DOE Early CAREER PI Award Final Report

Toward Highly Secure and Autonomic Computing Systems: A Hierarchical Approach

School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332

Principal Investigator: Hsien-Hsin S. Lee
leehs@gatech.edu Tel: (404) 894-9483 Fax: (404) 385-3137

1 Project Overview

The overall objective of this research is to develop novel architectural techniques as well as system software to achieve a highly secure and intrusion-tolerant computing system. Such system will be autonomous, self-adapting, introspective, with self-healing capability under the circumstances of improper operations, abnormal workloads, and malicious attacks.

The scope of this research includes: (1) System-wide, unified introspection techniques for autonomic systems, (2) Secure information-flow microarchitecture, (3) Memory-centric security architecture, (4) Authentication control and its implication to security, (5) Digital right management, (5) Microarchitectural denial-of-service attacks on shared resources. During the period of the project, we developed several architectural techniques and system software for achieving a robust, secure, and reliable computing system toward our goal. We described our research activities and findings chronically in the following sections.

2 Research Activities

The descriptions of our research activities on this project are detailed individually in the subsequent sections. Broadly, we investigated the security and automatic computing capability from several different levels — microarchitectural, architectural, and memory subsytem. For the detailed intellectual outcomes, please refer to our project website at http://arch.ece.gatech.edu/research/research-secure.html.

2.1 Memory-centric Security Architecture

2.1.1 Problem Statement and Approaches

There is a growing interest in creating tamper-resistant and copy protection systems that combine the strengths of security hardware and secure operating systems to fight against both software attacks as well as physical tampering. These systems are aimed at solving a variety of security issues such as digital rights management, virus/worm detection, rootkit detection, intrusion detection and prevention, digital privacy, etc. Recent industry thrusts including Intel's LaGrade Technology using Trusted Platform Module (TPM) and IBM's SecureBlue 4758 on-a-chip were developed to address these potential threats.

These prior secure systems mostly achieve protection by encrypting the instructions and data of a user's process with one single master key burned inside the processor core. Although such a closed system solution does provide security for software execution, however, they are less practical for real world applications as most of them are multi-domained where a user process often comprises components supplied from heterogeneous program sources with distinctive security requirements. One example is the pervasive use of dynamic linked libraries (DLL) or shared libraries written by various vendors. It is quite natural that these library vendors would prefer a separate copy protection of their own intellectual properties decoupled from the user's application. Furthermore, it is also common for different autonomous software domains to share and exchange classified information at both the inter- and intra- process levels. To enable the secret shar-

ing between different software components under the same application space, new solutions are needed to achieve this goal.

Similar to object-oriented management, the basic idea of our work is to provide protection on software integrity and confidentiality based on an atomic system element what we call *memory capsules*. A memory capsule is a virtual memory segment with a set of common security attributes associated with it. It itself is an information container that may hold either code or data memory page or both. It is designed to be shared among multiple processes. For example, one can consider one DLL provided by vendor A as a single code memory capsule. The vendor will designate the security attributes including desired security protection level, one or more symmetric memory encryption keys, the capsule's memory authentication signature, access control information, etc. The multi-vendor memory capsules of a running process are maintained and managed by the secure OS kernel. During software's distribution, software vendors encrypt the security attributes associated with a memory capsule using the secure processor's public key. Then the secure processor authenticates and extracts the security attributes using its corresponding private key. Based on this concept, we develop a secure architecture called *Memory-centric Security Architecture* or *MESA*. To provide security seamlessly, architectural support and microarchitectural enhancement were proposed for MESA. They include new instruction support for allocating data on private heap, passing confidential secrets between subroutines in a secure way, new memory management modules in the hardware to enforce security policies defined by software vendors, information access control mechanism in the cache, dynamic checking hardware for verifying each indirect memory access.

2.1.2 Research Findings

To evaluate our proposed architecture, we used Bochs, an open-source full system emulator and TAXI, an x86 execution-driven simulator integrated within Simplescalar. In our simulation work, we assume that the application software and the system software need to be protected separately. We evaluated seven Windows NT applications including Internet Explorer 6.0, Acrobat Reader 5.0, Windows Media Player 2, Microsoft Visual Studio 6.00, Winzip 8.0, Microsoft Word, and Povray 3. We are interested in knowing the performance impact when such security features are included into a secure processor. We consider two styles of secure processors — counter-mode based secure processor and block-cipher based one. On average, we found that the entire security implementation based on a counter-mode secure processor caused roughly 6% performance degradation compared to a baseline processor without any security enhancement. When switching to a block-cipher based secure processor, the slowdown was increased up to 15%. Both results were based on the assumption that all the software components are to be protected and encrypted. Oftentimes, this is not the case. The vendors can choose to selectively protect and encrypt those parts that are considered secret. To evaluate this more realistic scenario, we leave all the system DLLs belonged to the OS unencrypted. By doing so, we observed the slowdown of a counter-mode processor dropped down to 4.4% while the slowdown of a block-cipher based processor dropped to 9.5%. For supporting security in a processor, such performance degradation is reasonably tolerable. Please refer to [10] regarding the detailed architecture design and experimental results.

