A Case for Near Data Security*

Akhila Gundu, Ali Shafiee Ardestani, Manjunath Shevgoor, Rajeev Balasubramonian
University of Utah

Abstract

Security is a vital requirement in many high-end systems, especially those that make up modern cloud infrastructures. Cloud systems are vulnerable to many attacks, including those by untrusted cloud operators that have access to physical hardware. Memory authentication confirms that an attacker is not modifying the values being returned by the memory system. But it imposes a severe memory bandwidth overhead that limits its adoption in highly cost-constrained cloud systems. The ideas proposed in this position paper attempt to lower these overheads and make memory authentication more palatable for cloud systems.

While near data processing has been explored to improve application performance and power efficiency, it has not been leveraged to improve auxiliary operations for security. This paper argues that logic for memory authentication should be placed on the memory module itself so it has access to significantly higher bandwidth. To preserve security guarantees, the near data processor and its link to the main host processor have to be made secure. We describe this design and estimate the first-order potential for benefit.

1. Introduction

While cloud services are attractive for their low cost, they pose several vulnerabilities. In particular, cloud servers are vulnerable to physical attacks where an untrusted cloud operator may modify the hardware to gather customer data. It is expected that sensitive applications will encrypt all data emerging from the processor e.g., with a framework similar to Intel's SGX [8], to reduce vulnerability to such physical attacks. But this is not enough to keep data safe. In a passive attack, an untrusted cloud operator may snoop on the memory bus and gather information based on the address trace (which is not encrypted on a commodity DDR system) [9]. In an active attack, the cloud operator could attach a custom DIMM that returns spurious data to either cause disruptions or launch a replay attack [3]. In fact, the latter attack is easier to accomplish and more hazardous. To overcome such an active attack, the concept of memory authentication was introduced [3]. Memory authentication provides a guarantee to the user that the value returned by a memory read request is exactly the last value that was written to that address. In spite of several recent innovations [5, 18, 14, 13, 15, 4, 2, 6, 11, 12], state-of-the-art memory authentication continues to pose a high bandwidth overhead of 6X. This is especially problematic for the multi-core memory-intensive workloads that are so popular in the big-data era.

This paper exploits near data processing (NDP), or more specifically, near data security, to shield the processor from this high bandwidth overhead. No prior work (to the best of our knowledge) has attempted to use NDP to target a security feature. This is an especially compelling application of NDP since high bandwidth is a primary advantage of NDP, and security features like memory authentication are especially hungry for memory bandwidth. We introduce the concept of a Secure-DIMM that has a secure processor on it that can perform memory authentication at much higher speeds than a baseline processor. This is possible because the internal bandwidth on the DIMM is significantly higher than the bandwidth into a modern processor. Once the secure processor on the DIMM has a verified response for the data request, it communicates this result via a secure channel back to the processor. The secure channel leverages well-known approaches for hardware/software authentication and secure communication.

While the proposed solution can substantially reduce bandwidth requirements and hence queuing delays, it introduces an additional delay for encryption and decryption over the secure channel. It also results in more expensive hardware because a specialized DIMM is required. The approach can also easily be adapted for use in a 3D-stacked memory device.

2. Background

2.1. Physical Attacks

Attacks can be categorized as Passive and Active attacks [3]. In a passive attack, an adversary silently observes critical information as it moves on the bus. Data that is transferred over the bus can be protected by encryption. In an active attack, an adversary has physical access to the computing system and can modify the data being exchanged. This is typically done by replacing a genuine DIMM with a malicious custom DIMM [3]. Active attacks can be of different types. A *splicing* or *relocation* attack is where the adversary can replace a memory block with a block at a different address. A *Spoofing* active attack involves an adversary exchanging an existing memory block with a malicious one. A *Replay* attack is the third frequent active attack that replaces a memory block located at a given address with the memory block that existed at that location at an earlier point in time.

