Investigating the Potential for Near Data Processing to Reduce Secure Memory Overheads

Casey Nelson[1], R. Iris Bahar[1], Tamara Lehman[2]
casey_nelson@brown.edu, iris_bahar@brown.edu, Tamara.Lehman@colorado.edu
Brown University[1], University of Colorado Boulder[2]

Abstract

Despite gaining increasing popularity, secure memory is known to decrease performance and increase energy consumption when accessing data from main memory. Prior work has proposed using metadata caching as a means to reduce overheads. However, the use of caching has introduced new complications because the heterogenous miss costs and reuse distances of security metadata are not compatible with traditional caching algorithms. This work investigates an alternative to caching, specifically near data processing, as a means for reducing secure memory overheads. This work concludes that the use of near data processing has the potential to cut the performance overheads of secure memory in half warranting further investigation.

1. Introduction

Secure memory architectures are used to protect a user's data when they do not have physical control over the hardware. When user's do not have control over the hardware, their data is vulnerable to attacks that snoop data as it travels across the memory bus or swap out the memory device entirely.

In secure memory implementations, such as Intel SGX, the chip is treated as the trusted computing base (TCB) while everything else is untrusted. As such, secure memory implementations must add security primitives to protect the confidentiality and integrity of data stored off chip [1]. Data confidentiality is provided through the use of counter mode encryption on blocks of data [7], and data integrity is verified through a Merkle Tree variant, specifically, a Bonsai Merkle Tree [8]. These security mechanisms require 7 additional metadata requests per every 1 data request to main memory (1 counter, 1 data HMAC, and 5 tree nodes) assuming the max Intel SGX enclave size of 128 MiB [4]. This overhead increases with the amount of memory being protected.

A common method for reducing secure memory overheads takes advantage of the fact that caches are within the trusted computing base to cache security metadata (counters, HMACs, and tree nodes) on chip to cut down on the additional metadata computations needed per data request. Prior work has proposed caching metadata in the LLC, a dedicated metadata cache, or a combination of the two [9] [1]. However, the benefits of metadata caching on secure memory performance and energy overheads are limited by the fact that the various metadata types have incompatible miss costs and reuse distances with existing caching algorithms [1].

This work investigates near data processing as an alternative to metadata caching to improve the performance of secure memory architecture. Prior work has proposed near data processing to improve the performance and reduce the energy consumption of data structures with poor cache locality [5]. Since the access patterns of security metadata are similarly incompatible with caching, there is reason to believe that applying near data processing to secure memory computations can provide a similar level of improvement. While [2] made a case for the use of near data processing in secure memory architectures, this work is the first to actually attempt to evaluate such a scheme.

This work provides:

1. A proposed scheme for using near data processing in the computation of security metadata
2. Estimates of the expected improvements the proposed scheme can provide to justify pursuing it further

2. Using NDP to Process Security Metadata

The proposed design is depicted in figure 1. In this design, all metadata computation begins with an on-chip memory controller, or metadata controller as it is called in this work. First, when the metadata controller receives a data request, it creates all of the corresponding metadata requests and forwards them to memory. The address space in this scheme is laid out so that the different types of metadata requests for a given data request are each routed to a different vault.

For data, data HMACs, and counters, the values will be returned back to the metadata controller. The metadata controller will then compute the HMAC using the encrypted data and its counter, compare this value with the HMAC read from memory, and, finally, decrypt the data using the counter. The hash of the counter will also be computed in the on chip metadata controller.

For tree nodes, each node will be read and hashed in memory. Then, each node with its hash appended will be returned to the metadata controller. The metadata controller will then compare the hash of the counter to the last level node, the hash of the last level node to its parent node, etc. This chain will continue all the way to comparing the hash of the top level node with the root stored on chip, which will complete the verification of the counter. This implementation does not take hash comparisons off of the critical path of a data request. However, this is acceptable because the real delay in secure memory comes from having to make several read requests and compute several hashes sequentially. The overhead of making comparisons is trivial. This implementation is able to effectively remove all metadata reads and eliminate all but 2 hash computations out of the critical path of the data request. The hash of the

data must be computed in the metadata controller because it is based off of both the counter and the ciphertext. Additionally, the hash of the counter must be computed on chip to avoid a replay attack. If counter hashes were to be computed in memory, an adversary could roll back the ciphertext, data HMAC, and counter to a previous instance of these values. However, they could still have the current counter's hash returned. As such, both the counter and the data will pass verification even though they were rolled back. By computing the hash of the counter in memory, it can be guaranteed that the counter verified is the same as the counter used for decryption.

The scheme proposed is resistant to both replay and arbitrary rewrite attacks. First, an attacker cannot replay an old set of metadata and have it verified because they cannot roll back the root node stored on chip. As such, the verification process will fail with the final comparison to the root node. Additionally, an attacker will not be able to send back an arbitrary data value that can pass verification because they do not know the secret key to be able to compute accurate hashes.

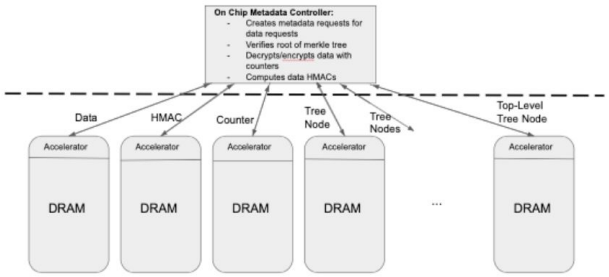

Figure 1: Proposed Design for Secure Memory with NDP

2. Evaluation

2.1. Simulator Configuration

The evaluation was conducted using the full-system simulator, SMCSim [6], which is a Gem5 [10] based simulator with added support for NDP architectures. The configuration used for the simulator can be found in table 1.