2.2 InfoShield: Protecting Information Usage

2.2.1 Problem Statement and Approaches

In this particular thrust of our work, we tried to understand the common exploit mechanisms used by adversaries and investigate how secrets and privacy are compromised from the system memory. The exploit techniques studied include memory scans, buffer overflow exploits, password stealing worms/Trojan horses, as well as invalid memory pointers manipulation that reveals secrets.

The causes of data privacy violation are diverse, so are the solutions. There are few studies in the past that directly address the problems from the above attack models. In many cases, data privacy often comes as a by-product of safe programming practices using either security analysis tools or an intrinsically safe source

language. Prior research efforts such as *access control* and *information flow analysis* attempted to address the data safety issues directly. Nevertheless, traditional access control approaches do not provide sufficient protection on information security because the OS-based access control is too coarse-grained. On the other hand, using static information flow analysis to ensure privacy and security of information is sometimes too restrictive and cumbersome for real applications where sharing information is mandatory and frequent.

To address these shortcomings, we propose a new concept called *InfoShield* in this work. The fundamental idea and distinction of our proposed solution is the following. Instead of tracking and protecting information flow, we protect the privacy of sensitive information based on *information usage*. As can be readily seen from Internet viruses and spyware, misuse of information is the major threat to information security. Many attacks stem from abnormal or unauthorized usage of information. For example, a password or encryption key must be strictly used for access authentication by the designated functions based on a specific flow of instruction usage. Any other usage e.g. memory scan or redirecting execution to a different instruction path, must be considered unsafe and prohibited. Our model provides the most restricted information manipulation mechanism enforced by both software writers and architectural support. The characteristics of our information usage based protection include improved data privacy, increased enforceability, optimized performance and composable security. Our protection is basically achieved by the use of a set of new security-aware instructions. Note that since we are only interested in protecting critical data such as the code regions that read and authenticate personal information, hence the overhead of the additional instructions in the code will be minimal. Conceptually, the implementation of information usage protection is fairly simple. Consider a hypothetical example where an encryption key is created and then used. Upon the declaration of the encryption key or its pointer, the programmer annotates the source code indicating the contents of the key are sensitive. When the storage of the key is created (e.g. via malloc), the compiler inserts secure-aware instructions that records the returned address with the annotation indicating that it is an address for sensitive data. Prior to passing the pointer to the surrounding code for proper handling, the compiler inserts additional instructions to *guard* the data itself. Working in tandem with a dedicated hardware security table, this address of the key is entered along with the PC of the *next* instruction that is permitted to access this address. Every single load and store operation needs to check this security table to ensure that no access incurs when the PC is not the designated next-access PC by the compiler. Such violations raise a security exception. In essence, these instructions that manipulate sensitive data form an authentication chain. Any attempt to break the chain or insert bogus security instructions within the chain is subject to be detected, followed by a security exception.

2.2.2 Research Findings

To evaluate the idea of information usage protection and the specifics of InfoShield design, we used a number of network applications that manipulate sensitive user data such as login passwords, cryptographic keys, or other credentials. They include FTP server, Apache web server, email server, FTP client, web browser (Lynx) and SSH daemon. Similar to memory tainting techniques, we manually identified sensitive data based on the application source code and annotate the application for our emulation. Bochs, the x86 full-system emulator is used again in this work.

Based on our emulation results, the performance impact of InfoShield on application execution is very small, that is understandable as compared to the entire application, the amount of data and code that handles passwords, crypto-keys is rather insignificant. For all regular memory accesses, less than 0.002% instructions access sensitive information for five applications (out of eight we evaluated). The worse case is the Apache webserver that contains 1.8% of the memory instructions accessing sensitive information. In addition, we also evaluated the extra hardware requirement based on the workloads we vaulted. For all these applications, a 32-entry table (consuming around 1KB space) is sufficient to provide needed security of information usage. For more details, please refer to [4].

2.3 Autonomic Computing via Multicore Processors

2.3.1 Problem Statement and Approaches

Another area of preventing from security being compromised is to detect, recover and survive through malicious attacks in an autonomic manner. In other words, the system is self-aware and self-healing with built-in self-recovery mechanism, thus requiring less attention from human staff. In some sense, fault-tolerance and security are dealing with the same issue from different perspectives: one focuses on events of unintentional faults due to soft errors or program bugs, while the other fights against the intentional exploits from malicious adversaries. An effective solution can alleviate or even eliminate one issue will be likely to work for the other. To succeed toward this goal, a system needs to comprehend system behavior, adapt from dynamic scenarios, and respond with appropriate countermeasures in order to detect or repair itself from damages caused by either malicious exploits or transient faults. All of which will require additional if not substantial computational capability. In general, we call such systems *introspective systems*.

