SecCMP:
Enhancing Critical Secrets Protection in Chip-Multiprocessors

Li Yang, University of Tennessee at Chattanooga, USA

Lu Peng, Louisiana State University, USA

Balachandran Ramadass, Louisiana State University, USA

ABSTRACT

Security has been considered as an important issue in processor design. Most of the existing designs of security handling assume the chip as a single secure unit. However, such assumption is vulnerable to exposure resulted from a central failure point. In this article, we propose a secure Chip-Multiprocessor architecture (SecCMP) to handle security related problems such as key protection and core authentication in multi-core systems. Matching the nature of multi-core systems, a distributed threshold secret sharing scheme is employed to protect critical secrets. A critical secret (e.g., encryption key) is divided into multiple shares and distributed among multiple cores instead of being kept a single copy in one core that is sensitive to exposure. The proposed SecCMP can not only enhance the security and fault-tolerance in secret protection but also support core authentication. SecCMP is designed to be an efficient and secure architecture for CMPs.

Keywords: Cybernetics; Data Sharing; Front-End Computers; Processor Architecture

INTRODUCTION

Computer networking makes every computer component vulnerable to security attacks. Examples of such attacks include injection of malicious codes (e.g., buffer overflow), denial of service (DoS) attacks, and passive eavesdropping between CPU cores and off-chip devices. Also off-chip or on-chip devices taken over by an adversary can launch attacks to other components of a computer. Pure software solutions itself cannot counter all attacks, therefore, enforcing security in processor design has drawn more and more attention. Currently many proposed works focus on encryption and authentication of hardware memory in single-core systems (Gassend, Suh, Clarke, Dijk, & Devadas, 2003; Lee, Kwan, McGregor, Dwoskin, & Wang, 2005; Shi, Lee, Ghosh, Lu, & Boldyreva, 2005; Yan, Rogers, Englender, Solihin, & Prvulovic, 2006; Yang,

Zhang, & Gao, 2003). They usually assume the processor core as a safe and secure unit. When Chip-Multiprocessors (CMPs) have become mainstream products, applying encryption scheme of existing works to each core independently is one possible solution to enforce security in CMPs. The weakness of this solution is that the critical secrets (e.g., encryption key) stored or processed by one processor core can be easily exposed to adversaries through remote exploit attacks such as buffer overflow or Trojan horse, which leads to a central failure point. Once a core is compromised or taken over, the adversary could either access the critical secrets or wait until the compromised thread migrating onto another clean core then access unauthorized critical secrets. Therefore, this is not an effective approach to protecting shared critical secrets for CMPs.

Utilizing the distributed nature of CMPs is an alternative solution to reinforce the security of CMPs. Not only the computation load but also the security risks are distributed among multiple processor cores that are designed to collaboratively protect and access the critical secret. No individual core is possible to access the critical secret alone. We proposed a novel *Secure Chip-Multiprocessor (SecCMP)* architecture (Yang & Peng, 2006) to protect critical secrets based on a distributed *Secret Sharing* 9. Instead of protecting a secret in one processor core, *Secret Sharing* is employed to distribute the secret among multiple cores that protect the secret collaboratively. The distributed security management matches the nature of multi-core architecture in CMPs. By employing a threshold *Secret Sharing* scheme, critical secrets are protected safely in a CMP processor even when one or more processor cores are compromised. In this article, we integrate the SecCMP architecture with identity-based cryptography to support remote information access and sharing. The performance degradation of our approach is studied through simulation. Low overheads and improved fault-tolerance are two major features of our approach. Low overhead is achieved via distributing the encryption and decryption load among multiple cores. Fault-tolerant is achieved via *(k, n)* secret sharing where at least k out of n cores are required to recover the secret. From a secret protection point of view, fewer than *k-1* cores are not able to recover the secret (i.e., the encryption key) such that our solution is resistant to the compromise of fewer than *k-1* cores. From a service protection point of view, k cores are able to provide the secret recovery service (i.e., retrieve the encryption key) such that our solution is tolerant to failure (i.e., hardware failure, DoS attacks) of up to *(n-k)* cores. Moreover, confidentiality and authentication among cores are supported through core authentication in *SecCMP*. Core authentication, which identifies whether a core is compromised, could be performed during critical information reconstruction or periodically. If not enough authenticated cores available, a system error will be called. The user may restart the system and reconstruct the critical secrets.

