Rethinking Metadata Caches in Secure NVMs

Samuel Thomas
Brown University
USA

Hammad Izhar
Brown University
USA

Eliott Dinfotan
Boston Univeristy
USA

Tali Moreshet
Boston University
USA

Maurice Herlihy
Brown University
USA

R. Iris Bahar
Colorado School of Mines
USA

1 Introduction

As computation workloads become increasingly memory intensive, non-volatile main memories (NVMs) have become more dense in order to increase capacity, issue improved bandwidth and provide lower latency. However, a consequence of increased device density is that volatile and non-volatile storage devices are increasingly susceptible to physical vulnerabilities and attacks like Rowhammer, RowPress, and RAS Clobber [2, 4, 6]. These vulnerabilities, coupled with the increasing move towards cloud computations means that securing NVMs is a critical problem.

The *secure memory* protocol keeps data in an encrypted state by implementing *counter-mode encryption* [8], where each data has a hashed message authentication code (HMAC) and temporally and spatially unique encryption counter in memory. To protect against replay attacks, these encryption counters are then protected by an "integrity tree," or a tree of hashes such that all inner nodes of the tree can be considered hashes of their children (i.e., Merkle tree [5]). The root of the Merkle tree is stored in trusted hardware on-chip, and acts as a root of trust for all values in memory. To fetch some data from NVM, the secure memory protocol fetches the associated HMAC, encryption counter, and path through the integrity tree protecting that data. Once these values are fetched, their hashes are computed and verified to ensure that data hasn't been corrupted nor replayed since its last trusted state.

To address this, modern secure memory protocols are heavily dependent on *metadata caches* [1, 3]. This cache resides on-chip, and holds recently accessed secure memory metadata. The performance benefits of this are twofold: (1) like traditional caches for application data, recently accessed values can be fetched with lower latency than main memory, and (2) values in the cache are trusted, so data authentications only need to go up to the first cache hit. That is, the *metadata per data* required to authenticate some data is reduced on average relative to not having a metadata cache. Like data caches, however, use of a volatile metadata cache implies that writeback caching protocols provide insufficient *crash consistency*. On a power loss, if there has been some update in the cache state, then the value is stale in NVM. Providing crash consistency results in even further slow downs in secure memory. As such, navigating the benefits of the metadata cache while incurring its fundamental incompatibility with NVM is a non-trivial challenge.

In this paper, we ask the question *how else may we reduce the metadata required per data authentication?* Answering this question effectively can take two forms: (1) we achieve this end without the use of metadata caches, and (2) we can increase the effectiveness of metadata caches in scenarios in which the metadata cache performs poorly. Both outcomes potentially have implications on the underlying crash consistency protocol, which we explore in this paper.

In this work, we proposing addressing the limitations of the metadata cache for secure NVMs by Huffmaninzing the secure memory Merkle tree. As such, more frequently accessed addresses in NVM will have shorter authentication paths, thereby reducing the metadata required per data authentication. To do so, we propose a secure memory state machine in which authentications and Huffman tree reconstructions can both occur safely in the same system. To reduce the cost of reconstruction, we consider several approaches to constructing the Huffmanized Merkle Tree – including dynamic Huffman encoding [7] and a novel approximate Huffman tree construction technique.

2 Design

We propose a secure NVM protocol in which the authentication path length through the integrity is reduced as addresses are accessed more frequently. This is done by maintaining access counters for each page in NVM, and constructing a Huffman tree based on these counters. If done naïvely, however, the amount of work to build a Huffman tree from an 8GB NVM would take billions of memory accesses, and would need to occur somewhat frequently in order for the shape of the tree to reflect application behavior. Instead, we propose implementing an adaptable Huffman tree based on the FGK algorithm [7]. This design decision ensures that updates are more reasonable (tens of accesses), but occur more frequently.

Unlike traditional secure memory systems, the Huffmanized Merkle Tree faces unique challenges at runtime and in preserving crash consistency by dynamically changing the shape of the Merkle Tree. In particular, suppose an authentication or crash were to occur in the middle of a Huffman reconstruction. In this case, the hashes that make up the root of the tree still reflect the tree state before construction.

In the case of an authentication, this may result in a false positive detection of corruption. In the case of a power loss, the computed root of the restored integrity tree post reboot is inconsistent with the stored root.

