Using a Fast Subtree for Efficient Secure NVMs

Samuel Thomas
Brown University
USA

Kidus Workneh
University of Colorado, Boulder
USA

Jac McCarty
Bryn Mawr College
USA

Joseph Izraelevitz
University of Colorado, Boulder
USA

Tamara Lehman
University of Colorado, Boulder
USA

R. Iris Bahar
Colorado School of Mines
USA

1 Introduction

Suppose an application is dependent on some value in memory x at address a. Will the value of x ever be in a state other than what the application expects it to be in? As it turns out, traditional systems with volatile memory are subject to active and passive attacks where data in memory can be selectively and precisely be targeted and corrupted to potentially malicious values. These attacks can be performed by legitimate guest software on the system that doesn't need to have access permissions to sensitive data, so defenses in software are insufficient.

Such is the motivation for *secure memory*. Secure memory describes hardware-enforced integrity protection and privacy of all data in sensitive memory devices (i.e., off-chip DRAM, etc). Generally, confidentiality is achieved by encrypting values as they pass from trusted to untrusted storage hardware via counter-mode encryption and the integrity of data is protected by a Bonsai Merkle Tree (BMT) with a keyed hash message authentication code (HMAC) [2, 3]. The BMT is a tree of hashes where the root resides in the trusted hardware (i.e., on-chip) to ensure that values and hashes in memory cannot be replayed. To fetch a value from an untrusted storage device, the hardware also needs to fetch its associated encryption counter to perform decryption as well as its path through the BMT and its HMAC to compare the computed hashes against the trusted root. If the computed hashes match the stored hashes, then the hardware can guarantee that the value in memory hasn't been tampered.

While such a scheme successfully protects the confidentiality of data in memory, it comes at steep performance cost. What was once a single fetch to retrieve some data now results in several fetches to retrieve both data and secure memory metadata. This is typically optimized by including a volatile cache that is reserved for secure memory metadata, but the emergence of non-volatile memories (NVMs), provides even further challenges to secure memory in that secure memory metadata (i.e., encryption counters, BMT nodes, and HMACs) need to be *crash consistent* with application data in NVM. To provide crash consistency, the secure memory protocol *could* persist all values that are written to the metadata cache directly to memory. In doing so, the caching policy can be referred to as a write-through cache, as opposed to its typical writeback nature. This scheme, termed *strict metadata persistence*, is crash consistent because each metadata value is persisted directly and atomically, so all values in memory are in a crash consistent state at all times. However, this scheme is not realistic, in that it can lead to steep performance overheads (up to $25X$) at runtime.

An alternative approach, dubbed *leaf metadata persistence* [4], addresses the performance issue by taking a *lazy* approach to crash consistency. The tree nodes are written to the volatile metadata cache and only written back to memory on eviction (i.e., they are not written-through directly). After a crash, at system recovery, each of the inner nodes of the integrity tree are recomputed from the hashes of its leaves. If the computed tree root matches the stored tree root, then the system can be safely rebooted. These two extreme baselines describe an inherent trade-off between runtime *performance overhead* and *recovery time*, with an additional axis in the solution space on *hardware cost*.

In this paper, we propose a hybrid, adaptable metadata persistence scheme for secure NVMs in which a single, small subtree of the underlying BMT implements leaf metadata persistence, whereas most of memory is protected by strict metadata persistence. In doing so, our proposed protocol implements an adaptable crash consistency policy, which adapts to changing application behavior with minimal hardware cost. Such a solution ensures that our protocol scales with changing workload demands and as on-chip architectures continue to develop and change.

2 Design

We propose a secure NVM protocol in which there is a single *fast subtree* that balances a reasonable runtime overhead with controllable recovery times and minimal hardware overhead. In particular, we achieve this goals by using a dynamic hybrid persistence strategies within the same BMT.

We work from the assumption that a small number of contiguous addresses in physical memory are frequently accessed (i.e., "hot"). Given this, our proposed protocol protects a small region of physical memory with a fast persistence protocol while most addresses are persisted strictly to keep the work required at recovery time low. In our fast subtree protocol, there is a dynamic persistence protocol that tracks hot regions of physical memory, which is the hottest subtree of the underlying BMT in order to adapt to changing in-memory hotness at runtime. We use an additional register