We use an application to demonstrate secure and remote critical information access and sharing supported by our *SecCMP*. Integrated with identity-based cryptography (Bonh, & Franklin, 2003) the *SecCMP* provides a secure and reliable way to generate and distribute encryption keys between local host and remote site when prior distribution of keys is not available. Each local host has a pair of *master public key* (*MUK*) and *master private key* (*MRK*). In addition, each account has a pair of *account public key* (*AUK*) and *account private key* (*ARK*). In the local host which contains a multi-core processor, the MRK is divided and distributed among multiple cores and the ARK is generated from the MRK. On the remote site, the MUK and an Account ID will generate an AUK, which is used to encrypt the requested critical information. After receiving the encrypted critical information, k authenticated cores in the local host involve in generating the ARK, which finally decrypts received information.

To support critical information protection on CMPs, each processor core maintains two registers for the secret share and a public/private key pair for core authentication. These registers can only be accessed by a trusted application

which constructs an account private key and decrypts the information. An encrypted I/O channel is employed to support user input and critical information receiving. To avoid bus or interconnection eavesdropping, all critical information related cache blocks are encrypted.

The rest of this article is organized as follows: Introduction of related work; statement of the attack and thread model of this article; discussion of the *master private key* protection and core authentication of the *SecCMP* architecture; an application of the *SecCMP* is to support critical information remote access and sharing followed by security and computational complexity analysis; introduction of performance evaluation; and, finally, we summarize the article.

RELATED WORK

Lee et al. (Lee et al., 2005) proposes a *"secret-protected (SP)"* architecture focusing on key protection and management, featured by secure on-line access of users' keys from different network computing devices. The keys are organized as a tree structured key chain rooted at a secret *"User Master Key"*. With helps from additional hardware features supporting *Concealed Execution Mode (CEM)* and *Trusted Software Module (TSM)*, the *SP* architecture protects confidentiality and integrity of sensitive data transmitted between processor chip and off-chip devices. Our proposed mechanism can enhance the security for the SP processor architecture working on a CMP. With a threshold distributed secret sharing, even if one or more pieces of critical secrets are released, the adversaries still cannot obtain the secrets as long as the number of released pieces is less than the threshold.

In Shi et al. (Shi, Lee, Falk, & Ghosh, 2006) the authors present an integrated framework utilizing multi-core processors to detect intrusions and recover from infected states. The processor cores are divided as resurrectors and resurrectees and memory space is also insulated. Resurrectees cannot access resurrectors' memory but resurrectors can access all the memory space. Fine grain internal state logging for low privileged cores, resurrectees, is employed. Resurrectors dynamically check the states of resurrectees. If any suspicious intrusions are detected, a logged state will be recovered. This design presumes that there are one or more master cores which are immune to attacks. In our scheme, we assume that all cores inside a chip are organized to a peer-to-peer relationship. Any cores could be compromised. However, if there are not enough authenticated cores, the system can be recovered by restarting.

There are two schemes to protect memory integrity and confidentiality for symmetric shared memory multiprocessor systems (SMP) proposed in (Shi, Lee, Ghosh, & Lu, 2004; Zhang, Gao, Yang, Zhang, & Gupta, 2005). In Shi et al. (Shi et al., 2004), the authors propose a one-time-pad based memory encryption scheme and an SHA256 hash function based authentication approach to protect bus communication. The scheme proposed in (Zhang et al., 2005) further improves security by generating a Cipher Block Chaining (CBC) encryption pad from snooped data. In (Rogers, & Solihin, 2006) the authors propose a memory encryption and authentication mechanism for Distributed Shared Memory (DSM) systems. All above proposals assume that a processor is a single secure unit. In this article, we assume that adversaries could intrude one or more cores in a multi-core processor and they need more efforts and time to intrude more than one cores than they compromise a single core.

Many other works (Gassend et al., 2003; Lee et al., 2005; Shi et al., 2005; Yan et al., 2006; Yang et al., 2003) emphasize on memory encryption and authentication by efficient hardware approaches in single-core systems. Our proposed scheme focuses on on-chip secret protection in multi-core processors. Additionally, there are a few proposals provide efficient bus or interconnection protection (Gao, Yang, Chrobak, Zhang, Nguyen, & Lee, 2006; Zhuang, Zhang, & Pande, 2004). Incorporating with the above memory/bus protection

and recovery schemes, our proposal will further enhance security and fault-tolerance of CMP systems.