We had first examined a system-wide monitoring mechanism called *Owl* [3, 11]. The Owl framework employs pervasive field-programmable gate arrays (FPGA) to perform constant, non-intrusive monitoring at all event sources such as memory buses. This technique improves the shortcomings of existing performance counter based approach by employing an active, reconfigurable monitoring hardware. Additionally, it is flexible and programmable, hence the system owners can update the monitored events and adapt the potential threats with new software patches from time to time. Due to the emergence of multicore processors (or chip multiprocessors), on-chip computing power has been largely leveraged. One advantages of a multicore system is the reduction of core-to-core communication overhead, that enables close-coupled introspection capability using different cores. We studied the feasibility and published our findings in [8]. Continuing from our previous effort, we proposed a new system architecture called *INDRA* for Integrated framework for Dependable and Revivable Architecture. Based on a multicore processor, INDRA integrates novel mechanisms that perform on-line security/fault detection and recovery with an objective of minimizing performance impact. The basic idea is to configure a multicore processor asymmetrically — some cores are used as normal cores running generic service applications and other cores are configured to be sentinels monitoring all events encountered by the service cores. As such, the system creates a *hardware sandbox* with security insulation. With the tightly coupled processor cores on the a single chip, fine-grained instruction level introspection can be carried out with little performance overhead. In our current work, we also proposed a novel backup and recovery scheme at the microarchitectural level to provide instant, timely recovery once a threat is detected. We call our approach *Delta page update* containing a light-weight cost amortized across the entire software execution. Since the recovery mechanism is demand-based, a corrupted memory state is only rolled back on a per-cache line basis when it is accessed again later in the application.

2.3.2 Research Findings

We used Bochs and TAXI again to evaluate our proposed INDRA architecture. We ran two copies of Bochs and used the existing network communication ports in Bochs to communicate between two emulated cores. Several exploits published by Common Vulnerabilities and Exposures (CVE) website were injected into the application cores to mimic the remote exploits. The applications we experimented include FTP server, HTTP server, Sendmail, bind DNS server, NFS file server, and IMAP email server. We implemented three major introspection mechanisms in the software running on the monitoring core. They are: code origin inspection, return address inspection, and control transfer inspection. In terms of the effectiveness from security standpoint, INDRA was able to detect and trigger instant recovery in the face of the remote exploits we injected. Also note that, such introspection is useful in catching transient faults as well, a by-product of the INDRA architecture. Compared to the traditional check-pointing scheme which usually suffers large performance degradation (in our experiment, from 2x to 12x), our INDRA framework only incurs less than 50% slowdown with a worst-case of 2.5x. We also quantified several other system behavior. Please refer to [7] for detailed information.

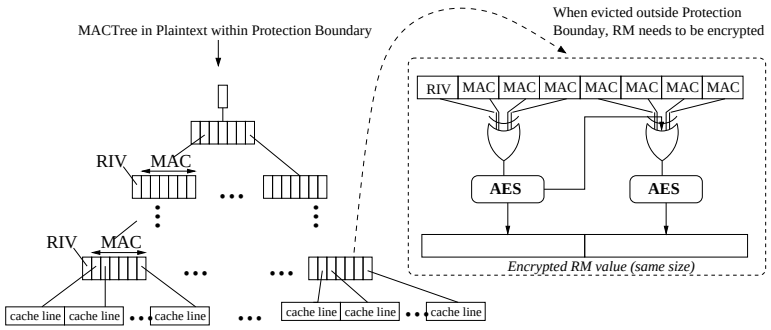


Figure 1: Structure of the MACTree in plaintext within protection boundary. Note that a hash value needs to be encrypted when being evicted out of the protection domain.

2.4 M-TREE Secure Architecture

2.4.1 Problem Statement and Approaches

Secure processor architectures enable new sets of applications such as commercial grid computing, software copy protection, and secure mobile agents by providing secure computing environments that are immune to both physical and software attacks. Despite a number of secure processor designs have been proposed, they typically made trade-offs between security and execution efficiency. To address these shortcomings, we investigated and proposed a novel secure processor architecture called M-TREE [2] that offers a significant performance advantage without compromising security strength. The M-TREE architecture uses a novel technique called hierarchical Message Authentication Tree (MACTree) for protecting applications' integrity at a minimal performance overhead. The M-TREE processor also introduces a new one-time-pad class encryption mechanism that accelerates security computation over the existing block cipher based schemes with high security guarantee.

The baseline model of the M-TREE secure processor assumes all data accessed within the processor core are considered secure whereas once a data block is evicted from the processor, it needs to be protected using a secret key only visible by the processor hardware itself. This key is provided by application vendors and is encrypted with the public key of the secure processor at the beginning of the application's execution. Along with the cipher engine built inside the processor, this key provides confidentiality of data under protection. To provide a tamper-evident computing environment, any unauthorized modification attempts to the applications must be detected with a robust integrity check mechanism. Toward this objective, we proposed the MACTree integrity protection. The construction of the MACTree starts from each data block, in our case, equivalent to the size of a cache line. Each cache line address and data along with the application's secret key are fed into the SHA-256 hash function to generate a 256-bit MAC. Dividing them into 32-bit chunks, we bitwise-XOR them into one single 32-bit MAC (*the root MAC*. All 32-bit MACs of data blocks are concatenated in groups to form the leaf nodes of a hierarchical MACTree shown in Figure 1. Each group consists of one random initiation value (RIV) with seven MACs. To compute a new node one hierarchical level up, the same procedure to produce one 32-bit MAC is repeated until only one MAC is achieved. During a cache line eviction out of the protection boundary (i.e. the secure processor), the RIV/MAC value is encrypted with AES block cipher as shown inside the dotted box in Figure 1. Note that the *root MAC* is always kept inside the protected boundary to avoid any potential tampering.

