Cordelia: A Huffmanized Merkle Tree for Secure Memory

Samuel Thomas
Pomona College
Claremont, CA, USA
samuel.thomas@pomona.edu

Gal Sela
EPFL
Lausanne, Switzerland
gal.sela@epfl.ch

Tali Moreshet
Boston University
Boston, MA, USA
talim@bu.edu

Maurice Herlihy
Brown University
Providence, RI, USA
mph@cs.brown.edu

R. Iris Bahar
Colorado School of Mines
Golden, CO, USA
ribahar@mines.edu

Abstract

Securing data stored in memory is a critical problem that is expensive to overcome. Typically, the memory controller logic can be extended to implement secure memory protections via a Bonsai Merkle Tree (BMT), a static, balanced authentication structure. We propose Cordelia, an optimization for secure memory in which the BMT is constructed as a Huffman tree such that frequently accessed addresses have shorter BMT leaf-to-root paths. By using a Huffman tree, the work required to authenticate data in secure memory becomes a function of how frequently that data is accessed. In so doing, Cordelia achieves efficient secure memory with limited on-chip complexity by storing Huffman metadata in memory, where space is less constrained. Our approach demonstrates performance improvements of 22% relative to the state-of-the-art on average for high-performance computing workloads. We further show that these benefits scale even with strain on the metadata cache.

CCS Concepts

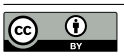
• **Security and privacy** → **Hardware-based security protocols**; Hardware attacks and countermeasures; • **Computer systems organization** → *Architectures*.

Keywords

Secure memory, efficient memory systems, architecture security.

1 Introduction

Applications, developers, and computing platforms alike are responsible for protecting the security of sensitive proprietary- and user-data. Unfortunately, secure computation cannot be achieved through safe software or trusted hardware alone [27, 37, 68], as memory devices are subject to a variety of well known physical vulnerabilities [47, 56, 70, 85]. These vulnerabilities can be leveraged through active or passive attacks that can be performed from software gadgets by legitimate guest processes [40] without needing physical access to the device and provide a powerful attack surface to malicious adversaries [1, 2, 10, 14, 21, 50, 58, 66, 68].



This work is licensed under a Creative Commons Attribution 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846843

Given the rate at which these attacks continue to be discovered [36, 40, 47, 86], safe memory devices should be robust against *arbitrary corruption*. Thus, *secure memory* devices have been proposed to protect the integrity of data in memory [69] by extending the memory controller logic to implement a memory encryption engine (MEE) [28]. In particular, secure memory describes a scheme in which data has its integrity protected by security *metadata* (e.g., cryptographic hashes) arranged as an *integrity tree* (i.e., a Bonsai Merkle Tree, or BMT [51, 63]). That is, because hashes need to be stored in untrusted memory, they too are recursively hashed such that a single hash, which serves as a global root of trust, can be securely stored in trusted hardware on-chip. Then, to authenticate a memory read, the MEE fetches the data and its associated metadata to verify that it is safe from corruption. Its integrity is verified by computing the hash and comparing it to the expected value (the stored hash), and this process is repeated recursively up to the root of trust (the BMT root).

Secure memory comes with a steep performance cost, both in terms of latency and the additional memory bandwidth due to maintaining its metadata [38]. To optimize this protocol, state-of-the-art proposals and commodity secure memory hardware alike deploy a *metadata cache* [5, 23, 38, 43, 44] for recently accessed metadata. The benefits of this cache are twofold: ① metadata can be fetched at lower latency than going to memory, and ② the metadata cache resides on-chip, so cached values are trusted. That is, when authenticating some data, BMT nodes need to be fetched from the leaves up to the first metadata cache hit (or the BMT root if none of the path is in the cache). By retrieving fewer metadata, the amount of work to perform an authentication is reduced.

We propose a secure memory architecture that decouples the potential optimization benefits from the constraint on the metadata cache. In particular, we reduce BMT accesses by redesigning the data structure such that shorter leaf-to-root paths may be achieved in the face of poor metadata cache utilization. We achieve this by restructuring the BMT to be a *Huffman tree* [35], which maintains the provably shortest leaf-to-root path relative to access frequency. The benefits of Huffman encodings have been well-studied in a variety of contexts [22, 25, 59, 75, 78]. Notably, our approach enhances optimization ② of using a metadata cache without adding on-chip components. The metadata cache reduces BMT accesses for *recently* used metadata while the Huffmanized BMT data structure reduces accesses for *frequently* accessed metadata, even when these addresses are not captured by temporal or spatial locality. The Huffmanized BMT will place frequently accessed nodes closer

to the tree root, which means that even uncached authentication paths will have fewer traversals in our approach. This optimization for shorter leaf-to-root authentication paths will be provided by the authentication data structure construction and reconstruction regardless of the cache utilization, which saves constrained on-chip area for other components (e.g., LLC).

In this paper, we propose *Cordelia*[1], a secure memory architecture that re-organizes the BMT as a Huffman tree [35], where metadata to authenticate "hotter" addresses is placed closer to the root. The implication of this is that fewer BMT nodes will need to be accessed to authenticate more frequently accessed data, and the global overhead of secure memory is reduced. Achieving this in practice is non-trivial, as the Huffman tree should be adaptable [17, 22, 35, 41] to changing application behavior and overall performance should not be bound by work to construct the new tree shape. We safely and successfully apply these benefits of Huffman encoding to secure memory architectures.

Cordelia achieves adaptability by using the Faller-Gallager-Knuth (FGK) [17, 22, 41, 78] adaptable Huffman tree algorithm. This design decision is important, as using the construction method in the original Huffman paper [35] works best with *a priori* knowledge of global access frequencies and that these access frequencies would be static, which does not apply to the secure memory problem. Cordelia adapts both the FGK and BMT structures for safe integration. Consequently, Cordelia improves performance at runtime by reducing the number of memory accesses associated with frequently accessed addresses. This alleviates the bandwidth requirement needed for secure memory.

We summarize our contributions as follows:

(1) We propose *Cordelia*, a secure memory architecture whose integrity tree is structured as a dynamic, Huffmanized Merkle Tree to reduce metadata accesses.
(2) We adapt the classical FGK dynamic Huffman tree algorithm to enable dynamic restructuring of the BMT.
(3) Our evaluation demonstrates the scalability limitations of the metadata cache and shows that Cordelia benefits performance by 22% relative to the state-of-the-art and 31% relative to commodity secure memory on average and by up to 55% for HPC workloads.

2 Background

This next section provides relevant background for secure memory and summarizes Huffman tree constructions, highlighting the associated challenges.

2.1 Trusted Execution Environments

Trusted execution environments, or TEEs, (e.g., Intel SGX [5], Intel TDX [12], AMD SEV [67], ARM TrustZone [57, 60], RISC-V Keystone [42], and Penglai [18] for RISC-V) propose platform specific secure hardware with developer-facing interfaces to deploy secure applications. Several platforms achieve this via *enclaves* [5, 18, 42], in which applications execute with special privilege on dedicated hardware. This hardware is platform specific and consists of several components, often including a memory encryption engine (MEE) to implement secure memory. While work towards improving the holistic design of these platforms is important, it is orthogonal to our work.

There has been movement away from integrity trees (described in Sec. 2.3) in commodity secure memory hardware. This has been motivated by their performance overheads rather than the importance of their protections. For instance, Intel has claimed the integrity tree, which was deployed in early versions of SGX, is too bandwidth intensive for scalable deployment and that SGXv2 has a larger attack surface to exploit as a result [38]. Hence, a primary goal of this paper is to propose a secure memory protocol that optimizes the integrity tree for performance and bandwidth without compromising on protections for more robust TEEs.

2.2 Encryption

Secure memory enforces data confidentiality by encrypting data prior to its storage using *counter-mode encryption* (CME) [48, 49]. In CME, a unique counter for each data is tracked in memory and used to produce a temporally and spatially unique encryption seed. For encryption and decryption, this seed is fed as an input to an Advanced Encryption Standard (AES) engine. The output from the AES engine is a one-time pad, which can be XOR'ed with plain-text to produce cipher-text or with cipher-text to produce the plain-text.

Instead of keeping a large counter for each 64B word of data, a split counter scheme is used [80], such that each 64B word of data uses a 7-bit minor counter and each 4kB page of data uses a 64-bit major counter. Thus, major and minor counters associated with a page can be stored in a single word. To encrypt a 64B plain-text data word, the associated minor counter is incremented prior to encryption to ensure the temporal uniqueness of the encrypted data. If the minor counter overflows, the associated major counter is incremented and the associated minor counters are reset to zero. All data in the page must be re-encrypted when the major counter is incremented. If the major counter overflows, then the private key used in the AES engine needs to be reset, and all data in memory is to be re-encrypted. Fortunately, major counter overflow is designed to be extremely unlikely (monotonically increasing 64-bit integer).