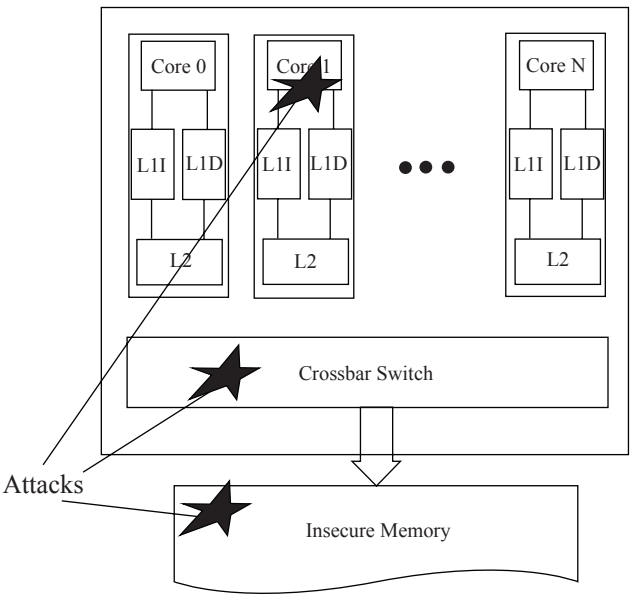
ATTACK MODEL

In this article, we focus on vulnerabilities resulted from the remote exploit network attacks. The system suffers attacks at different points including memories, crossbar switch or one or more cores on multi-core processors, as shown in Figure 1. A firewall or intrusion detection system (IDSs) helps to protect users from attacks such as known virus or denial-of-service. However, they cannot protect the system once attacks have bypassed firewalls or IDSs. Therefore, the system suffers from a variety of vulnerabilities and attacks that expose critical secrets of a core to adversaries. The different kinds of attacks include eavesdropping, buffer overflow, cascading breakdowns, message spoofing, and message blocking.

The adversaries can access critical secrets by *eavesdropping* the crossbar switch connecting cores. During an eavesdropping attack, an attacker tries to learn critical secrets such as the root password that he/she was unable to access. *Buffer overflow* attack happens when a process attempts to store data beyond the boundaries of a fixed length buffer, and the extra data overwrites adjacent memory locations. By doing so, possible malicious code is injected into an execution path. If executed, the injected malicious code grants attackers unauthorized privileges to access critical secrets. Therefore, both eavesdrops and buffer overflow try to increase privileges of an attacker and expose the critical information to unauthorized attackers. Adversaries can break down other processors in a cascading manner if one core's critical secret is compromised. *Message spoofing* can occur if a fake message is generated and attributed to other senders. Examples are message insertion or replaying. Also, a message destined to a processor can be *blocked illegally* when it is transmitted through the crossbar switch.

A single authentication or encryption algorithm is not able to counter all aforementioned attacks. The section Secure Chip Multiprocessor Architecture, describes how SecCMP enhances confidentiality by distributing secrets among

Figure 1. Attack model



multi-cores and augments authentication by employing digital signature from each core. The section *SeeCMP* Supported Clinical Inofrmation Access and Sharing, demonstrates countermeasures proposed in SecCMP by using an example. The example shows how to support access of the critical information shared remotely

SECURE CHIP MULTIPROCESSOR ARCHITECTURE

Each processor has a pair of *master public key* and *master private key*. The *master public key* is available to all other network devices and hosts. The *master private key* is protected by a threshold secret sharing scheme in the *SecCMP*. Protection of the *master private key* allows critical information related to an application to be stored and accessed over public network. Each core in the processor also employs new hardware features to support trusted applications. Trusted applications generate and distribute application related keys and encrypt, decrypt the remotely shared critical information. Because all activities including key computations, distributions and critical information encryptions and communications are protected by the trusted applications, an adversary can not observe other cores' secret share even when one or more cores are compromised. Most importantly, the *SecCMP* comprises two unique components: protection of a master (chip) private key and core authentication.

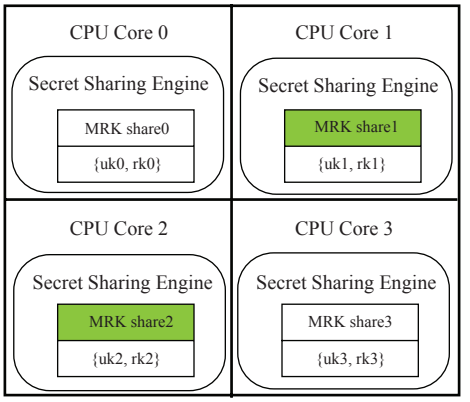
Protection of the Master Private Key

The *master private key* is used to generate an application private key (we call it as an *account private key* in the section *SeeCMP* Supported Clinical Inofrmation Access and Sharing and decrypt the application related critical information stored on-line and accessed over public network, that is, banking PIN, PGP keys. To avoid a central failure point in a single processor system, the *master private key* is divided and distributed among multiple CPU cores in a processor chip. In a (*k, n*)-threshold secret sharing, each core c_i holds a secret share s_i, and any k of these n cores can reconstruct the *master private key*. Any collection of less than k partial shares cannot get any information about the *master private key* (Pederson, 1992). Here, k is the threshold parameter such that $1 \leq k \leq n$. Each processor will authenticate itself in fine grained intervals. Therefore, it is difficult for adversaries to obtain k or more pieces of secret shares during a short time.