2.2. Integrity Trees for Memory Authentication

Researchers have proposed authentication primitives like cryptographic hash functions [3, 14] and Message Authentication Code (MAC) functions that can be used to authenticate data. These functions are applied to every memory block and their nonces (random numbers, created on every write of a block, used as an input to prevent replay attacks). The resulting hashes can be stored on the secure processor. This creates excessive storage overheads on the secure processor. Tree-based structures called Integrity trees were proposed to eliminate this storage overhead [11, 6, 2]. The tree splits the memory space into M equal size blocks which form the leaf nodes of an integrity tree. The remaining tree levels are created by recursively applying a hash over the children of that node. This recursive application of the authentication primitive yields a single root node that is stored on the secure tamper-resistant processor [16]. The root reflects the current state of the entire memory. The memory blocks and intermediate tree nodes are stored in the main memory. The number of checks required to verify the integrity of a leaf node depends on the number of

*This work was supported in parts by NSF grants CNS-1302663 and CNS-1423583, and by IBM Research.

memory blocks. The number of checks corresponds to the number of tree levels given by $log_A(M)$ [3] where A is the arity of the tree and M is the number of memory blocks.

2.3. Tree Authentication

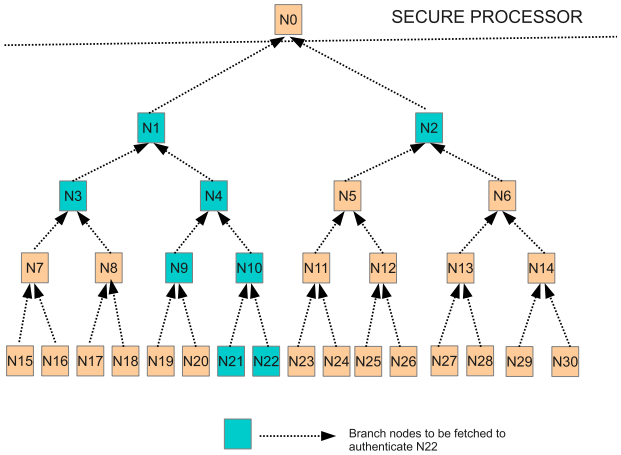

Figure 1: Merkle tree authentication

For each memory block M (i.e., leaf node), a branch [3] exists which starts at the leaf M and ends at the root of the tree. To verify the authenticity of the memory block M, we verify the authenticity of every node on this branch. This is best explained with the example in Figure 1 that shows an integrity tree with 16 memory blocks with an arity of 2. To authenticate a memory block, say N22, the nodes along N22's branch must be fetched along with N22. To re-compute the root node, the branch nodes' siblings must be fetched too. Thus, for authenticating N22, the nodes to be fetched from the untrusted memory are the nodes in blue – N22 and N21, N10 and N9, N4 and N3, N1 and N2. The hash is applied to N21 and N22 to confirm the value of N10. Next, the hash is applied to N9 and N10 to confirm the value of N4, and so on. Finally, the hash is applied to N1 and N2 to confirm the value of the root node on the processor. If the adversary has modified any subset of blocks fetched from memory, at least one of these checks will fail.

2.4. Tree Update

A legitimate change to a memory block will require the tree branch of that block to be updated to reflect the new changed value of the block. Authentication [3] of that memory block has to be done first. Then, the new branch values are computed on-chip and finally stored off-chip.

2.5. Bonsai Merkle Tree

Rogers et al. [12] make the following observation. Assume that a data block and an associated counter value are being encrypted together and being stored. When data is read and decrypted, the processor sees data and the counter value. If the counter value is correct, the data value is correct as well (with a very high probability). So a separate *Bonsai* Merkle tree is constructed out of the counters for every block. This tree is read to confirm that the counter obtained through decryption matches the counter obtained through the Bonsai Merkle tree (BMT). The BMT is significantly smaller because it is based on smaller counter values and not on the entire data value. The BMT can also be organized with high arity to reduce the number of nodes. All the children of a node can also be stored as a single block in memory to further reduce the number of data fetches. For example, the Merkle tree for a 16 GB data set would require fetching 56 blocks, but with a BMT over 8-bit counter values and arity of 64, we would only have to fetch 6 blocks.

3. Designing a Secure-DIMM

Just as secure computation requires a specialized processor, memory authentication in our proposal requires a specialized memory system. The host secure processor would have a specialized (but simple) memory controller that issues a high-level request to the external memory system. A read request issues the address and a pending transaction ID (since the response time is not known). A write request issues the address and the data in encrypted form. The host processor and DIMMs are either connected with point-to-point channels or shared channels with appropriate arbitration mechanisms.