2.3 Integrity

To protect the integrity of data in memory, the secure memory protocol employs keyed hash message authentication codes (HMAC) [8, 80] and an *integrity tree* (i.e., a Merkle tree [51]) called a Bonsai Merkle tree (BMT) [63]. The BMT, HMACs, and encryption counters (i.e., security metadata) are depicted in Fig. 1. Data integrity is partially protected by maintaining its HMAC; however, the HMAC alone is insufficient as data in memory and its HMAC can be *replayed* to some prior state (threat model described in Sec. 3). Optimizations that relax this requirement of integrity provide insufficient protections against *arbitrary corruption of data* in memory [3, 16]. The integrity of the encryption counters (which are used as part of the key in HMAC computation) is protected via the BMT. The BMT is essentially a tree of hashes; the root of the tree is kept on-chip (i.e., in trusted hardware) to build a root of trust over the state of the encryption counters. By design, the BMT is 8-ary in order to reduce tree height and keep update paths relatively short. This logic, along with the encryption logic, is implemented as an

[1]Cordelia is the youngest daughter in Shakespeare's play *King Lear*, and was the heir deserving of the majority resource allocation due to her behavior.

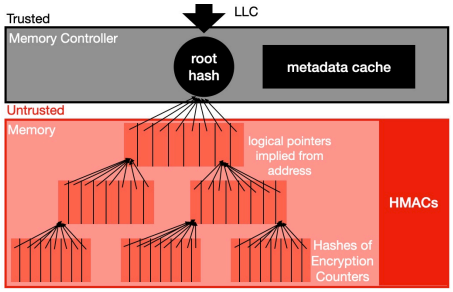


Figure 1: Bonsai Merkle Tree (BMT) is a tree of hashes built from encryption counters. Inner tree nodes are trusted on metadata cache hit.

extension to the memory controller called the memory encryption engine (MEE) [28].

On a data read, the cipher-text, associated encryption counter, leaf-to-root path through the BMT, and HMAC are fetched. The data is decrypted with the encryption counter, and the HMAC is computed from the state of the fetched data and compared to its stored, expected state. To ensure that the counters have not been corrupted, their integrity is verified using the BMT path as detailed below. Conversely, on a data write, all of the relevant metadata needs to be updated atomically to remain consistent with the data. That is, the encryption counter must be updated (as per Sec. 2.2), which requires updating the path through the tree and HMAC. The specific atomicity mechanism is orthogonal to our approach; however, atomic updates can either be achieved via a write-pending queue and ready bit [88] or through undo logging.

State-of-the-art and commodity deployments of secure memory employ a metadata cache to optimize the protocol [6, 11, 19, 20, 30, 34, 45, 61, 77, 81–83, 88]. The optimization is two-fold: ① metadata tends to exhibit locality, so recently accessed metadata values can be retrieved at lower latencies than memory fetches, and ② because values in the metadata cache do not reside in a vulnerable memory device, they can serve as intermediate roots of trust during authentication. A consequence of ② is that the authentication procedure now requires fetching BMT nodes only up to the first metadata cache hit. In other words, the metadata cache *reduces the number of memory fetches for metadata* required to authenticate untrusted data, which alleviates some of the bandwidth pressure of secure memory. Unfortunately, this cache is relatively small (i.e., 64kB in Intel SGX [31]), so the efficacy of this cache may decrease as workloads face increasing memory footprints [15, 24, 32, 84].

2.4 Huffman Trees

A Huffman tree [35] is a binary tree structure where path lengths are proportional to the frequency with which leaves are accessed. The more frequently that a node is accessed, the closer it will be to the root. Huffman [35] presents an algorithm to construct such a tree: two nodes with minimal frequencies are chosen from a priority queue, and a new node is created whose frequency is the sum of their frequencies and which has these two nodes as its children. The newly created node is then inserted into the priority queue

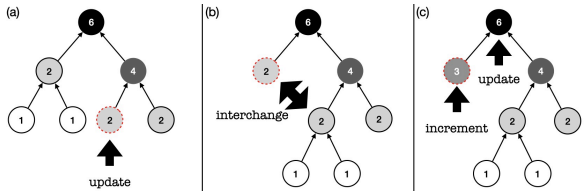


Figure 2: Sample call to `update` in an FGK Huffman tree.

and the process is repeated recursively until a single node remains. This node serves as the Huffman tree root.

While Huffman trees provide an interesting framework for reconsidering BMT structures for secure memory, there are several challenges to directly applying the encoding procedure in [35] to secure memory. For one, a Huffman tree assumes that the access frequencies are globally known *a priori* and static throughout the execution of an application. Such an assumption is unreasonable in the context of memory hardware: applications are highly dynamic in their behavior and memory devices operate agnostic to the behavior of any individual application. To counteract this, one could imagine reconstructing the Huffman tree from time to time. Unfortunately, the $\Theta(n \log n)$ cost to construct a Huffman tree is an arduous operation when n refers to gigabytes or terabytes of physical addresses.

Given this observation, our work considers alternative constructions of Huffman trees. In particular, we focus on *adaptive* protocols, such as the FGK algorithm [17, 22, 41, 78]. The FGK algorithm is based on the characterization of a Huffman tree (described in [41, 78]), in which nodes are strategically organized such that weights are sorted from smallest to greatest as nodes are scanned from the bottom left to the right and upwards (i.e., $\rightarrow, \uparrow, \rightarrow$, etc). Within this organization, any two nodes of the same weight may be swapped (i.e., *interchanged*) without violating this organization. This insight can be leveraged when incrementing a node N (when its frequency is to be updated) to first interchange it with the highest order (i.e., upper-most, right-most) node in the structure prior. That way, after its weight is incremented, the systematic organization of nodes is preserved. Given this configuration, incrementing and adjusting the shape of the tree is achieved with significantly less work per operation as the algorithm takes into account the shape of the prior distribution. The algorithm is described formally by Vitter in [78].

We overview the update protocol with an example, shown in Fig. 2. To update (increment) the weight of a leaf node, FGK calls an `update` operation that applies the following iterative modifications to the tree to retain it as a Huffman tree: starting with the leaf as the current node N (i.e., the red dotted node in Fig. 2a), the subtree of N is "*interchanged*" with the subtree of the highest "ordered" node of the same weight (i.e., the node that is furthest "up and to the right" of the same frequency, shown in Fig. 2b). After interchanging, the weight of node N is incremented. Notice that, because N was the "highest order node" of its frequency prior to incrementing, the "Huffman" arrangement of nodes by frequency is preserved. Then, the process is repeated with the parent of node N becoming the new current node (i.e., Fig. 2c). The procedure stops after incrementing the root's weight.

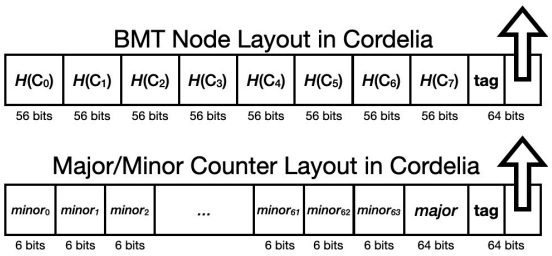


Figure 3: Layout of counters and inner nodes in BMTs to make space for 32 bit tag and 32 bit pointer in Cordelia.

Although the FGK adaptive algorithm provides a strong basis for reimagining the shape of a BMT, a naïve implementation faces several key pitfalls that we address in this paper: ① The original conception of FGK describes a protocol in which the tree shape adapts to *any* change to the distribution of accesses. In the context of secure memory, this means all memory accesses would result in tree reconstruction. ② The FGK algorithm is a fundamentally sequential algorithm, and applying sequential algorithms to parallel environments (such as memory devices) can lead to consistency violations (e.g., authentication failures). ③ Swapping pointers and interchanging nodes may lead to replayable states, which presents a new attack surface for malicious adversaries.

3 Threat Model

We assume a well understood threat model in the context of secure memory [20, 43, 64, 65, 74, 79, 80, 83, 88]: on-chip data is trusted, but attackers can read and tamper with in-memory data, which is therefore untrusted. Furthermore, corruption due to tampering can be precise and specific. We make no assumption about the means by which an attacker can corrupt memory, though realistic mechanisms include Rowhammer [40], RowPress [47], RAS Clobber, etc. Tampering can occur through physical access or through software (i.e., a more capable attacker than in [16]), so a root of trust off of the vulnerable device is necessary.

Secure memory ensures the integrity of data as it crosses the boundary from untrusted to trusted regions. Because caches (including the metadata cache), registers, and other on-chip data stores are not subject to the same physical vulnerabilities, protection of on-chip data and resources is out of the scope of this paper. Furthermore, side channel attacks are out-of-scope in this work, which is consistent with prior secure memory literature [20, 64, 73, 74, 79, 88].