Figure 2 is an example of a (2, 4)-threshold scheme among four cores where a *master private key MRK* is divided into four unique pieces (Secret Share 0, 1, 2, 3), such that any two of them can be used to reconstruct the *master private key MRK*. Traditional Shamir's secret sharing scheme suffers from the requirement of a trust authority and the absence of share verification. We employ a scheme based on nine, which is an extension to Shamir's secret sharing without the support of a trust authority. We also deploy the verifiable secret sharing (Pederson, 1992) to detect the invalid share that some shareholders generate to prevent reconstruction of the *master private key*.

For each core in a processor supporting SecCMP, we have a 128 bit register for the secret share and a 128 bit register for the public/private key pair which is used for core authentication. These dedicated registers can only be accessed by the trusted application which generates the account private key. To ensure security, a thread running on one core cannot access another core's secret share register and public/private key register. During the procedure of critical information generation, threads involved are not allowed to migrate to another core. Besides, an encrypted I/O channel is required for a user to input the account ID (AcctId) and receive the decrypted critical information. To keep confidentiality of the decrypted critical information, we employ an AES based encryption (Menezes, Oorschot, & Vanstone, 1997) scheme to encrypt a related cache block before it goes outside of the core. This can prevent eavesdropping attacks over the crossbar interconnection.

Figure 2. A secure chip multiprocessor (seccmp) with master private key MRK sharings



For performance consideration, however, other cache blocks are not necessary to be encrypted. To identify critical information related cache blocks, we add one bit into all on-chip cache tags. This bit is set if a cache block contains part critical information.

Core Authentication

One or more cores may be compromised and their secret share is exposed to an attacker. We assume that an honest core will present the correct share to authenticate itself, and a compromised core will present a random number instead of the correct share. The attacker learns the secret shares from compromised cores and interrupts the *master private key* reconstruction. Failure of *master private key* reconstruction will result in a denial of service (DoS) attack. In order to exchange secret share securely, each core c_i holds a public/private key pair $\{<uk_i, rk_i>\}$ to encrypt the secret share and authenticate each other. Each core signs its secret share and hash code with its private key (digital signature). Then the signed message is encrypted with requesting core's public key. The requesting core decrypts the message with its private key. Then requesting node checks the signature to authenticate the sender, checks the hash code to make sure the integrity of the secret share. The key pair is created during core installation based on each core's identity. An adversary is not able to observe the encrypted share without a correct key pair. The private key is also used to encrypt the critical key sent to and from off-chip devices. In our design, core authentication also allows that a sender core to authenticate a receiver core. This is implemented by sending an authentication request message to the receiver core and checking the returned signature.

We not only passively protect the *master private key*, but also actively detect the compromised core. In order to detect the compromised cores, we design a series of *m master private key*s such that none of the participants knows beforehand which is correct. The *master private key*s are ordered incrementally based on their values, except for the real key. The participants combine their shares to generate one key after the other, until they create a correct key that is less than the previous key. This helps us to detect the compromised core before the master private key is exposed to the cheating compromised core. The detection and prevention of cheaters in threshold schemes (Lin, & Harn, 1991) is adopted in our approach. Once the compromised core is detected, we isolate the compromised one. The work in (Martin, 1993) allows a new sharing scheme to be activated instantly once one of the cores becomes untrustworthy.

SECCMP SUPPORTED CRITICAL INFORMATION ACCESS AND SHARING

An application of *SecCMP* is to support critical information remote access and sharing. *SecCMP* provides secure channels to generate, store and exchange encryption keys for a local host and remote sites to share critical information associated with a specific account (i.e., a bank account, an email account). Each local host has a pair of *master public key* (*MUK*) and *master private key* (*MRK*). In addition, each account has a pair of *account public key* (*AUK*) and *account private key* (*ARK*). Based on the identity-based cryptography (Bonh, & Franklin, 2003) a user *account public key* can be any arbitrary string. In other words, users may use some well-known information, such as email address, IP address, URL as their *account public key*.