Each Secure-DIMM has a buffer chip (similar to the ones seen on LR-DIMMs [1]) that receives all signals on the channel. The buffer chip is the key innovation in our proposal. It is a secure logic block that implements memory controller functionality (dealing with timing parameters and the states of memory banks), memory authentication functionality (verifying the contents of a BMT), and the basic primitives expected in a secure processor (authenticating itself and its code, and establishing a secure communication link).

The buffer chip is connected to several DRAM chips on the Secure-DIMM with a dense on-DIMM interconnect. Note that a DIMM with four ranks has four times as much internal bandwidth as its external bandwidth. By adding more interconnect layers on the Secure-DIMM, it can provide even higher intra-DIMM bandwidth between its DRAM chips and its buffer chip. These intra-DIMM interconnects can also run at frequencies much higher than typical DDR memory channels.

A cache line request is handled entirely by one Secure-DIMM. Each Secure-DIMM organizes its data as a BMT, similar to that of prior work [12]. Once the buffer chip receives a read request from the main host processor, it generates all the necessary read requests for memory authentication and places them in its memory controller. The memory controller issues these requests to the ranks on the Secure-DIMM using a standard DDR protocol. Once data is received, the necessary checks are performed. Note that the buffer chip now stores the root node of the BMT. After the received data value has been authenticated, it needs to be returned back to the main host processor. To avoid arbitration overheads, each Secure-DIMM takes turns in round-robin order to return a pending request back to the processor.

Since the secure host processor has off-loaded memory authentication to the buffer chip, it has to verify that it is indeed receiving responses from a buffer chip that it trusts. We walk through the required steps below. At a high level, note that these steps are similar to what the cloud user must do to initiate her application on the cloud's secure processor [15, 17].

First, the secure buffer chip needs to establish its own private/public key combination. This is either done by the manufacturer or at run time by the user with the help of a PUF circuit [15, 17, 4]. This private/public key may be used by the secure host processor and the secure buffer chip to periodically establish session keys for communication. The secure buffer chip uses its

private session key to encrypt the following tuple:
$< data\ response,\ pending\ transaction\ id,\ ECC,\ code\ hash >$.
The *code hash* is tracked by the secure buffer chip in a special register. The buffer chip is only capable of executing a single kernel and a single application. Every time either of these is changed, the hardware computes a hash of the kernel and the application codes and stores it in the special register. It is computationally intractable for an attacker to place her own code on the buffer chip such that it hashes to the same value as the code being provided by the cloud user. When the host processor receives the encrypted tuple, it uses the buffer chip's public session key to decrypt the tuple. It confirms that it has received a valid pending transaction ID, ECC, and code hash. It then proceeds with the data value it has received. Note again that the private/public key encryption of data on the link guarantees that the cloud user is communicating with a valid buffer chip, while the code hash guarantees that the buffer chip is executing trusted code. The pending transaction ID can also include a hash of the address to thwart any possible replay attacks by a fake DIMM on the same channel.

4. Expected Benefit

Consider a baseline secure processor that, like many high-end processors today, has four DDR3 channels, where each channel can support three LR-DIMMs at a frequency of 533 MHz [7]. We assume that each LR-DIMM implements its own BMT; this allows the secure processor to work on 12 different memory accesses at the same time. Alternatively, all 12 LR-DIMMs may collectively form a single BMT to leverage more channels for a single memory authentication. Either way, the net bandwidth available to the secure processor is 533 MHz $\times$ 4 channels $\times$ 64 bits $\times$ 1/D, where D is the bandwidth overhead of memory authentication. If we assume that each LR-DIMM has a capacity of 16 GB and implements its own BMT, the value of D is 6.