4 Design

We now describe *Cordelia*, a Huffmanized Merkle Tree for secure memory. Our implementation uses a modified FGK adaptive Huffman tree algorithm [17, 22, 41, 78] to the BMT in secure memory. To motivate our proposed modifications, we note several observations and challenges involving FGK and secure memory deployment scenarios.

Observation 1: The key space of the FGK nodes describes the physical address space of the memory, which is fixed overtime.

In Cordelia, the integrity tree is initially constructed as a balanced tree, where all leaves are initialized with access frequency 0. When the system is booted up, physical addresses will be accessed (e.g., frequencies will be updated) which will cause nodes to be interchanged. Notably *no new nodes are created at runtime*, so the storage overhead of Cordelia is a fixed, constant value.

Observation 2: The original conception of the FGK algorithm describes a binary tree, whereas a BMT is an 8-ary tree.

Cordelia implements lightweight changes to the FGK algorithm to perform this modification. We note that, without loss of generality, the FGK algorithm applies so long as "right" child describes "right-most" child for tree arities greater than 2.

Challenge 1: BMT nodes need to maintain a reference to their parent to implement dynamically adaptable paths.

To account for Challenge 1, Cordelia must modify the layout of inner tree nodes to store a reference to their parent. Furthermore, the reference should not provide a new attack surface; any corruption of the reference should be detectable by the authentication procedure. In Sec. 4.2, we discuss how, if done naïvely, such an attack is possible and how Cordelia is resilient to this attack.

Challenge 2: Implementing an FGK tree requires additional ordering metadata, which has storage implications.

It is insufficient to store frequency weights *in tree nodes*, as doing so would occupy space that could otherwise be used to store HMACs or counters. Furthermore, frequency counter overflows were beyond the scope of the theoretical instantiation of FGK, but could have detrimental performance impacts in real hardware. Cordelia proposes a new mechanism to track access frequency counters and Huffman weights that explicitly addresses these pitfalls. We describe how Cordelia does this in Sec. 4.2 and Sec. 4.3.

Challenge 3: The FGK algorithm is a sequential algorithm, but memory devices are highly parallel.

Cordelia implements mechanisms to ensure that the tree shape can safely change without paths changing mid-authentication. This safety mechanism allows for parallel memory accesses and authentications in the device while still having an adaptive tree shape. We describe this mechanism in Sec. 4.4.

4.1 Protocol Overview

Upon receiving a LLC miss (e.g., a memory read), the MEE in Cordelia must authenticate the state of that data. It does so by

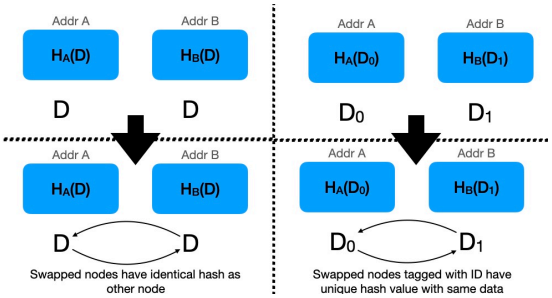


Figure 4: Cordelia tags pointers with a unique ID to ensure different nodes with the same value produce different values, even when residing at the same address.

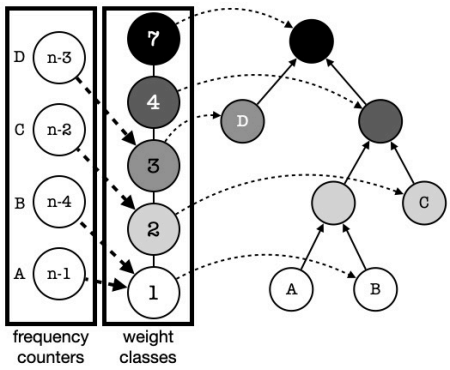


Figure 5: The organization of FGK frequency tracking structures in Cordelia.

issuing a request for the HMAC and encryption counter to the metadata cache, whose addresses are computed as a function of the data address. This is consistent with prior approaches that use a fixed integrity tree shape. If the encryption counter fetch results in a metadata cache miss, then it must be authenticated by fetching against its parent node. Unlike in prior approaches, the parent node of any member of the integrity tree is determined by *following a pointer* in the node (depicted in Fig. 3 with detailed design in Sec. 4.2). This process is repeated recursively until a trusted node is found (in the worst case, the integrity tree root). By implementing a pointer-chasing tree, Cordelia enables dynamic paths, which is a prerequisite for FGK construction.

On a memory write, Cordelia performs two tasks: ① updating the security metadata state to ensure the data and metadata are consistent (as before) and ② update FGK metadata to ensure that the tree shape reflects the memory access pattern. To do ②, Cordelia increments the frequency counter associated with this data (maintained at a per-page granularity). If this frequency counter overflows, then a call to FGK `update` occurs to maintain the appropriate tree shape. This procedure is depicted in Fig. 5 and described in detail in Sec. 4.3 and 4.4.x

4.2 Pointer-Chasing Tree Nodes

We describe how Cordelia accounts for the challenges presented by Challenge 1. In prior BMT-based secure memory schemes, the address of the data and metadata is used to compute the address of the respective encryption counter and parent node [13, 28, 63]. As in prior approaches, the MEE finds the encryption counter associated with a data block by performing a computation on the data address to return a counter address. However, in order to have dynamic paths, BMT nodes in Cordelia need to maintain an explicit reference to their parent, as the parent node may change in calls to FGK `update`. To make space for the reference, we modify the encryption counter and inner integrity tree node structures to maintain a pointer to their parent (depicted in Fig. 3).

In Fig. 3, we demonstrate how space is utilized in Cordelia's integrity tree nodes. Recall that per-page encryption counters are composed of sixty-four 7-bit minor counters and one 8-byte major counter in counter-mode encryption. Cordelia's nodes need 8 bytes of space, and we make this space by reducing the minor counters to 6 bits. To account for this change, Cordelia pads the minor counter field of the AES engine input for encryption/decryption. The 64 bits that are freed as a result of this change are used to store the encryption counter's parent pointer field (see Fig. 3). Inner BMT nodes in Cordelia are composed of eight 7-byte cryptographic hashes to make space for the parent pointer. These structural modifications are intentionally lightweight by design in accordance with Challenge 2.

By reducing minor counter sizes from 7- to 6-bits, Cordelia increases the likelihood of minor counter overflow. On minor counter overflow in CME, the major counter is incremented and all data within the page is re-encrypted. In Cordelia, these updates will occur on frequency counter overflows (detailed in Sec. 4.3). Seeing as re-encryption and tree restructuring are operations that can occur in parallel, Cordelia amortizes the cost of these more frequent re-encryptions within the latency of restructuring operations. In both operations, hash values in inner nodes need to be recomputed, and as such we hide the integrity tree update cost due to re-encryption as being the cost of updating the tree shape in these cases.

Note that the parent reference in Cordelia nodes must be *tagged* with the original node index in the integrity tree to remain robust against replay attacks. To highlight how such an attack could occur, suppose two different nodes in the tree have identical metadata states, as demonstrated in the left side of Fig. 4. Such a scenario implies that interchanging these two different metadata values could be legitimately verified by the same parent. This primitive is dangerous; if different metadata values can be authenticated by the same parent node, the MEE cannot distinguish between legitimate, uncorrupted metadata from maliciously corrupted values. Thus, Cordelia tags the pointers with the original value to ensure that similar data always has a unique identifier per tree node. This is demonstrated on the right side of Fig. 4.

4.3 Frequency Tracking

As conveyed in Challenge 2, storing frequency counters in BMT nodes themselves would restrict the space in the node for security metadata. Thus, Cordelia tracks frequencies in two auxiliary structures: a set frequency counters and a weight-class list. Each

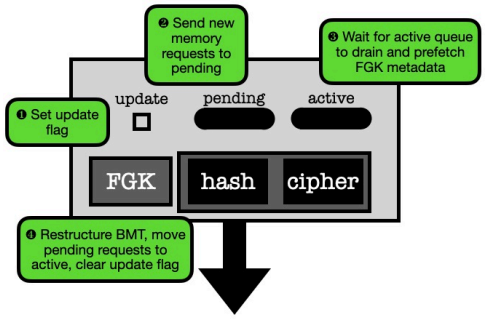


Figure 6: Overview of concurrency control in Cordelia.

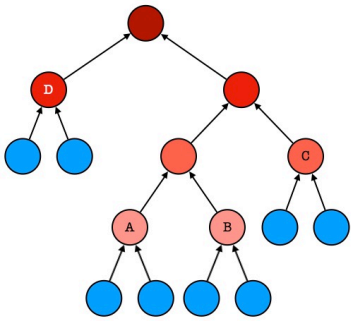


Figure 7: Huffman and non-Huffman regions of a tree with 1 frozen level. Red nodes are "dynamic" and blue nodes are "static."

frequency counter is composed of an n-bit counter value and a reference to an element in the weight-class list. The weight class list is a doubly-linked list, where nodes maintain the weight value of the class, a reference to the highest order (upper-most, right-most) node in the tree, a reference to the next element in the weight-class list, and a reference to the previous element in the weight-class list.