When a local host tries to retrieve critical information from a remote site, it creates a pair of MUK and MRK. The MUK is available to the remote site, and the MRK is distributed and stored in multiple cores of the local host. Such distributed design of the *master private key* is resistant to eavesdropping since at least k cores need to be compromised in the active session to reconstruct *master private key*. Moreover, *k* out of *n* cores needs to be contacted in order to create an *account private key* based on the account ID. A buffer overflow attack may expose secret share of a core or interrupt private key generation from a core. Our core authentication service could detect such attacks. Because an attack from network exploits usually cannot be performed in a very short duration, the undergoing attack can be reported and blocked before *k* cores are compromised.

Identity-based Cryptography

Identity-based systems 1 allow any party to generate a public key from a known identity value such as an ASCII string. The Private Key Generator (PKG) generates the corresponding private keys. To operate, the PKG first publishes a *master public key*, and keeps the corresponding *master private key*. Given the *master public key*, a public key can be generated corresponding to the identity by any party. To obtain a corresponding private key, PKG is contacted to generate the private key using *master private key* based on the identity. As a result, messages may be encrypted without prior key distribution between individual participants. Such solution is helpful when the pre-distribution of the authentication keys is not available. A major challenge of this approach is that the PKG must by highly trusted since it generates any user's private key and thus decrypt messages. In the *SecCMP*, multiple cores work together to provide a secure private key generation service when there is no prior distribution of keys.

Remote Information Access and Sharing

When a local host, a multi-core processor system, tries to access or retrieve critical information from a remote site. The local host and remote site need to authenticate and exchange *master public key* with each other. How two remote hosts authenticate each other and how local host authenticate the current user is valid or not are out of the scope of this article. The former can be accomplished either by a Certificate Authority (CA) or a trusted third party. The later can be achieved through access control or biometrics. We focus on how remote site and local host generate, store and distribute *master public key*, *master private key*, *account public key* and *account private key*.

Figure 3 shows the general procedure of remote information access and sharing. To initiate the remote critical information access, the local host sends its *master public key* to remote site in the step 1. In step 2, the local host also sends account ID (*AcctID*) whose critical information the user would like to access. The remote site computes an *account public key* based on a *master public key* and an account ID (*AcctID*). After that, the remote host encrypts requested critical information with the *account public key* and transmits requested critical information to public networks in step 3. In step 4, the local host computes *account private key* by interacting at least *k* out of *n* cores in

Figure 3. Remote critical information access

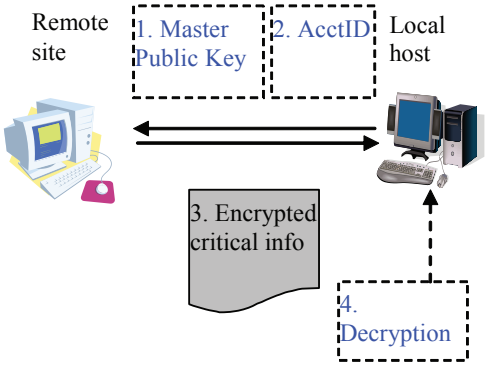
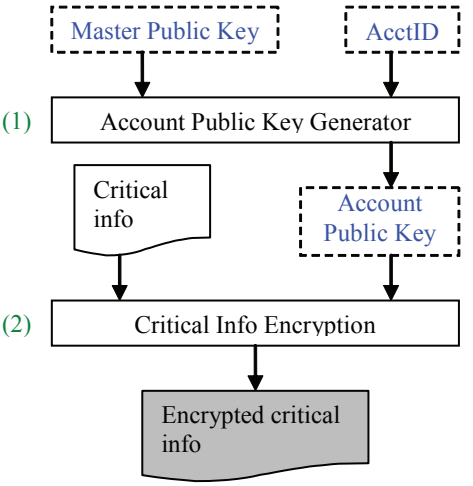


Figure 4 Account public key generation and encryption in the remote site



the local host processor, and uses the *account private key* to decrypt the received encrypted critical information.