The final M-TREE processor architecture integrates the M-TREE encryption and integrity checking mechanisms with three microarchitectural enhancements: an integrity verification unit (IVU), a RIV/MAC and MAC cache (RM/MAC Cache), and the encryption/decryption unit (EDU), in order to accomplish all the protection measures using specialized hardware rather than by software. Conceptually, the RM/MAC Cache is very similar to a victim cache for those RIV/MAC and MAC values being cast out of the processor.

The only difference is that the RIV/MAC and MAC kept inside the cache is unencrypted while the one kept outside the cache requires encryption protection. This cache serves the purpose of expediting the processing of decrypting these RM/MAC and MAC values after retrieving them from system memory which could already take hundreds of cycles in a high performance processor system.

2.4.2 Research Findings

To evaluate the performance implications, we simulate our infrastructure using Simplescalar executing Alpha binaries. We used selective SPEC benchmark programs for the applications. The subsetting was carried out by picking the applications with considerable L2 miss rates to demonstrate the effectiveness of our proposed architecture. We also implemented AES block cipher in behavioral Verilog HDL and synthesized our design for obtaining the timing overheads of the encryption and decryption. In addition, we compare to CHTree proposed by researchers from MIT to understand the pros and cons of our architecture. First of all, our performance results clearly showed the performance advantages of employing the M-TREE. Compared to a processor without any security protection, our scheme only degrades performance by 8% on average while the prior CHTree method degrades performance by almost 50% on average. When the size of the L2 cache is increased, as expected, both schemes will close the gap of performance degradation, with 35% for the CHTree and 5% for the M-TREE.

We then studied the sensitivity of the size of the RM/MAC and MAC cache and the sensitivity to the latency of the hash function. First, we found we will suffer approximately 17% performance loss if we did not employ a separate cache but using the existing L2 to keep these integrity related secret information. By adding an 8KB RM/MAC and MAC cache, the performance degradation is shrunk to 8% on average, making our proposed scheme very appealing and practical. With respect to the hash latency, we did not find the latency increase from 80ns to 160ns imposes any major performance issue in both our scheme and the CHTree scheme in general. For more research results, please refer to our publication in Journal of Parallel and Distributed Computing [2].

2.5 Authentication Point Control in Secure Processors

2.5.1 Problem Statement and Approaches

As mentioned earlier in Section 2.4.1, there are two important properties that a secret needs to be guarded against — confidentiality and integrity. M-TREE and many other prior secure processor designs were targeted to provide both guarantees to achieve a tamper-resistant and tamper-evident computing environment. Encryption is typically used to protect the confidentiality of secret while authentication (e.g. our MACTree) was used to detect and manifest any tampering of the protected data during their residence outside the protection boundary. At the first glimpse, the task of integrating a cryptographic engine into an out-of-order high performance processor for security support may seem deceptively straightforward. Nevertheless, we investigated such a *straightforward assumption* and realized that many prior designs and their security implications have not been fully understood. In particular, the role of integrity protection and its relationship with privacy protection in the context of secure processor was not sufficiently addressed. These prior performance techniques often assume that it is secure to disassociate the decryption operation and the authentication operation. For example, they allow the issue of decrypted instructions prior to the completion of verifying their authenticity.

To understand the potential security implications, in particular caused by side-channel attacks using memory fetch address as a side-channel, we explored and scrutinized the design space of such decryption and authentication dissociation. We classified, investigated, and evaluated five different design alternatives: (1) authen-then-issue, (2) authen-then-commit, (3) authen-then-write, (4) authen-then-fetch, and (5) address obfuscation plus authen-then-commit. Under authen-then-issue policy, a conservative secure execution model, a secure processor is forbidden to issue instructions or any operand whose integrity has not been fully verified. Under the authen-then-commit policy, a secure processor is allowed to speculatively issue unauthenti-

cated instructions and data to the pipeline and commits (or retires) completed instructions only after both of them are authenticated. The option of authen-then-fetch allows bus cycles to be granted to a memory fetch only if all the instructions and data that the memory fetch depends on due to dependency are authenticated. For the authen-then-write, all permanent changes to the memory state must be made based on the results derived from the authenticated instructions and operands. Finally, we examined address obfuscation and its relation to these four options.

In addition to these studies, we also identified a few interesting yet realistic threat models that force the unauthenticated data as a side-channel through memory fetch to disclose the secret. First, the adversary may apply *pointer conversion exploit* which converts encrypted sensitive data into pointers such that its value will be automatically disclosed when the pointers are de-referenced. One such example is *linked list attack*. An adversary may use input manipulation or control flow reconstruction based on fetch trace to either force a linked list to end at some known location or discover when and where a linked list terminates. The adversary can alter the NULL pointer to point to the secret. When the linked list is traversed, the secret will automatically be revealed as a fetch address. More details and the threat models were documented in our publication [5] in MICRO-39.