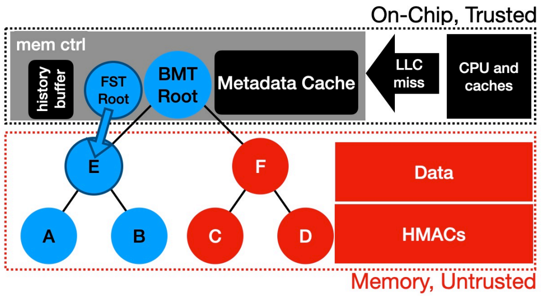

Figure 1. Proposed architecture. Red nodes implement strict persistence. Blue nodes implement leaf persistence.

on-chip to track this subtree. The subtree implements a leaf persistence strategy, where tree nodes assumed to be stale at the time of a system failure (blue nodes in Fig. 1). The rest of the BMT implements strict persistence to minimize recovery time after a crash (red nodes in Fig. 1). Implementing a strict persistence strategy in the region outside of the subtree, while slow at runtime, will not occur often, thereby minimizing impact on overall performance and reduces the work required at recovery time.

As application behavior changes, our protocol adapts the leaf persistence region to the new hot region. When transitioning from subtree T to T', all inner integrity nodes of T must be persisted before T' can implement the leaf persistence protocol in order to preserve the crash consistency and security guarantees. Note that the only ancestral paths from subtree T that need to be written to memory are those originating from modified (dirty) data, so we can quickly determine which nodes these are by scanning the dirty bits in the metadata cache. Only nodes in the metadata cache that fall within the subtree will have their dirty bits set as all other metadata are written-through to memory. The path from T to the root also must always be persisted on movement.

But how safe is it to assume that applications use only a small contiguous region of physical addresses? If multiple programs are mapped to distinct virtual address spaces, it is highly unlikely that they will exhibit in memory physical locality. One possible solution would be to have a per-core fast subtree, but this incurs memory complexity that is scalably worse for many-core devices. Instead, we propose *hardware-software co-design* to keep hardware complexity low and modify application behavior from the operating system's memory management unit to increase the in-memory subtree locality. In particular, we modify the physical page reclamation process to traverse the `free_area` structs for the subtree with the most free pages, and re-organize the list to put those free pages at the head. In Linux, pages allocated by fetching the head elements from the lists in the `free_area` structs, so our modified operating system increases the likelihood that consecutive allocations will occur within the same subtree region. Seeing as this is consistent for all processes, putting this modification in the OS keeps it agnostic to which process is requesting the physical memory.

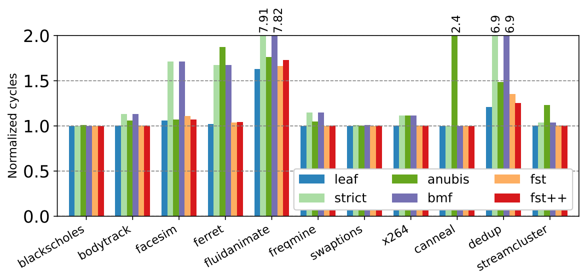

Figure 2. Cycles to execute region of interest of PARSEC benchmarks normalized to volatile secure memory.

3 Evaluation and Conclusion

We evaluate our fast subtree protocol on the PARSEC benchmark suite version 3.0 with the simlarge inputs in gem5, a cycle accurate architecture simulator. We configure our 8GB memory to have 305ns read latency and 391ns write latency, and we configure the secure memory hardware to have a 64kB metadata cache with an 8 level BMT to remain consistent with Intel SGX.

Fig. 2 shows the number of cycles executed for each hardware configuration normalized to volatile secure memory, so lower is better. In this evaluation **leaf** refers to the leaf metadata persistence policy, **strict** refers to the strict metadata persistence policy, **anubis** and **bmf** refer to our implementation of prior arts [5] and [1], respectively, both requiring substantially higher hardware space. **fst** portrays our proposed "fast subtree" protocol without the modified operating system, and **fst++** portrays our protocol with the modified operating system. These benchmarks demonstrate a wide range of CPU workloads. Our results demonstrate that our approach consistently provides performance similar to that of leaf persistence, while offering the recovery benefits of strict persistence for most addresses.

In summary, the fast subtree protocol uses hot region tracking to create an adaptable crash consistency protocol for secure NVMs.

References

[1] A. Freij, H. Zhou, and Y. Solihin, "Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1227–1240.

[2] 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 *Proc. International Symposium on Microarchitecture (MICRO)*, 2007.

[3] J. Yang, Y. Zhang, and L. Gao, "Fast secure processor for inhibiting software piracy and tampering," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2003.

[4] M. Ye, C. Hughes, and A. Awad, "Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2018.

[5] K. A. Zubair and A. Awad, "Anubis: ultra-low overhead and recovery time for secure non-volatile memories," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 157–168.