When an integrity tree node associated with an FGK leaf is accessed (BMT leaves by default), Cordelia also increments its associated frequency counter. On frequency counter overflow, the weight class must be incremented, thereby triggering a call to `update` in the FGK algorithm. Additionally, Cordelia must update the auxiliary weight-class list structure. When re-assigning an element from weight class w to a new weight class $w+1$, it is either the case that $w+1$ is in the weight-class list or not. This can be quickly deciphered by checking the next reference from the w element in the list. If so, the frequency counter reference is set to $w+1$. If not, a new element needs to be added to the weight-class list from a free list of weights (not depicted in Fig. 5 for simplicity). In either case, the next and previous references in elements w, $w+1$, and (if w`.next` $!= w+1$) w`.next` must be updated. When removing the element from w, w may become an empty class. Suppose the element that was `updated` has order o before the increment. If $o-1$ is the highest order node of w`.prev`, then w is an empty weight class and may be coalesced into the free list.

The maximum number of possible weights is a function of the number of FGK nodes in the tree. In the worst case, each node could belong to its own weight-class. On the other hand, the number of frequency counters required is exactly the number of leaves in the FGK tree. Therefore, Cordelia reserves space for each of these structures alongside the secure memory metadata at boot. Given Observation 1, the size of this reserved address region is fixed.

4.4 Concurrency for Secure FGK

While coordinating secure memory operations (authentications and updates to tree state) is solved in prior literature [28, 29], Cordelia faces a new challenge in the coordination between normal MEE operations and tree restructuring (updates to tree shape). As described in Challenge 3, the FGK algorithm is an inherently sequential algorithm, so Cordelia must put some mechanism in place to allow authentication and restructuring (i.e., calls to `update` in the FGK tree) not to interfere with one another. Without explicit coordination, paths could change while memory requests are being processed (which would lead to incorrect authentications and path updates). Therefore, Cordelia explicitly coordinates secure memory operations and tree restructuring operations with a reader-writer lock [33] (RW lock). All requests for data and its associated metadata (e.g., LLC misses/writebacks) are classified as "reads" in the context of the RW lock as they do not update the tree shape. Conversely, calls to FGK `update` are "writes." This ensures that normal secure memory operations can operate in parallel[2]. Path updates through the tree can only occur in the absence of normal memory requests (i.e., the RW lock being held by a writer).

As depicted in Fig. 6, the lock is implemented as two queues, for "active" and "pending" requests, and an "update" flag. If the FGK engine decides to restructure the tree shape, it sets the "update" flag. When this flag is set, incoming memory requests from the processor are buffered in the "pending" queue. Once the last request is processed from the "active" queue, the FGK engine issues the requests to update the tree pointers and restructure its shape. Note, the FGK metadata state can be fetched and the updated state can be computed while waiting for the "active" queue to drain. After the FGK updates have responded, the processor requests can be moved from the "pending" queue to "active" queue, and the "update" flag can be cleared so normal execution can resume.

Unfortunately, using this concurrency control can quickly degrade performance. If every memory access calls `update`, then utilizing the RW lock turns Cordelia into a blocking memory. To address this, we relax the notion of "access frequency" as described in Sec. 4.3. Relaxing this definition has the effect of fewer memory accesses triggering the RW lock to block authentications (where the application advances). Consequently, the memory device is able to fetch and authenticate several data requests in parallel. Spending time with the "update" flag set can still lead to pauses in application fetches; however, we observe that, in practice, by using 64 as a default value for n, the typical blocking case only incurs approximately 10-25 memory accesses to re-balance the tree and that the cost of restructuring the shape is amortized by the benefit of fewer metadata accesses. In the worst case, however, the amount of time restructuring the tree is a function of the longest path of

[2]According to the definition in [33], this would mean multiple "readers" hold the RW lock.

Huffman tree nodes. We discuss mitigating the worst-case behavior in Sec. 4.5.

4.5 Worst Case and Spatial Overhead Mitigation

Both the size of Cordelia's auxiliary structures and the worst case bound for time spent blocking is bound by the number of nodes in the Huffman tree. To mitigate this, we propose a scheme in which we relax the granularity at which we track access frequencies by "freezing levels." That is, we define various regions of the tree as being "static" and other regions as "dynamic." Note, according to these definitions, encryption counters are permanently frozen to maintain the mapping from data into the integrity tree. Fig. 7 depicts the organization of these regions. The size of the static region is configurable and can be set by the end-user in the BIOS. This approach is inspired by Awad *et al.* [7], which proposes a similar tree partitioning mechanism for crash consistency.

The space that Cordelia reserves for frequency counters is a function of the number of *Huffman* leaves in the tree. That is, a Huffman leaf is the root of an n-level BMT subtree if n levels are frozen. Given the fan-out property of the 8-ary BMT, "freezing" n lowest levels of the BMT out of the Huffman region decreases the number of Huffman nodes by approximately a factor of 8^n. Reducing the number of leaves means that less space needs to be reserved for the frequency counters and reducing the number of Huffman nodes reduces the amount of space to be reserved to maintain the weight classes (depicted in Fig. 5). In addition, reducing the number of Huffman nodes reduces the longest potential path, which in turn reduces the worst case pause in execution during rebalancing.

Partitioning the BMT into Huffman and non-Huffman regions implies that "Huffman" leaves essentially serve as roots over a static subtree. That is, if Cordelia is configured with more "frozen levels," then the resulting tree is more balanced. Consequently, our mitigation strategy reduces the precision at which access frequencies can be tracked. In practice, however, we see that less granular views of access frequency do not negatively affect Cordelia and, in some cases, actually benefit performance as the work to maintain the precision of the Huffman tree outweighs the benefit promised by this degree of precision.

Table 1: Evaluation framework configuration.

On-Chip Configuration	
Processor	4 cores, x86 ISA, out-of-order 1GHz clock, 1 thread/core
L1 cache	48kB icache, 32kB dcache 2-cycle latency, 64B/block
L2 cache	512kB, 20-cycle latency, 64B/block
L3 cache	8MB, 32-cycle latency, 64B/block
Security Configuration	
BMT	8-ary tree nodes, 64-ary counters
Metadata Cache	64kB, 2-cycle latency
DDR4-based Memory Configuration	
Capacity	64GB, dual channel
Latency	150ns [55], 3200 MT/s [39]

5 Evaluation

We implement Cordelia in gem5 [46] and evaluate it with the hardware configuration specified in Table 1. Our simulator configuration is modeled after prior work [7, 20, 61, 64, 73, 74, 83, 88] for ease of comparison, which is standard methodology. We implement and evaluate four prior arts for comparison: ASSURE [61], VAULT [73], Morphable Counters [64], and Bonsai Merkle Forest [20]. In choosing these baselines for our evaluation, we selected approaches across an array of optimization strategies that have been influential in follow on works (e.g., memory is protected in the Penglai enclave with the integrity tree proposed in [73]). Note, our presented evaluation of [64] only applies to encryption counters as our implementation describes a BMT rather than a tree of counters, but we present evaluation of the full deployment of such a Morphable Tree in Sec. 5.1. We refer to the commodity secure memory deployment from [28] as "gueron-mee" which serves as the baseline approach for comparison. This approach allows us to evaluate the impact of details concerning secure memory on end-to-end application performance. This section also analyzes the spatial overhead of Cordelia and provides a security analysis of our approach.

To demonstrate memory intensive behavior, we perform evaluations using high performance computing workloads from the SPEChpc 2021 benchmark suite [15] a random access microbenchmark (mb), in which random indices are accessed in a large array serving as a memory-intensive adversarial workload for Cordelia (e.g., no re-use locality). In using the SPEChpc benchmarks, we use a 2MB LLC and the "tiny" workload to demonstrate the efficacy of our approach. This is the largest benchmark that we were able to simulate in gem5 given the system requirements of the benchmark, but we shrink the LLC to emulate the increased memory demand of the "small" workloads. We then simulate the region of interest as defined by the source in the benchmark suite. The mb workload preallocates an array, and we measure one billion accesses to random indices of a large array as the region of interest. This application serves as an adversarial worst case workload, as Cordelia depends on application locality to produce skew in the integrity tree.

5.1 Performance

We evaluate the end-to-end runtime of the SPEChpc 2021 benchmark suite and random access microbenchmark (mb) in Fig. 8. The figure shows several metadata cache configurations of both the baseline secure memory protocol and Cordelia to highlight its scalability. In particular, our evaluation entails a "large" metadata cache of 256kB and a "small" metadata cache of 16kB in addition to the metadata cache in a commodity deployment (i.e., 64kB [31]). By using these configurations, we demonstrate the impact of metadata cache size on performance. That is, a smaller metadata cache demonstrates what the expected behavior should be when the cache is more constrained by the active set of metadata. This emulates the effect of future workloads that project to be even more memory intensive.