More in-depth analysis on our original hardware obfuscation technique proposed in [14] for preventing side-channel attacks by exploiting control flow was performed along with this work. This new scheme published in [1, 13] addresses the issue of excessive memory accesses per address permutation and redundant permutations performed. It also avoids the large number of page faults. Further details can be found in our publication [1, 13] in PACT-15 and JPDC.

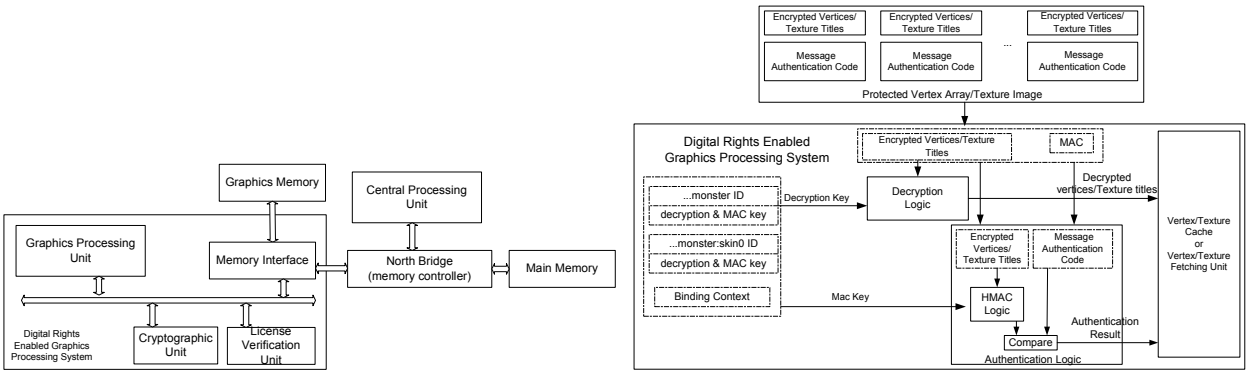
2.5.2 Research Findings

We followed the same simulation framework we developed for M-TREE to carry out our performance evaluation. We evaluated the performance and security implications for these five speculative execution models discussed above. Based on the instruction per cycle (IPC) evaluation, we found the authen-then-issue, the most conservative approach, and the authen-then-commit + address obfuscation demonstrated the worst performance. They achieved only 87% and 86% IPC of a baseline implementing decryption but without integrity checking mechanism. In contrast, authen-then-write shows the best performance at around 98% performance of the baseline processor model, followed by authen-then-commit which shows a performance of 96% of the baseline. The authen-then-fetch achieves about 92% IPC of the baseline. However, from security perspective, authen-then-issue and authen-then-commit + address obfuscation are relatively more secure while authen-then-commit and authen-then-write are less secure. Obviously, there are trade-offs to be made between performance and security. The analysis and the results of this work provides valuable risk assessment for guiding the design of a tamper-proof secure processor. Please refer to our publication in [5] for further details.

2.6 Digital Rights Management

2.6.1 Problem Statement and Approaches

Another growing interest of security issues is with respect to how to protect digital rights. The industry of real-time graphics applications such as video games, interactive avatars, 3D online games, handheld mobile games, etc. grows rapidly. However, it remains a great technological and legal challenge to enforce digital rights protection for these graphics applications. The problem is getting worse with the emergence of 3D graphics commerce on the Internet. In the virtual space, these trade-marked and proprietary 3D models or textures in the forms of digitized sculptures, characters, avatars, vehicles, weapons, outfits, wallpapers, etc., possess real monetary values to gamers, collectors and artists. There is little research being performed in the area of protecting digital rights of graphics data and 3D objects for the consumer market. In this work, we explored the technologies of digital rights management for graphics processing unit (GPU) for countering piracy of real-time graphics entertainment software and graphics assets.