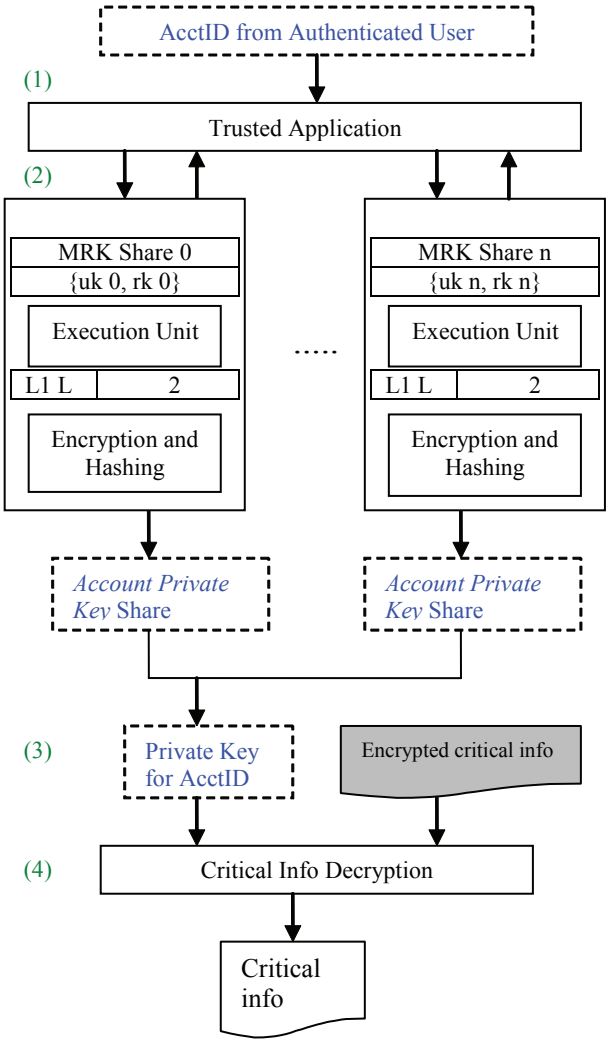
The remote site needs to generate an *account public key* to encrypt requested critical information and send it to public networks. The local host will obtain *account private key* to decrypt the critical information encrypted using its corresponding public key. Figure 4 shows the *account public key* generation and critical information encryption in the remote site. There is a generator producing an account public key from the master public key and the input account id. The critical information is encrypted by the account public key and transmitted back to the requestor over a public network.

The local host generates an *account private key* by which the encrypted critical information is decrypted, as shown in Figure 5. The method to obtain the *account private key* is to contact at least *k* cores, present the account identity and request private key generation service.

The trusted application is the only application that can access the secret share in registers of a core. It can be implemented as a system call which can only be executed by a thread with a special privilege. This application is called when an authenticated local user sends a request for critical information to a remote site. When a trusted application sends the *MUK* to a remote site, the *MRK* was divided into *n* secret shares and kept in *n* cores respectively. The distributed secret shares and account public key pairs are stored in a set of special registers which can only be accessed by the trusted application. The general procedure of the account private key generation and decryption in the local host is listed as follows:

1. Authenticated users input to the trusted application and *AcctID* whose related critical information will be retrieved through public network by an authenticated user.
2. Each core generates an *account private key* share.
3. *K* shares of the *account private key* construct the corresponding *ARK* for *AUK* of the *AcctID*. Note each core only constructs its own share and then sends the output results to the next core. Therefore the confidentiality of the *MRK* is preserved here. Meanwhile, during the transmission, core authentication will also be preformed. To keep the constructed Account Private Key secret, the *kth* core has to be authenticated. Before the *(k-1)th* core

Figure 5. Account private key generation and decryption in the local host



sends out its decrypted secret share, it has to authenticate the last receiver. If no available authenticated cores exist, there are no enough cores to decrypt requested information. It will report this failure to the user. The user may require restart the system. Details of core authentication refer to the Core Authentication section.

4. The constructed *ARK* is used to decrypt the cipher text encrypted by the corresponding *AUK*. We will perform this step on an authenticated core identified by step 3. By doing so, a user can securely access and share critical information remotely. The received critical information can be protected until the user session finished.

SECURITY AND COMPLEXITY ANALYSIS

Confidentiality and *Integrity* are taken care of by the encryption and hash function performed on secret share. Hash function guarantees that a share being transferred is never corrupted due to non-benign failure. *Availability* ensures the survivability of a processor chip despite denial of service attack. In our schema we take care of this problem by making use of *(k, n)* threshold secret sharing algorithm, as any *k* out of *n* cores work together for critical master key reconstruction. Thus our security solution is tolerant to *k-1* compromised cores. *Authentication* is taken care of by digital signature that enables a core to ensure the identity of the peer core it is communicating with. During a network exploit attack, the adversary compromises the processor cores one by one through attacks such as buffer overflow. We authenticate cores during the critical information generation or actively perform authentication periodically. If there are no enough cores for decryption, a system error will be triggered. Therefore, our secret sharing mechanism prevents an adversary from spoofing a secure core and gaining unauthorized secret share. Main computations in our approach come from secret share reconstruction and encryption. The reconstruction *computational complexity* depends on the number of thresholds. The encryption *computational complexity* is same as the traditional schemes and depends on the size of shares. The shorter length of a share results in less resource consumption. The computations will be accelerated by involvement of multi-cores.