Cordelia improves performance by 26% relative to ASSURE, 21% relative to VAULT, by 24% relative to Morphable Counters, and by 22% relative to Bonsai Merkle Forest with a commodity metadata cache (i.e., 64kB) on average across the suite. We similarly compare

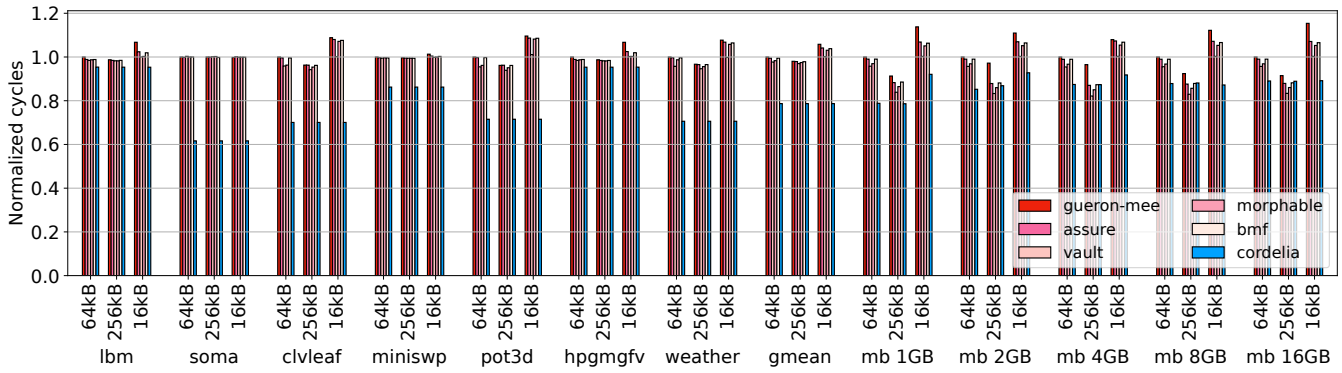


Figure 8: Cycles to execute the region of interest in each of the SPEChpc 2021 suite and random access microbenchmark (mb) with various metadata cache sizes. Results are normalized to the baseline design described in [28] (referred to as `gueron-mee`) with a 64kB metadata cache. Lower is better.

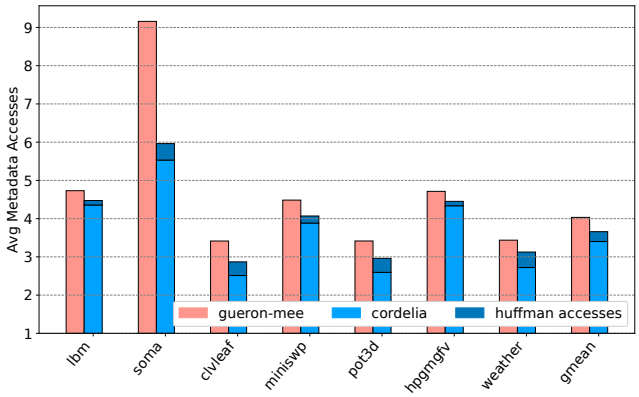


Figure 9: Breakdown of average metadata accesses per data request across the SPEChpc 2021 benchmarks.

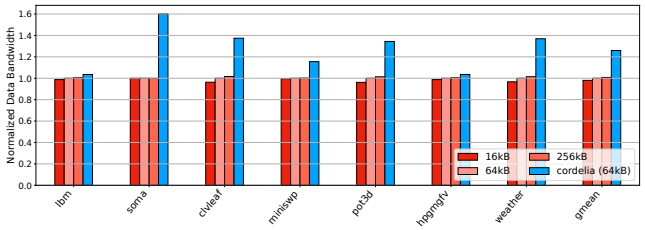


Figure 10: Data bandwidth saved by Cordelia relative to the baseline secure memory protocol [28] with variable metadata cache size.

against a 128-ary tree to emulate the fully Morphable Tree in [64], and find that Cordelia improves performance by 18% relative to this scheme. We attribute this benefit to the fact that Cordelia is able to reduce path lengths more than ASSURE, in which the subtree may only move horizontally. Furthermore, despite the improved utility per metadata cache block in VAULT and Morphable Counters, Cordelia enhances the reduced authentication path length optimization beyond the metadata cache alone.

We find that the baseline from [28] as well as prior works are highly sensitive to metadata cache size. In particular, Morphable Counters [64] outperforms the baseline by 6% with a 64kB metadata cache, but its performance with a 16kB cache is 1% worse than the baseline with a 64kB cache. On the other hand, Cordelia's performance is cache size agnostic for these HPC workloads. This indicates that these workloads may suffer as future workloads become increasingly memory intensive and cache unfriendly.

To further study this effect, we also analyze Cordelia and the prior art against a memory intensive random access microbenchmark (mb). We find that performance improvements due to Cordelia relative to [28] range from 22% with a 1GB workload to 11% with a 16GB workload (i.e., increasingly adversarial) when using a commodity metadata cache. We attribute this phenomenon to the decreasing locality of the workload as the array size increases, so the resulting Huffman tree does not produce as much skew. As we have demonstrated, real world workloads like the HPC suite exhibit greater locality than random accesses, which is leveraged by Cordelia to reduce authentication costs.

We attribute Cordelia's benefits to the reduced number of secure memory metadata fetches as a result of the skewed tree shape. Fig. 9 shows the number of memory fetches for an average authentication required with commodity cache sizes for each of the SPEChpc benchmarks. These are broken down by the authentication metadata (i.e., counter, HMAC, and BMT nodes), and the work to restructure the Huffman tree for Cordelia. The figure shows that, across workloads, the majority of accesses to the memory device are due to authentication accesses to the BMT as opposed to FGK restructuring operations. As such, we conclude that the restructuring of the integrity tree shape is *not a bottleneck* in Cordelia. Furthermore, Cordelia reduces the number of memory accesses by 14% on average across the suite as compared to the baseline

A direct consequence of reducing the number of metadata fetches is increasing the available bandwidth in the memory device. Fig. 10 shows that Cordelia consumes up to 21% less memory bandwidth than the baseline case (`hpgmgfv`), and 13% on average despite the additional work to implement and maintain the Huffman tree (8% of

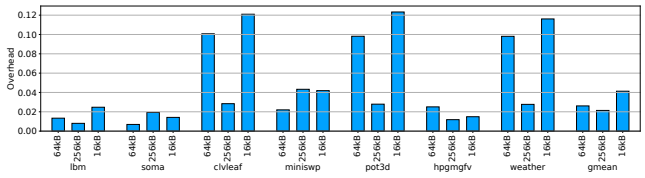


Figure 11: Overhead of balanced pointer-chasing BMT relative to eager fetch (i.e., [28]).

the total memory fetches on average). We find that this reduction in fetches can benefit performance to the extent that these workloads are bound by memory fetches. That is, the performance benefit is greater for workloads (such as `clvleaf` and `pot3d`) in which there is a significant reduction in additional fetches for metadata (16% and 18%, respectively) and the application is highly memory intensive. Additionally, fewer metadata accesses implies reduced authentication work on the critical path of an average memory fetch, so data may be returned more quickly to the processor.

5.2 Feature Analysis

Cordelia implements two features that incur overhead relative to the baseline secure memory protocol: ① a pointer-chasing authentication tree and ② a reader-writer lock to safely update the tree state. In this section, we demonstrate the overhead of these features and to highlight its relatively negligible impact on Cordelia's performance.

Cordelia implements a pointer-chasing integrity tree to allow for dynamic paths through the tree to be formed at runtime. An undesired consequence is that a parent node cannot be fetched until the child is accessed and provides a pointer. To avoid this added latency, we run two static, balanced BMTs: ① fetches parent nodes eagerly (i.e., the baseline from [28]) and ② fetches parent nodes once the child data is returned (i.e., Cordelia). To isolate the cost of pointer-chasing, we do not evaluate the cost of reconstruction in this study.

We find that, Cordelia's pointer-chasing tree design incurs less than 5% overhead relative to a tree with eager fetches on average. We consider this overhead negligible relative to the benefits promised by the shorter authentication path. In the worst case (`pot3d`), the overhead on a per-operation basis can be up to 12%. However, the average case requires paying for the overhead of a metadata cache response relative to a tag lookup, whose overhead is largely hidden by slower encryption and authentication operations.

