam, and X. Qin, "A Prefetching Scheme for Energy Conservation in Parallel Disk Systems," *Proc. NSF Next Generation Software Program Workshop*, April 2008.

[18] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," *Proc. 6th USENIX Conf. File and Storage Technologies*, Feb. 2008.

[19] M. Nijim, A. Manzanares, and X. Qin, "An Adaptive Energy-Conserving Strategy for Parallel Disk Systems," *Proc. 12th IEEE Int'l Symp. Distributed Simulation and Real Time Applications*, Oct. 2008.

[20] C. Liu, X. Qin, S. Kulkarni, C.-J. Wang, S. Li, A. Manzanares, and S. Baskiyar, "Distributed Energy-Efficient Scheduling for Data-Intensive Applications with Deadline Constraints on Data Grids," *Proc. 27th IEEE Int'l Performance Comp. and Communications Conf.*, Dec. 2008.