(a) Digital Rights Enabled GPU (b) Processing Encrypted Vertices

Figure 2: DRMed GPU

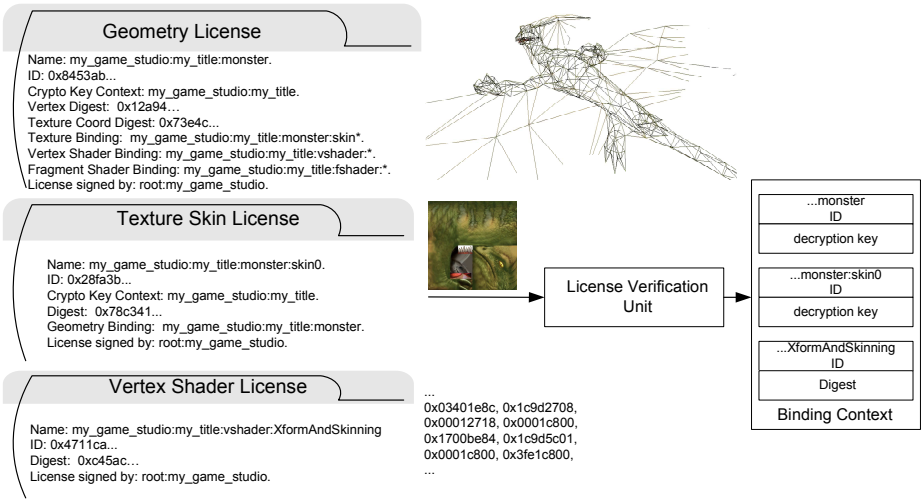


Figure 3: Graphics Asset Licenses and Binding Context

Compared to a conventional GPU architecture, a digital rights enabled GPU contains at least two more components: a *cryptographic unit* to decrypt protected graphics contents, and a *license verification unit* to process the licenses of graphics data. Figure 2(a) illustrates the concept of a digital rights enabled GPU. Similar to digital rights licenses used in other content protection cases, the graphics digital rights licenses released by their content providers specify and designate the desired usage of the graphics data. A digital rights enabled GPU features the necessary means to authenticate the licenses. During the actual graphics rendering, it is guaranteed that the graphics data be used strictly in compliance with the license agreement. Figure 2(b) shows the components and steps for decrypting or authenticating protected geometry data during graphics processing. Each individual vertex and its attributes are separately encrypted. A MAC is stored alongside with the vertex attribute for integrity verification.

In addition to the hardware support, we also introduced the idea of restrictive binding in-between geometry input, textures and shaders. An example of a license of a geometry object is given in the left-hand side of Figure 3. It comprises of a name, decryption key context, digests of the encrypted geometry data, a geometry data ID, the binding constraints as to what textures or shaders can be applied to this object, and the digest of its license signed by a certified content provider. The binding permission can be inherited by all the sub-classes of this object. Given a set of licensed geometry input, textures and shaders, the digital rights enabled graphics processing system creates a binding context that comprises decryption keys that will

be used during graphics processing for decrypting protected contents such as geometry models or textures as shown at the right-hand side of this figure.

2.6.2 Research Findings

In this work, we realized that to provide a digital rights enabled GPU, one needs to coordinate the design effort among the hardware support from processor architects, graphics APIs from programming language designers, and license context from content providers. To quantify the performance, we used a cycle-time based GPU architecture model called *Qsilver* which captures OpenGL commands and data traces and models the flow of data and computation through each state of a generic GPU pipeline including vertex processing, rasterization, fragment processing, frame buffer and z-buffer. The necessary components to enable digital rights management were instrumented into Qsilver. In addition, we implemented the AES crypto-standard and SHA-256 hash function in Verilog and obtained their respective synthesized timing information. The open source Quake 3 Arena was used as our evaluation workload. All the geometry data such as skinned characters and mesh objects and textures, mipmap textures for level-of-details were considered protected. Depending on the cipher chosen, the frame rate will be reduced from less than 5% to more than 30%. More details of our complete proposed mechanisms including hardware and API enhancement, context binding protocols, etc., can be found in our paper published in ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware [9].

2.7 Security Analysis for Hardware-based Protection

One major goal in this aspect is to enable our system with the capability of autonomic monitoring. The broader definition of monitoring for the healthiness of a system includes characterizing the following: system workload characteristics, memory access patterns, and detecting any abnormal application/system behavior and security-related intrusions. Prior to delving into our solution space, we studied the pitfalls of the existing hardware-based solutions which were often aimed at addressing software copyright protection. Several tamper-resistant processors were proposed and studied in literature to prevent illegal software duplication, unauthorized software modification, and unauthorized software reverse engineering. We found these prior techniques all focused on feasibility and design details but often neglected the analysis of other security implications and risks, worse, potential attacks from an adversarys perspective. Toward this, we investigated and performed detailed analysis for potential vulnerability, weakness, pitfalls and attacks on such security architectures. The objective is to provide useful and valuable insights to the system designers.

In our risk analysis, we examined a few areas in the current security architecture designs and identified their weakness as well their potential vulnerability. They include *lazy authentication, regularity vulnerability in encrypted RISC instructions, information control flow leakage, software slice based protection, weak cipher and short encryption key, limited integrity protection,* etc. Our concern came from the recent emerging side-channel attack models, which bypass the strength of cryptography and exploit the inadvertently information leaked from so called side channels, including memory addresses, execution time measurement, power signature, thermal gradient, electromagnetic waves, etc. Our studies presented the arguments for the necessities of hardware-based cryptography, a detailed analysis and understanding of different, innovative, security break models, a taxonomy of online and offline attack models, the limitation of a single-processor protection model and per-process protection model, along with new insights for a lower cost implementation of memory authentication code (MAC). With the thorough understanding of the vulnerability of current systems from our studies, system designers will be able to explore the design space more effectively and efficiently in facilitating security monitoring detection and protection. Our research findings were reported in [5].