PERFORMANCE ANALYSIS

User activities can either not involve the request of remote critical information (e.g., browse unclassified daily news) or involve the request of remote critical information (e.g., access bank account). The former one does not call secret protection procedure so that the system performance is not affected. However, the latter one calls the secret protection procedure demonstrated in Figure 5, resulting in performance degradation. The performance degradation in the local host of Figure 3 is considered and simulated in this article. We evaluate the proposed architecture by a multi-core processor based on SESC (http://sesc.sourceforge.net/) with SPLASH2 applications and kernels. The setting of the processor simulator is listed in Table 1. For each program, we skip the first 100 million instructions and collect statistics for the next 200 million instructions. Three schemes are evaluated: the baseline machine has no secret sharing mechanism; a CMP triggering a secret sharing engineering every 1 million cycles and a CMP triggering a secret sharing engineering every 100 thousand cycles. We measure the average number of cycles executed for all eight cores. We calculate performance degradation by comparing average execution cycles for each scheme. The simulated secret sharing engine is a (4, 8) scheme, which means that a secret can only be reconstructed by at least four correct sharing. During secret generation, the current executed program will be paused and the secret sharing engine will use caches for it calculation. Performance degradation comes from the latency of the secret sharing engine and cache pollution. Figure 6 shows performance degradation for simulated programs. We can see that average performance degradation are 0.26% and 2.35% respectively when the secret sharing scheme is triggered every 1 million cycles and every 100 thousand cycles. Ocean, which is sensitive to cache misses, slows down 9.17% when the secret sharing engine is triggered every 100 thousand cycles.

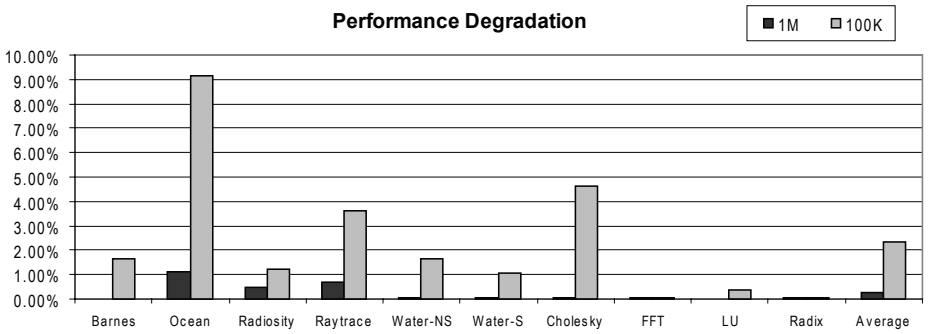
CONCLUSION

In this article, we introduce a low cost secure architecture design for CMPs. The proposed architecture employs a threshold *Secret Sharing* scheme to protect critical secrets and support core authentication for a CMP system. It supports online critical information retrieval and protection in a local host. In stead of keeping a whole copy of a critical secret, the secret is divided and distributed among multiple cores. A user can only reconstruct the secret if the number of authenticated cores is equal to or

Table 1. Processor and memory hierarchy parameters

Processor	8 cores, 5GHz, Out-of-order execution pipeline. Issue width: 1.
L1 instruction cache	Two way set-associative. 32 bytes block size. 32KB inst. cache for each core. 1 cycle access delay.
L1 data. cache	Four way set-associative. 32 bytes block size. 32KB inst. cache for each core. 2 cycles access delay.
L2 shared unified cache	16-way set-associative. 128 bytes block size. 1MB. 9 cycles access delay. MESI protocol for L1 cache coherence.
Memory latency	500 cycles.
Secret Sharing Engine latency	80 cycles.

Figure 6. Performance degradation for a seccmp triggering a secret sharing engine every 1 million and every 100 thousand cycles



larger than the threshold. Compared with existing mechanisms, the proposed scheme is more secure and fault-tolerant.

REFERENCES

Bonh, D., & Franklin, M. (2003). Identity-based encryption from weil pairing. *SIAM Journal of Computing*, *32*(3), (pp. 586-615).

Gassend, B., Suh, G., Clarke, D., Dijk, M., & Devadas, S. (2003). Caches and hash trees for efficient memory integrity verification. *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*, (pp. 295-306).

Gao, L. Yang, J., Chrobak, M., Zhang, Y., Nguyen, S., & Lee, H. H. (2006). A low-cost memory remapping scheme for address bus protection. *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Seattle, Washington, USA, (pp. 74-83).