Our proposed protocol explicitly addresses each of these concerns. At runtime, our proposed protocol conforms to a state machine with three states: *normal*, *Huffman*, and *pending*. In normal state, all data can be authenticated normally and the frequency counters are updated. Once the frequency counters result in a need to restructure the Huffman tree, the state machine transitions from normal to Huffman state, in which the relevant metadata is fetched from NVM and the new tree shape is recomputed. While in Huffman state, the state machine blocks all authentications, and authentications can only resume when the state machine returns to normal state. In the event that no nodes are interchanged, the state machine returns to normal state as the hashes through all paths remain consistent. However, in the event of any interchanges, the state machine transitions from Huffman state to pending state in which all relevant updates are sent to memory. Only once all memory ops that adjust tree state have returned does the state machine return from pending state to normal state.

In order to ensure crash consistency, the Huffmanized Merkle Tree needs to write-through all writes to NVM that occur while the state machine is in pending state. Otherwise the tree shape in NVM may be stale on a crash, regardless of the state machine state. Typically, updates to NVM are made to appear "atomic" by loading tree updates to the write-pending queue (WPQ) and only sending those to memory when a done bit is set. However, this is insufficient when the number of updates may be larger than the WPQ, as may be the case with restructuring the integrity. Instead, the Huffmanized Merkle Tree needs to perform undo logging in NVM (where the prior pointer and address are saved to NVM). Then, the done bit is only set when transitioning from pending state back to normal state. At recovery time, the status of the done bit is checked, and the tree shape is returned to the state as defined by the undo log if the done bit is not set. Even without a metadata cache, the Huffmanized Merkle Tree needs to perform undo logging when updating the tree state to ensure crash consistency in the event of a power loss during a tree restructuring operation.

3 Evaluation and Conclusion

We implement and evaluate the Huffmanized Merkle Tree in gem5, a cycle accurate architecture simulator. We then measure its performance against the graph500 benchmark, which performs breadth-first search against graphs of varying sizes. Our evaluation demonstrates performance with graphs of 125MB, 250MB, 500MB, 1GB, and 2GB. We configure an 8GB memory to have 305ns read latency and 391ns write latency to be consistent with Optane. Our evaluation compares the baseline secure memory protocol to the Huffmanized Merkle Tree with and without a metadata cache.

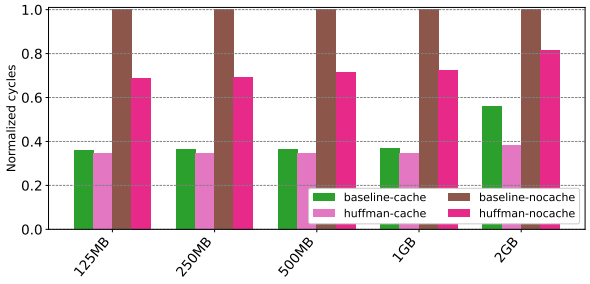

Figure 1. Normalized cycles to perform BFS in graph500 normalized to the baseline secure memory protocol without a metadata cache.

Fig. 1 shows the cycles to execute the benchmark normalized to the baseline secure memory protocol without a metadata cache. A few things become clear from this evaluation: (1) the efficacy of the metadata cache becomes worse as the application demands more memory; (2) when the metadata cache is ineffective, the Huffmanized Merkle Tree provides performance benefits up to 45.6%; (3) without a metadata cache, the Huffmanized Merkle Tree provides significant performance benefits. In these cases, the Huffmanized Merkle Tree reduces metadata per data by 22% on average.

In summary, Huffmanized Merkle Tree is a fundamentally different approach for optimized secure NVMs in which performance is not tightly bound to the efficacy of the metadata cache.

References

[1] A. Awad, S. Suboh, M. Ye, K. A. Zubair, and M. Al-Wadi, "Persistently-secure processors: Challenges and opportunities for securing non-volatile memories," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 610–614.

[2] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 747–762.

[3] J. Huang and Y. Hua, "A write-friendly and fast-recovery scheme for security metadata in non-volatile memories," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 359–370.

[4] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Rowpress: Amplifying read disturbance in modern dram chips," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–18.

[5] R. C. Merkle, "Protocols for public key cryptosystems." in *Proc. Symposium on Security and Privacy (SP)*, 1980.

[6] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[7] J. S. Vitter, "Design and analysis of dynamic huffman codes," *Journal of the ACM (JACM)*, vol. 34, no. 4, pp. 825–845, 1987.

[8] G. X. Yao, R. C. C. Cheung, and K. F. Man, "Counter Embedded Memory architecture for trusted computing platform," in *Proceedings of 2010 21st IEEE International Symposium on Rapid System Protyping*. Fairfax, VA, USA: IEEE, Jun. 2010, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/5656329/