2.8 Acceleration of Memory Encryption and Authentication

One essential component shared by all the hardware-based tamper resistant systems is hardware protection of software confidentiality and integrity. When a memory block containing data or instruction (e.g., a cache

line) is brought into the secure processor, it is decrypted and verified. When a cache line is evicted from a secure processor domain, it is re-encrypted prior to being stored to any external memory. This hardware cryptography is not only necessary for making system difficult to reverse engineer but is also a critical component for content-based digital rights protection. For protecting randomly accessed memory, standard encryption modes that directly encrypt memory such as block cipher including ECB, CBC, or OCB, or counter-mode encryption have been employed in secure architectures proposed by academia as well as in commercial security co-processor. In addition, authentication mechanism using message authentication code (MAC) was also proposed to provide integrity guarantee.

From processor architects design perspective, the major overhead of such hardware protection on software confidentiality and integrity is the increased memory latency. Both encryption and authentication need to go through a series of repetitive data scrambling process. Much of our effort toward this crypto-design is to minimize or completely hide (in the ideal case) the latency of such operations, thus making the entire protection transparent to the users. Based on our prior experiences and new observations, we proposed a novel architectural framework to enable cipher-text speculation, a latency hiding mechanism that combines value prediction and hardware cryptography for direct encrypted memory using block cipher. This technique, frequent value cipher speculation reduces or eliminates the decryption latency by speculatively encryption frequent values and matching their ciphertext results with the one fetched. In other words, we use the wait time for fetched data line to perform speculative encryption. In our findings, 40% of the fetched data contained frequent values. In this framework, we also introduced a new scheme to accelerate memory integrity verification by pre-computing of memory blocks MAC value based on frequent value prediction. This technique, MAC speculation, pre-computes MAC with the corresponding MAC being fetched from main memory. By combining these techniques in a secure processor, we show significant performance improvement in instruction per cycle (IPC) ranging from 10 to 30%. This work was published in [6].

2.9 Denial-of-Service Attack on Multicore Processors

Given the continuing miniaturization trend predicted by Moores Law and physical constraint in design verification and frequency scaling, chip multiprocessor or multicore processor has become the de facto architecture standard for all processors that will be delivered from now on. Through resource sharing, applications running on a multicore processor can achieve better resource utilization and faster inter-core communication, leading to higher overall throughput for the entire system. From a security perspective, however, such architectures are also more susceptible to a new kind of attack — "Denial-Of-Service (DoS)" attack to their common resources. Furthermore, as the number of cores increases, attacks similar to Distributed DoS (DDoS) attacks over the Internet can be employed to throttle these on-chip resources with the presence of multiple, cooperative malicious applications. Such DoS or DDoS attacks aim at reducing performance or barricading the entire system from making any forward progress.

In this work, different from all prior architecture work, we try to understand such implication from security standpoint. We focus on the vulnerability of shared last-level cache (LLC). The likely attack models can be combined with other techniques such as rootkits (See Section 2.4). An unwanted malicious process can run in stealth to consume either the capacity of the shared LLC, or the shared bandwidth on a shared-bus based multi-core system. As we demonstrated, to implement such a malicious program is almost a no-brainer. All the attackers need to do is to have a program that marches memory locations with a large, fixed stride. Besides cooking up the malicious attacking codes, we also proposed a few microarchitectural and OS-level solutions attempting to minimize the impact of such microarchitectural DoS or DDoS attacks. This ensembles much similarity to other work on so-called fairness issues in a multicore processor, in other words, to avoid monopoly of certain particular processes, if the system cannot differentiate the good processes from the bad. One option is to implement monitoring and throttling scheme for those processes that dominate the use of shared resources (i.e., to be fair for all processing cores). Another option is to segregate these resource-heavy consumer processes, or simply let the bad guys rot in hell. The idea of this scheme is to

let all resource-demanding processes to compete pre-allocated shared resources and allow the other regular processors to use a reasonable amount of resources pre-allocated. Our research was published in [12].

3 Scholarly Contributions

The project activities undertaken resulted in several high-quality publications in premium conferences. These publications are listed in the *References* section at the end of this final report. We also accumulated our expertise in multi-domained tools and established our own simulation methodology to continue our future research endeavor. To summarize, we have made the following research contributions in this project over the years.

- We researched and presented a thorough security analysis and risk assessment for a security computing system built on tamper-resistant and tamper-evident processors.
- We contrived and detailed several novel and realistic attack models to these security processors. These models provide invaluable insights to the requirement of future security architecture design.
- We proposed a memory-centric security architecture or MESA to enable information sharing and exchange with their individual security requirement and intellectual proprietary protection.
- We proposed an M-TREE security architecture with a novel hierarchical message authentication tree for protecting applications' integrity with minimal performance overhead.
- We investigated microarchitectural techniques to hide the performance penalty caused by memory encryption and authentication, providing a performance-transparent tamper-resistant and tamper-evident system.
- Information leakage issues in a protected system for confidentiality and integrity was defined and classified. Attack models of exploiting information leakage to reveal sensitive data was discussed. We then proposed hardware-based dynamic obfuscation techniques to address these security issues.
- We studied the emerging issues of digital rights management for virtual properties, a burgeoning business in the Internet game sector. We proposed hardware enhancement to effectively address the ownership issue for proprietary virtual assets.
- We identified and studied the often-overlooked security issue in previously proposed authentication mechanisms which leave unnecessary side channels open subject to exploits.
- We studied the architectural support of an introspective architecture with high-speed self-recovery mechanism based on the emerging multicore processor systems.