Lee, R. B., Kwan, P. C. S., McGregor, J. P., Dwoskin, J., & Wang, Z. (2005). Architecture for protecting critical secrets in microprocessors. *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, (pp. 2-13).

Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., & Horowitz, M. (2000). Architectural support for copy and tamper resistant software. *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, (pp. 168-177).

Lin, H. Y., & Harn, L. (1991). A generalized secret sharing scheme with cheater detection. *Proceedings of the International Conference on the Theory and Applications of Cryptology: Advances in Cryptol-*

ogy, Lecture Notes in Computer Science (739), (pp. 149-158).

Martin, K. M. (1993). Untrustworthy participants in perfect secret sharing schemes. *Cryptography and Coding III*, M. J. Ganley, ed., Oxford University Press, (pp. 255-264).

Menezes, A. J., Oorschot, P. C. van, & Vanstone, S. A. (1997). *Handbook of applied cryptography*. CRC Press, LLC.

Pedersen, T. P. (1991). A threshold cryptosystem without a trusted party. *Proceedings of EUROCRYPT*, (pp. 522-526).

Pederson, T. P. (1992). Non-interactive and information-theoretic secure verifiable secret sharing. *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, Lecture Notes in Computer Science (576), (pp. 129-140).

Rogers, M. P., & Solihin, Y. (2006). Efficient data protection for distributed shared memory multiprocessors. *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, (pp. 84-94).

SESC, Retrieved on XXX from, http://sesc.sourceforge.net/.

Shi, W., Lee, H. H., Ghosh, M., & Lu, C. (2004). Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems. *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, (pp. 123-134).

Shi, W., Lee, H. H., Ghosh, M., Lu, C., & Boldyreva, A. (2005). High efficiency counter mode security architecture via prediction and precomputation. *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, (pp. 14-24).

Shi, W., Lee, H. H., Falk, L., & Ghosh, M. (2006). An integrated framework for dependable and revivable architectures using multicore processors. *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, (pp. 102-113).

Suh, G. E., Clarke, D., Gassend, B., Dijk, M. van, & Devadas, S. (2003). AEGIS: architecture for tamper-evident and tamper-resistant processing. *Proceedings of the 17th International Conference on Supercomputing (ICS)*, (pp. 160-171).

Yan, C., Rogers, B., Englender, D., Solihin, Y., & Prvulovic, M. (2006). Improving cost, performance, and security of memory encryption and authentication. *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, (pp. 179-190).

Yang, L., & Peng, L. (2006). SecCMP: a secure chip-multiprocessor architecture. *Proceedings of the first Workshop on Architectural and System Support for Improving Software Dependability (ASID)*, (pp. 72-76).

Yang, J., Zhang, Y., & Gao, L. (2003). Fast secure processor for inhibiting software piracy and tampering. *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*, (pp. 351-360).

Zhang, Y., Gao, L., Yang, J., Zhang, Z., & Gupta, R. (2005). SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors. *Proceedings of the 11th Intl. Symposium on High-Performance Computer Architecture (HPCA)*, (pp. 352-362).

Zhuang, X., Zhang, T., & Pande, S. (2004). HIDE: An Infrastructure for Efficiently Protecting Information Leakage on the Address Bus. *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, (pp. 72-84).

Li Yang is an assistant professor in the Department of Computer Science and Electrical Engineering at University of Tennessee at Chattanooga. Her research interests include network and information security, databases, and engineering techniques for complex software system design. She authored papers on these areas in refereed journal, conferences and symposiums. She is a member of the ACM.

Lu Peng received his bachelor's and master's degrees in computer science & engineering from Shanghai Jiaotong University, China. He obtained his PhD degree in computer engineering from the University of Florida in Gainesville in April 2005. He joined the electrical and computer engineering department at Louisiana State University as an Assistant Professor in August, 2005. His research focus on memory hierarchy system, multi-core interconnection, power efficiency and other issues in CPU design. He also has interests in Network Processor. He received an ORAU Ralph E. Powe Junior Faculty Enhancement Awards in 2007 and a Best Paper Award from IEEE International Conference on Computer Design in 2001. Dr. Peng is a member of the ACM and the IEEE Computer Society.

Balachandran Ramadass received his bachelor's degree in electronics and communication engineering from Pondicherry University, India. He is currently doing his Master of Science degree in electrical and computer engineering in Louisiana State University. He is also currently doing his Internship in Enterprise power path finding group, Intel Corporation, DuPont. His research focuses are CMP architecture, on-chip power optimization, interconnection network, heterogeneous CMP architecture.