In the Secure-DIMM, the internal bandwidth is estimated as follows. We conservatively assume that only a single 64-bit data channel connects the buffer chip to the DRAM chips on the Secure-DIMM. We assume that the channel on the Secure-DIMM can operate at a frequency of 800 MHz. The net internal bandwidth on all 12 Secure-DIMMs is therefore 800 MHz $\times$ 1 internal channel $\times$ 64 bits $\times$ 12 $\times$ 1/D. This bandwidth is 4.5$\times$ the bandwidth available to the host processor. Therefore, assuming a large number of threads and a bandwidth-constrained system, the Secure-DIMMs can perform memory authentication 4.5$\times$ faster, i.e., the overhead of memory authentication is reduced from 6$\times$ to 2.1$\times$. The bandwidth into the host processor is also underutilized by a similar amount because data responses include the code hash and transaction id.

The proposed approach does introduce additional delays on every read because of the encryption/decryption required for the secure link. This delay is expected to be about 45 ns [10]. The cost of the system also goes up because of our use of a specialized DIMM; for reference, an LR-DIMM that has comparable design complexity has a cost-per-bit that is 48% higher than that of commodity DIMMs. Note that even though we now have a secure host processor and a secure buffer chip, we are not doubling our processor real estate. The secure buffer chip does not include a full-fledged processor and cache; also, we are moving memory controller and BMT functionalities from the host to the buffer chip, not replicating them.

5. Conclusions

The paper argues that near data security has the potential to shelter the processor from the high bandwidth requirements of features like memory authentication. The near data processor will need logic to perform memory controller duties, perform BMT authentication operations, and authenticate itself and the code it runs. A first-order analysis on an example server configuration points at a 4.5$\times$ reduction in memory authentication bandwidth overheads. This motivates a more detailed analysis of the proposed approach.

References

[1] "Load-Reduced DIMMs," http://www.micron.com/products/dram-modules/lrdimm.

[2] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli, and P. Guillemin, "TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks," in *Proceedings of Cryptographic Hardware and Embedded Systems*, 2007.

[3] R. Elbaz, D. C. C. Gebotys, R. Lee, N. Potlapally, and L. Torres, "Hardware Mechanisms for Memory Authentication: A Survey of Existing Techniques and Engines," in *Transactions on Computational Science IV*, 2009.

[4] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and Merkle Trees for Efficient Memory Authentication," in *Proceedings of Ninth International Symposium on High Performance Computer Architecture*, 2003.

[5] B. Gassend, G. E. Suh, D. E. Clarke, M. van Dijk, and S. Devadas, "Caches and Hash Trees for Efficient Memory Integrity Verification," in *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture (HPCA'03), Anaheim, California, USA, February 8-12, 2003*, 2003.

[6] E. Hall and C. S. Jutla, "Parallelizable Authentication Trees," in *SAC'05 Proceedings of the 12th international conference on Selected Areas in Cryptography*, 2006.

[7] HP, "Configuring and using DDR3 memory in HP ProLiant Gen8 Servers."

[8] Intel, "Intel Software Guard Extensions Programming Reference," software.intel.com/sites/default/files/329298-001.pdf, 2013.

[9] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access Pattern Disclosure on Searchable Encryption: Ramification, Attack, and Mitigation," in *Proceedings of NDSS*, 2012.

[10] T. Kgil, L. Falk, and T. Mudge, "ChipLock: Support for Secure Microarchitectures," *SIGARCH Comput. Archit. News*, 2005.

[11] R. C. Merkle, "Protocols for Public Key Cryptosystems," *Security and Privacy, IEEE Symposium on*, 1980.

[12] B. Rogers, S. Chhabra, Y. Sohilin, and M. Prvulovic, "Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly," in *Proceedings of MICRO*, 2007.

[13] W. Shi, H. hsin S. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High Efficiency Counter Mode Security Architecture via Prediction and Precomputation," in *Proceedings of 32nd Intl. Symp. on Computer Architecture*, 2005.

[14] G. E. Suh, D. Clarke, B. Gassend, M. v. Dijk, and S. Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processors," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[15] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing," in *Proceedings of the 17th Annual International Conference on Supercomputing*, 2003.

[16] G. E. Suh and S. Devadas, "Design and Implementation of the AEGIS Single-Chip Secure Processor using Physical Random Functions," in *Proceedings of ISCA*, 2005.

[17] G. E. Suh, C. W. O'Donnell, and S. Devadas, "AEGIS: A Single-chip Secure Processor, PhD Thesis, Massachusetts Institute of Technology," 2005.

[18] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," in *Proceedings of ISCA*, 2006.