To study the impact of the reader-writer lock described in Sec. 4.4, we demonstrate the average memory bandwidth of data accesses (e.g., data throughput without accounting for metadata accesses) in Cordelia relative to the baseline secure memory protocol proposed in [28]. Our results are depicted in Fig. 10. We find that data bandwidth *improves* in Cordelia by 26% compared to the baseline protocol with a comparable cache (64kB) on average. Moreover, it achieves up to 24% better data bandwidth than [28] with a much larger cache (256kB). This result exemplifies that Cordelia is not bound by any memory system blocking due to synchronization, and that the cost of restructuring is amortized by the benefit of generating skew.

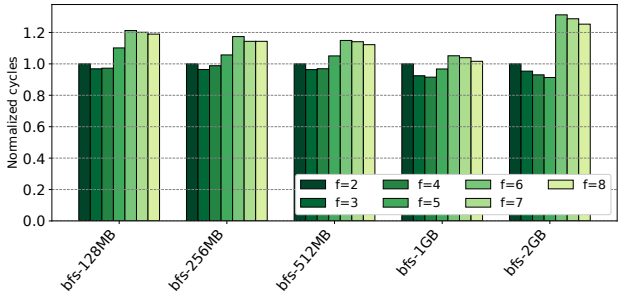


Figure 12: Sensitivity of number of frozen levels, f, while performing BFS in graph500.

5.3 Spatial Overhead

Cordelia maintains two auxiliary structures in order to implement the FGK algorithm: ① the frequency counters associated with each leaf and ② the list of node classes. Each of these structures requires an explicitly reserved region of physical memory where they are stored, and their size is a function of the number of nodes in the Huffman tree.

To implement relaxed updates in the FGK part of the tree, Cordelia tracks how many times a particular Huffman leaf has been updated between 0 and n, and maintains a reference to that leaf's associated weight class. By default, we assume n to be 64 (i.e., 6 bits of storage), and as described below there are t possible weight classes where t is approximately 300,000 for a 64GB memory (i.e., 19 bits of storage). Thus, each of the 262144 frequency counters in a 64GB can be expressed as an 4-byte field, resulting in 1MB to store the frequency counters.

Elements in the node class list maintain a reference to the highest order node in the weight class, a reference to the next greatest weight in the node class list, and a counter to maintain the current frequency for that class. The number of elements in the weight class list is equal to the number of Huffman nodes to account for the worst case (i.e., each node resides in its own weight class). Seeing as there are approximately 300,000 Huffman nodes in a 64GB memory, this means that each of these references can be expressed as 19-bit values. In order to ensure that elements can be neatly word aligned, we store 90 bits for the frequency counter. Thus, the size of each element in the weight class list is 16-bytes, and the total size of the list is approximately 4.57MB.

As described in Sec. 4.5, the spatial overhead of the auxiliary structures in FGK are largely a function of the number of Huffman leaves in the tree. By freezing the f lowest levels of the Huffman tree, we can reduce t by a factor of 8^{f-1} given the 8-ary nature of the BMT. That is, merely by setting f to 1 reduces the spatial overhead of Cordelia from 5.57MB to 714kB, setting $f = 2$ requires 90kB, etc.

5.4 Impact of Worst Case Mitigation

As described in Sec. 4.5, the worst case behaviors in Cordelia are largely a function of the number of nodes in the Huffman region of the tree. Freezing the f lowest levels of the Huffman tree can remedy some of these undesirable behaviors, but the impact of this decision

on performance is non-trivial. In particular, freezing the f lowest levels of the BMT out of the Huffman tree essentially trades off the precision at which Cordelia can track access frequencies with the spatial overhead required to perform such tracking. Surprisingly, we also see that there can be positive implications of freezing levels on performance beyond the theoretical case.

To highlight the impact of this design decision, in Fig. 12 we compare the normalized cycles to run the graph500 BFS workloads in Cordelia without a metadata cache using different levels for f. As demonstrated by the evaluation, increasing f can have a positive impact on performance to a point, especially for workloads with larger memory footprints. We find that by reducing the number of Huffman nodes, the amount of work for the FGK algorithm to do is reduced. That is, when applications access memory more uniformly, the resulting Huffman tree is more balanced. Thus, increased precision does not benefit performance in these cases. Consequently, reducing the number of Huffman nodes from Cordelia reduces the amount of work per call to `increment` in the FGK algorithm. This means that Cordelia can spend a greater percentage of time performing authentication. On the other hand, if too much precision is lost (i.e., $f = 6$ while performing BFS in a 2GB graph), then there can be a regression in performance as Cordelia cannot create enough skew to benefit performance.

5.5 Security Analysis

We base the methodology for our security analysis on prior secure memory literature [64, 73]. To the best of our knowledge, these serve as the standard methodology in analyzing the security of this literature. In particular, the analysis is predicated on the fact that the secure memory protocol is secure because it is highly unlikely that two non-malicious values recursively produce colliding hashes through the BMT. This is an important primitive, as a strong cryptographic hashing algorithm is one that randomly maps any data to a hash value with uniform probability, thereby making it impossible to compute a colliding hash value using brute force in a reasonable amount of time. When producing an 64-bit cryptographic hash, there is a probability p that two values will collide where p is $\frac{1}{2^{64}}$. In the baseline, when applying this to a BMT protecting 64GB, the resulting tree is ten levels (i.e., there are ten of these cryptographic hashes produced that protect this data). Thus, the probability that a particular path through the tree is produced for some particular data value is p^{10} or $(\frac{1}{2^{64}})^{10}$. The likelihood that two independent values will have colliding hashes is $p^{10} \times p^{10} = p^{20}$, or 4.8×10^{-386}.

Cordelia modifies the hash size by producing a 56-bit cryptographic hash (as opposed to 64-bits) and the shortest possible path from leaf-to-root is 2 (the most skew being leaf referring directly to the root), regardless of memory size. As a result, the probability that two hash values will collide in Cordelia is p', where p' is $1/2^{56}$ and the probability that two paths through the tree may collide is as high as p'^2. Thus, the likelihood that two independent values will have a colliding hash is $p'^2 \times p'^2 = p'^4$ or 3.71×10^{-68}. While this is a significant reduction in the number of values to test for colliding hashes, the task of finding colliding hashes in Cordelia is still sufficiently difficult to uphold the secure memory properties. That is, if an adversary could try 1000 data values per second (consistent attacker capability as prior work [64]), it would take approximately 8.55×10^{62} years to come up with a fully colliding data path through the tree by brute force. Furthermore, this will be even increasingly difficult with more levels frozen from the Huffman part of the tree at the cost of some degree of adaptability.

6 Related Work

There are several other works in the prior secure memory literature that have proposed dynamic integrity trees and/or shorter authentication paths through the BMT. We overview the related work in this area, and with respect to Huffman trees below.

6.1 Secure Memory

We have reviewed prior works by Suh et al. [69] and Rogers et al. [63] extensively. Some prior literature has proposed co-designing HMACs with ECC [65], compressing infrequently used encryption counters [64], and compressing HMACs [73] to increase the utility per metadata word. The aim of these works is to improve metadata cache utilization. Unfortunately, these works only provide modest improvements, and cache adversarial workloads will still result in a degradation of performance. Alternatively, per-application secure memory [72] allows for smaller trees, but limits the software that can benefit from this approach. On the other hand, Cordelia's optimization is cache-agnostic; it benefits from the cache when the cache is useful, but improves the underlying BMT data structure as well for cache adversarial cases. Another approach has been to eliminate the BMT entirely from secure memory [16]. Such a scheme, however, assumes that all memory traffic occurs across the memory bus (i.e., a less capable adversary). Cordelia considers protection against arbitrary corruption, in which a BMT is required.

Beyond these, other prior works have proposed maintaining a "fast" Merkle tree outside of the "true" BMT for frequently accessed data [4, 79]. These works depend on tracking whether or not certain pages are "hotter" than some threshold, and use a pointer to direct authentications to the fast tree. Like Cordelia, these works leverage the fact that the spatial overhead of BMT is much less than that of the HMACs, and use additional auxiliary structures to accelerate the secure memory protocol. However, unlike Cordelia, this approach limits the degrees to which a particular node is or isn't hot in the tree. That is, if a node in the fast subtree is accessed more frequently than other "hot" nodes in the fast subtree, its path is still bound by the size of the fast subtree.

Other works have considered creating shorter paths throu-gh the integrity tree by keeping more subtree roots on-chip [20, 61, 74]. More frequently accessed subtrees can be "cached" in one of these on-chip roots. BMF [20] does this by maintaining several roots and swapping a root for its child if the child becomes the hottest. Unfortunately, such an approach faces the same scalability issues as the metadata cache. ASSURE [61] moves a fast subtree horizontally to cover the most frequently accessed subtree root. AMNT [74] tracks a single fast subtree, but is primarily concerned with persistence protocols. However, these approaches take a less precise view of "hotness," and can only reduce authentication paths by a fixed amount as a result. Cordelia, on the other hand, takes a global view of "hot" addresses and ensures they have the shortest authentication path as defined by a Huffman tree.