3.1 Human Resources

Throughout the project period, we had successfully trained and awarded graduate degrees to the following students who at certain point were sponsored by this Department of Energy Career Award.

- **Weidong Shi, Ph.D. 2006.** First employer: Motorola Labs, Motorola, Inc., Schlumberger, Illinois. Current position: Entrepreneur of a startup company at Canada.
- **Mrinmoy Ghosh, Ph.D. 2009.** First employer: Corporate Research and Development, ARM, Austin, Texas.

References

[1] Lan Gao, Jun Yang, Marek Chrobak, Youtao Zhang, San Nguyen, and Hsien-Hsin S. Lee. A Low-cost Memory Remapping Scheme for Address Bus Protection. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT-15)*, pages 74–83, September 2006.

[2] Chenghuai Lu, Tao Zhang, Weidong Shi, and Hsien-Hsin S. Lee. M-TREE: A High Efficiency Security Architecture for Protecting Integrity and Privacy of Software. *Journal of Parallel and Distributed Computing*, 66(9):1116–1128, 2006.

[3] Martin Schulz, Brian S. White, Sally A. McKee, Hsien-Hsin S. Lee, and Jurgen Jeitner. Owl: Next Generation System Monitoring. In *Proceedings of the ACM Computing Frontiers 2005 (CF'05)*, pages 116–124, Ischia, Italy, May 2005.

[4] Weidong Shi, Joshua B. Fryman, Guofei Gu, Hsien-Hsin S. Lee, Youtao Zhang, and Jun Yang. InfoShield: A Security Architecture for Protecting Information Usage in Memory. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture (HPCA-12)*, pages 255–234, Austin, Texas, February 2006 (acceptance rate = 14%).

[5] Weidong Shi and Hsien-Hsin S. Lee. Authentication Control Point and Its Implication for Secure Processor Design. In *Proceedings of the 39th International Sympsoium on Microarchitecture (MICRO-39)*, pages 103–112, December 2006.

[6] Weidong Shi and Hsien-Hsin S. Lee. Accelerating Memory Decryption and Authentication with Frequent Value Prediction. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 35–46, 2007.

[7] Weidong Shi, Hsien-Hsin S. Lee, Laura Falk, and Mrinmoy Ghosh. An Integrated Framework for Dependable and Revivable Architecture Using Multicore Processors. In *Proceedings of the 33rd International Symposium on Computer Architecture*, pages 102–113, June 2006.

[8] Weidong Shi, Hsien-Hsin S. Lee, Guofei Gu, Laura Falk, Trevor N. Mudge, and Mrinmoy Ghosh. An Intrusion-Tolerant and Self-Recoverable Network Service System Using a Security-Enhanced Chip Multiprocessor. In *Proceedings of the Internatioanl Conference on Autonomic Computing (ICAC-05)*, pages 263–273, Seattle, Washington, June 2005.

[9] Weidong Shi, Hsien-Hsin S. Lee, Richard M. Yoo, and Alexandra Boldyreva. A Low-cost Memory Remapping Scheme for Address Bus Protection. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop of Graphics Hardware*, pages 17–26, September 2006.

[10] Weidong Shi, Chenghuai Lu, and Hsien-Hsin S. Lee. Memory-centric Security Architecture. In *Proceedings of the 2005 International Conference on High Performance Embedded Architecture and Compilers (HiPEAC)*, pages 153–168, Barcelona, Spain, November 2005 (acceptance rate = 20.2%).

[11] Taeweon Suh, Hsien-Hsin S. Lee, Sally A. McKee, and Martin Schulz. Evaluating System-wide Monitoring Capsule Design using Xilinx Virtex-II Pro FPGA. In *Workshop on Architecture Research using FPGA Platforms (WARFP) in conjunction with International Symposium on High-Performance Computer Architecture (HPCA-11)*, San Francisco, CA, February 2005.

[12] Dong Hyuk Woo and Hsien-Hsin S. Lee. Analyzing Performance Vulnerability due to Resource Denial-of-Service Attack on Chip Multiprocessors. In *Workshop on Chip Multiprocessor Memory Systems and*

Interconnects (CMPMSI) in conjunction with the 13th International Conference on High-Performance Computer Architecture (HPCA-13), 2007.

[13] Jun Yang, Lan Gao, Youtao Zhang, Marek Chrobak, and Hsien-Hsin S. Lee. A Low-Cost Memory Remapping Scheme for Address Bus Protection. *Journal of Parallel and Distributed Computing*, 70(5):443–457, 2010.

[14] Xiaotong Zhuang, Tao Zhang, Hsien-Hsin S. Lee, and Santosh Pande. Hardware Assisted Control Flow Obfuscation for Embedded Processors. In *Proceedings of the International Conference on Compilers, Architecture, Synthesis for Embedded Systems*, pages 292–302, Washington D.C., 2004.