System Configuration	
CPU	1 in-order processor (ARMv8)
NDP Cores	1 in-order processor per vault (ARMv8)
L1 Cache	32kB instruction cache + 32kB data cache, both 2-way set associative
L2 Cache	512kB, 8-way set associative
Main Memory	16 DRAM vaults, 128MB each

Table 1

2.2 Results

The results of the evaluations can be found in figures 4 and 5. All results were collected by fast forwarding the benchmark by 1 billion instructions and then simulating for 1 billion more. This work was evaluated using benchmarks from the SPEC CPU2017 testing suite [11]. Benchmarks were chosen for their distinct MPKI values, which represents the last level cache (L2 in this case) misses per 1000 instructions

Figure 2 shows the total simulated seconds required to run each of the selected benchmarks (mcf, gcc, lbm, and bwaves) with no security metadata (baseline), with security metadata processed in parallel (parallel), and with security metadata processed sequentially (serial). Processing metadata sequentially adds about a 50% to 120% overhead to the overall seconds needed to run the benchmark in comparison to the baseline. This overhead increases as the MPKI of the benchmark increases. The parallel optimization cuts the overall time overheads to only about 17% to 40% in comparison with the baseline.

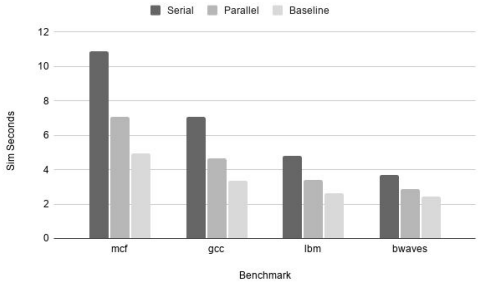

Figure 2: Time required for Baseline (non-secure memory), serial (standard secure memory), and parallel (secure memory with NDP) implementations

Figure 3 specifically shows the increase in DRAM time because DRAM is where most of the secure memory overheads are incurred. DRAM time is measured as the average amount of time a request spends being processed in DRAM. The overhead of processing metadata sequentially and in parallel are plotted with respect to a baseline without secure memory. For each of the benchmarks, processing security metadata in parallel cuts the average amount of time it takes to process a request in DRAM in half.

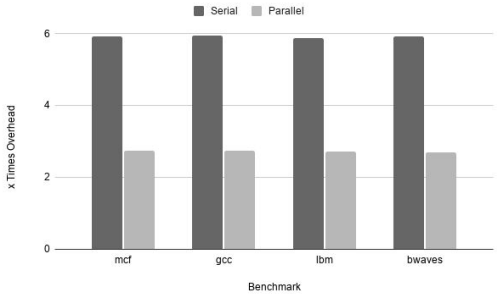

Figure 3: The DRAM overhead of processing metadata serially rather than in parallel (using NDP) in comparison to a baseline implementation (non-secure memory). DRAM overhead is computer using the number of cycles a data request spends being processed in DRAM

3. Conclusion

Initial evaluations show that near data processing has the potential to improve secure memory overheads by running metadata computations in parallel. Approximate evaluations show that NDP has the potential to cut secure memory metadata performance overheads in half. This is more significant than the performance gains of prior work which attempts metadata fetching in parallel [3]. As such, further investigation of the benefits of near data processing to secure memory is warranted.

References

[1] T. Silbergleit Lehman, A. D. Hilton and B. C. Lee. "MAPS: Understanding Metadata Access Patterns in Secure Memory". In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018

[2] A. Gundu, A. S. Ardestani, M. Shevgoor, and R. Balasubramonian. "A case for near data security". In *2nd Workshop on Near Data Processing*, 2014

[3] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, M. K. Qureshi. "SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories". In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018

[4] I. Anati, F. Mckeen, S. Gueron, H. Haitao, S. Johnson, R. Leslie-Hurd, H. Patil, C. Rozas, and H. Shafi. "Intel software guard extensions (Intel SGX)". In *Tutorial at International Symposium on Computer Architecture (ISCA)*, 2015

[5] J. Choe, A. Huang, T. Moreshet, M. Herlihy, and R. I. Bahar. "Concurrent Data Structures with Near-Data-Processing: an Architecture-Aware Implementation". In *31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019

[6] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. "Design and Evaluation of a Processing-in-Memory Architecture for the Smart Memory Cube". In *International Conference on Architecture of Computing Systems*, 2016

[7] S. Gueron, "A memory encryption engine suitable for general purpose processors". In *The Proceedings of the International Association for Cryptologic Research (IACR)*, 2016

[8] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and Bonsai Merkle trees to make secure processors OS- and performance-friendly". In *The Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007

[9] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas. "Caches and hash trees for efficient memory integrity verification". In *The Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2003

[10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. "The Gem5 simulator". A*CM SIGARCH Computer Architecture News 39*, 2011

[11] J. Bucek, K. Lange, and J. v. Kistowski. "SPEC CPU2017 - Next Generation Compute Benchmark". *ICPE' 18 Companion*, 2018