Szefer and Biedermann proposed a skewed BMT [71]. Such a structure would have shorter paths for the upper and lower addresses, which are those that are hot on initialization. They also propose several other potential tree shapes, but do not propose a scheme for where the BMT can adapt to these shapes. On the other hand, Cordelia allows for a BMT that changes shape dynamically to adapt to changing program behaviors.

Triad-NVM [7] proposes a means of partitioning the BMT by level to determine how to persist metadata. It proposes persisting lower levels strictly, and using a relaxed scheme for upper levels based on the tolerance for recovery relative to performance runtime. In Cordelia, we describe a similar approach to partitioning the integrity tree in our evaluation, where not all levels need to belong to the Huffman tree in order to reduce the reconstruction cost or in-memory overhead of the FGK structures.

6.2 Huffman Trees

Our work heavily depends on the FGK algorithm to construct an adaptive Huffman tree. Vitter's algorithm [78] is an alternative adaptive Huffman tree protocol. This algorithm is used to optimize the worst case behavior, but incurs some practical online overheads in terms of construction. Pigeon and Bengio [59] explored ways to reduce the memory cost of the FGK approach.

A few works have looked into Huffmanized Merkle trees for secure memory [52, 53, 76, 87]. While these works serve as influential prior contributions for Cordelia, they are limited by the lack of coordination between tree shape updates and authentications. As such, authentications will suffer unacceptable false negatives (i.e., failures without tampering). Unlike [53, 76], Cordelia addresses this with explicit concurrency control (Sec. 4.4), and additionally contributes a novel frequency tracking mechanism (Sec. 4.3), storage overhead mitigation (Sec. 4.5), and analyzes the amortization of construction cost (Sec. 5.4). Cordelia explicitly addresses each of these challenges, making its deployment practical. Furthermore, Cordelia also considers the scalability of its approach by considering various schemes for incrementing counters and freezing levels out of the Huffman region.

6.3 Trusted Execution and Confidential Computing

Vendors have deployed various secure hardware platforms for developers to leverage in their application development. Generally, these platforms may be categorized as one of Trusted Execution Environments (TEEs) or Confidential Computing platforms depending on their guarantees. Confidential computing merely guarantees that application state is encrypted, which relaxes the runtime overhead of the secure platform. However, confidential computing platforms are subject to random-corruption based attacks, which have been demonstrated to be a power attack surface [9]. On the other hand, TEEs describe platforms that guarantee the privacy, integrity, and isolation of applications running in the environment (typically an enclave). As described in Sec. 2.1, TEEs are the natural application of secure memory, as TEEs built on top of a non-secure memory are vulnerable to application corruption via the physical vulnerabilities of main memory devices [37].

Among deployed TEEs, there exist several strategies to implement secure memory. Early versions of Intel SGX [5, 28] and RISC-V based enclaves [18, 42] use a MEE to encrypt enclave data prior to its storage in main memory and use MACs and a fixed, balanced integrity tree to authenticate its state. The MEE in Intel SGXv2, Intel TDX [12], and AMD SEV [67] encrypt all data, both inside and outside of the enclave, to ensure its confidentiality in main memory [26, 62]. Neither Intel TDX nor AMD SEV make guarantees over the integrity of data in memory from physical corruption, whereas SGXv2 authenticates enclave data using MACs without an integrity tree. As such, data in memory is vulnerable to undetected corruption via replay attack [63, 69, 80]. The ARM TrustZone specification does not explicitly include language about physical memory protection, so MEE design is left to the platform developer [54].

7 Conclusion

Secure memory is a known bottleneck in memory intensive workloads. In this paper, we have proposed Cordelia, an alternative approach to implement secure memory by reconsidering the construction of its underlying data structures. We show that by considering the data structure as a means to optimize the secure memory protocol, we can improve performance of the system in a more scalable manner than by focusing on the metadata cache. Beyond this, reducing the authentication path length may have positive implications for emerging technologies, like non-volatile memories, which have limited device endurance.

References

[1] Martin Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. 2009. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)* 13, 1 (2009), 1–40.

[2] Andrew Adiletta, Caner Tol, and Berk Sunar. 2024. LeapFrog: The Rowhammer Instruction Skip Attack. arXiv:2404.07878 [cs.CR]

[3] Shaizeen Aga and Satish Narayanasamy. 2017. Invisimem: Smart memory defenses for memory bus side channel. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 94–106.

[4] Mazen Alwadi, Aziz Mohaisen, and Amro Awad. 2021. Promt: optimizing integrity tree updates for write-intensive pages in secure nvms. In *Proceedings of the ACM International Conference on Supercomputing*. 479–490.

[5] Ittai Anati, Frank Mckeen, Shay Gueron, Huang Haitao, Simon Johnson, Rebekah Leslie-Hurd, Harish Patil, Carlos Rozas, and Hisham Shafi. 2015. Intel Software Guard Extensions (Intel SGX). In *Tutorial at International Symposium on Computer Architecture (ISCA)*.

[6] Amro Awad, Suboh Suboh, Mao Ye, Kazi Abu Zubair, and Mazen Al-Wadi. 2019. Persistently-secure processors: Challenges and opportunities for securing non-volatile memories. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 610–614.

[7] Amro Awad, Mao Ye, Yan Solihin, Laurent Njilla, and Kazi Abu Zubair. 2019. Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture*. 104–115.

[8] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1996. Keying hash functions for message authentication. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1–15.

[9] Rodrigo Rubira Branco and Shay Gueron. 2017. Memory protection challenges: Attacks on memory encryption. *Cyber Security: A Peer-Reviewed Journal* 1, 3 (2017), 199–206.

[10] Nathan Burow, Scott A Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. 2017. Control-flow integrity: Precision, security, and performance. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 1–33.

[11] Zhengguo Chen, Youtao Zhang, and Nong Xiao. 2020. Cachetree: Reducing integrity verification overhead of secure nonvolatile memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 7 (2020), 1340–1353.

[12] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel tdx

demystified: A top-down approach. *Comput. Surveys* 56, 9 (2024), 1–33.
[13] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).
[14] Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen, Ahmad-Reza Sadeghi, Stefan Brunthaler, and Michael Franz. 2015. Readactor: Practical code randomization resilient to memory disclosure. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 763–780.
[15] Nan Ding, Pieter Maris, Hai Ah Nam, Taylor Groves, Muaaz Gul Awan, LeAnn Lindsey, Christopher Daley, Oguz Selvitopi, Leonid Oliker, Nicholas Wright, et al. 2023. Evaluating the potential of disaggregated memory systems for HPC applications. *Concurrency and Computation: Practice and Experience* (2023), e8147.
[16] Ali Fakhrzadehgan, Prakash Ramrakhyani, Moinuddin K Qureshi, and Mattan Erez. 2023. SecDDR: Enabling low-cost secure memories by protecting the DDR interface. In *2023 53rd Annual IEEE/IfiP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 14–27.
[17] Newton Faller. 1973. An adaptive system for data compression. In *Record of the 7-th Asilomar Conference on Circuits, Systems and Computers*. 593–597.
[18] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the PENGLAI Enclave.. In *OSDI*. 275–294.
[19] Alexander Freij, Shougang Yuan, Huiyang Zhou, and Yan Solihin. 2020. Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Memory. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 14–27.
[20] Alexander Freij, Huiyang Zhou, and Yan Solihin. 2021. Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1227–1240.
[21] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 747–762.
[22] Robert Gallager. 1978. Variations on a theme by Huffman. *IEEE Transactions on Information Theory* 24, 6 (1978), 668–674.
[23] Blaise Gassend, G Edward Suh, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. 2003. Caches and hash trees for efficient memory integrity verification. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*.
[24] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. AI and memory wall. *IEEE Micro* (2024).
[25] CR Glassey and RM Karp. 1976. On the optimality of Huffman trees. *SIAM J. Appl. Math.* 31, 2 (1976), 368–378.
[26] Tallas Tian Sheng Goo. 2023. Intel Total Memory Encryption: Functional Verification and Performance Analysis. (2023).
[27] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. 2018. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 245–261.
[28] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. *Proc. International Association for Cryptologic Research (IACR)* (2016).
[29] W Eric Hall and Charanjit S Jutla. 2006. Parallelizable authentication trees. In *Selected Areas in Cryptography: 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers 12*. Springer, 95–109.
[30] Xijing Han, James Tuck, and Amro Awad. 2021. Dolos: Improving the performance of persistent applications in adr-supported secure memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1241–1253.
[31] Youngkwang Han and John Kim. 2019. A novel covert channel attack using memory encryption engine cache. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
[32] Jeongmin Hong, Sungjun Cho, Geonwoo Park, Wonhyuk Yang, Young-Ho Gong, and Gwangsun Kim. 2024. Bandwidth-Effective DRAM Cache for GPU s with Storage-Class Memory. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 139–155.
[33] William C Hsieh and William E Weihl. 1991. Scalable reader-writer locks for parallel systems. (1991).
[34] Jianming Huang and Yu Hua. 2021. 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, 359–370.
[35] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.
[36] Jisung Im, Hansol Kim, Jinsu Kim, Seungmin Woo, Taeseong Kwon, Young Jun Yoon, Jong-Ho Bae, and Sung Yun Woo. 2024. Analysis of Row Hammer and Passing Gate Effect in DRAM Cells by BCAT Structural Design. In *2024 IEEE Silicon Nanoelectronics Workshop (SNW)*. IEEE, 91–92.
[37] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking down the processor via Rowhammer attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 1–6.
[38] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlata. [n. d.]. Supporting Intel SGX on Multi-Socket Platforms. Accessed: 2023-11-28. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf
[39] Dongkyun Kim, Minsu Park, Sungchun Jang, Jun-Yong Song, Hankyu Chi, Geunho Choi, Sunmyung Choi, Changhyun Kim, Minsik Han, Kibong Koo, et al. 2019. A 1.1-V 10-nm class 6.4-Gb/s/pin 16-Gb DDR5 SDRAM with a phase rotator-ILO DLL, high-speed SerDes, and DFE/ffE equalization scheme for Rx/Tx. *IEEE Journal of Solid-State Circuits* 55, 1 (2019), 167–177.
[40] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.
[41] Donald E Knuth. 1985. Dynamic huffman coding. *Journal of algorithms* 6, 2 (1985), 163–180.
[42] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the fifteenth European Conference on Computer Systems*. 1–16.
[43] Tamara Silbergleit Lehman, Andrew Douglas Hilton, and Benjamin C. Lee. 2016. PoisonIvy: Safe speculation for secure memory. In *Proc. International Symposium on Microarchitecture (MICRO)*.
[44] Tamara Silbergleit Lehman, Andrew Douglas Hilton, and Benjamin C. Lee. 2018. MAPS: Understanding Metadata Access Patterns in Secure Memory. In *Proc. International Symposium on Performance Analysis of Systems and Software (ISPASS)*.
[45] Sihang Liu, Korakit Seemakhupt, Gennady Pekhimenko, Aasheesh Kolli, and Samira Khan. 2019. Janus: Optimizing memory and storage support for non-volatile memory systems. In *Proceedings of the 46th International Symposium on Computer Architecture*. 143–156.
[46] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).
[47] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. 2023. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–18.
[48] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). *submission to NIST Modes of Operation Process* 20 (2004), 0278–0070.
[49] David A McGrew. 2002. Counter mode security: Analysis and recommendations. *Cisco Systems, November* 2, 4 (2002).
[50] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad-Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. 2016. The cybersecurity landscape in industrial control systems. *Proc. IEEE* 104, 5 (2016), 1039–1057.
[51] Ralph C Merkle. 1980. Protocols for Public Key Cryptosystems.. In *Proc. Symposium on Security and Privacy (SP)*.
[52] Matthew Millar, Marcin Łukowiak, and Stanislaw Radziszowski. 2022. Memory Protection with Dynamic Authentication Trees. In *2022 29th International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*. IEEE, 202–207.
[53] Matthew T Millar. 2021. *flexible Memory Protection with Dynamic Authentication Trees*. Rochester Institute of Technology.
[54] Muhammad Asim Mukhtar, Muhammad Khurram Bhatti, and Guy Gogniat. 2019. Architectures for Security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone. In *2019 2nd International Conference on Communication, computing and Digital systems (C-CODE)*. IEEE, 299–304.
[55] Janani Mukundan, Hillery Hunter, Kyu-hyoun Kim, Jeffrey Stuecheli, and José F Martínez. 2013. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 48–59.
[56] Onur Mutlu and Jeremie S Kim. 2019. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2019), 1555–1571.
[57] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. 2016. Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 445–451.
[58] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. 2016. A survey on systems security metrics. *ACM Computing Surveys (CSUR)* 49, 4 (2016), 1–35.
[59] Steven Pigeon and Yoshua Bengio. 1997. A memory-efficient Huffman adaptive coding algorithm for very large sets of symbols. *Université de Montréal, Rapport technique* 1081 (1997).
[60] Sandro Pinto and Nuno Santos. 2019. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)* 51, 6 (2019), 1–36.

[61] Joydeep Rakshit and Kartik Mohanram. 2017. Assure: Authentication scheme for secure energy efficient non-volatile memories. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.

[62] Charles W Rockett. 2024. Performance Analysis of Intel Total Memory Encryption-Multi-Key. (2024).

[63] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and Bonsai Merkle trees to make secure processors OS- and performance-friendly. In *Proc. International Symposium on Microarchitecture (MICRO)*.

[64] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A Joao, and Moinuddin K Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *Proc. International Symposium on Microarchitecture (MICRO)*.

[65] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K Qureshi. 2018. SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*.

[66] Felix Schuster, Thomas Tendyck, Christopher Liebchen, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. 2015. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 745–762.

[67] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53, 2020 (2020), 1450–1465.

[68] Daniele Sgandurra and Emil Lupu. 2016. Evolution of attacks, threat models, and solutions for virtualized systems. *ACM Computing Surveys (CSUR)* 48, 3 (2016), 1–38.

[69] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. Efficient memory integrity verification and encryption for secure processors. In *Proc. International Symposium on Microarchitecture (MICRO)*.

[70] Yumeng Sun, Xiang Liu, Noakim Wang, Jongsung Jeon, Blacksmith Wu, and Kanyu Cao. 2021. Trap-assisted passing word line leakage and variable retention time in DRAM. In *2021 IEEE 4th International Conference on Electronics Technology (ICET)*. IEEE, 338–341.

[71] Jakub Szefer and Sebastian Biedermann. 2014. Towards fast hardware memory integrity checking with skewed merkle trees. In *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. 1–8.

[72] Meysam Taassori, Rajeev Balasubramonian, Siddhartha Chhabra, Alaa R Alameldeen, Manjula Peddireddy, Rajat Agarwal, and Ryan Stutsman. 2020. Compact leakage-free support for integrity and reliability. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 735–748.

[73] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 665–678.

[74] Samuel Thomas, Kidus Workneh, Jac McCarty, Joseph Izraelevitz, Tamara Lehman, and R. Iris Bahar. 2024. A Midsummer Night's Tree: Efficient and High Performance Secure SCM. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24)*.

[75] Jan Van Leeuwen. 1976. On the Construction of Huffman Trees.. In *ICALP*. 382–410.

[76] Saru Vig, Rohan Juneja, Guiyuan Jiang, Siew-Kei Lam, and Changhai Ou. 2019. Framework for fast memory authentication using dynamically skewed integrity tree. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 10 (2019), 2331–2343.

[77] Saru Vig, Rohan Juneja, and Siew-Kei Lam. 2020. DISSECT: dynamic skew-and-split tree for memory authentication. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1538–1543.

[78] Jeffrey Scott Vitter. 1987. Design and analysis of dynamic Huffman codes. *Journal of the ACM (JACM)* 34, 4 (1987), 825–845.

[79] Bolin Wang. 2022. *Bo-tree: a dynamic merkle tree for enabling scalable memories*. Ph. D. Dissertation. University of British Columbia.

[80] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving cost, performance, and security of memory encryption and authentication. *Proc. International Symposium on Computer Architecture (ISCA)* (2006).

[81] Fan Yang, Youmin Chen, Haiyu Mao, Youyou Lu, and Jiwu Shu. 2020. ShieldNVM: An Efficient and Fast Recoverable System for Secure Non-Volatile Memory. *ACM Transactions on Storage (TOS)* 16, 2 (2020), 1–31.

[82] Fan Yang, Youyou Lu, Youmin Chen, Haiyu Mao, and Jiwu Shu. 2019. No compromises: Secure nvm with crash consistency, write-efficiency and high-performance. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[83] Mao Ye, Clayton Hughes, and Amro Awad. 2018. *Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[84] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2024. Deep learning workload scheduling in gpu datacenters: A survey. *Comput. Surveys* 56, 6 (2024), 1–38.

[85] Keun Soo Yim. 2016. The rowhammer attack injection methodology. In *2016 IEEE 35th symposium on reliable distributed systems (SRDS)*. IEEE, 1–10.

[86] Ranyang Zhou, Jacqueline T Liu, Sabbir Ahmed, Shaahin Angizi, and Adnan Siraj Rakin. 2024. Compromising the Intelligence of Modern DNNs: On the Effectiveness of Targeted RowPress. *arXiv preprint arXiv:2412.02156* (2024).

[87] Yu Zou and Mingjie Lin. 2019. Fast: A frequency-aware skewed merkle tree for fpga-secured embedded systems. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 326–331.

[88] Kazi Abu Zubair and Amro Awad. 2019. Anubis: ultra-low overhead and recovery time for secure non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture*. 157–168.