Survey of Persistent Memory Security

Genoveva Fossas

April 30, 2021

1 Introduction

Persistent memory, also known as non-volatile main memory (NVMM), is a recent development in memory technology that provides the byte-addressability of RAM and the durability of of disk drives. This technology is a promising solution for workloads that would benefit from high-performance recoverable memory, such as databases, long-running scientific computations, or cloud hosting services. Intel, for example, lists Cisco, Huawei Cloud, and Oracle (among others) as customers for their Optane NVDIMMs. [8]

Current research on NVMM spans several areas. Much of the research involves methods to improve the performance of the memory while still maintaining guaranteed recoverability. [7, 11, 4] However, there are also several papers that address security risks associated with NVMM. Given that NVMM is an attractive solution for workloads that may handle sensitive data (such as customer data in a cloud application), it is important to fully address and investigate the security vulnerabilities that NVMM is susceptible to. While there have been several papers addressing the security of NVMM, there has only been one survey paper that has outlined the state of research in the area. [19] This paper was published in early 2020, so it has missed some new impactful developments in the area, necessitating a new survey to accurately portray the area.

This paper addresses and outlines the current research regarding non-volatile memory security and its themes, providing critique and making connections to other work where appropriate. The paper will begin with a background section that explains all necessary information to place the research into context, followed by sections defined by the vulnerabilities that the given research addresses. The vulnerabilities are as follows:

1. Passive vulnerabilities, or vulnerabilities that don't require any extra tampering on the part of the attacker to carry out. This category primarily includes data remanence attacks.

2. Active vulnerabilities, or vulnerabilities that generally involve an attacker snooping on a running program or machine to manipulate or obtain data.

Each of these sections will discuss the papers that address these security concerns in depth. The paper concludes with a summary of open problems in secure persistent memory.

2 Background

2.1 Persistent Memory

Persistent memory, also known as non-volatile main memory, is an emerging memory technology that combines fast byte-addressable memory with persistence so that data is not lost on a crash. NVMM is considered to be a contender for main memory due to its recoverability and speed. [9, 17] The initial forms of this technology included storage-class memory such as phase-change memory, STT-RAM, and memresistors. [13] However, the most modern form of non-volatile memory is technologies similar to that of Intel's Optane DC Persistent Memory. The primary usage paradigms of persistent memory involve using it as storage for a file system or wrapping data structures in an object so that it may be independently stored in memory. [17]

The ability to recover upon a crash is reliant on strict write atomicity requirements. These atomicity requirements keep data consistent, which prevents memory from being corrupted due to partial writes before a crash. Memory persistency allows for reasoning about the write order of data, which can be used to ensure data consistency. The methodologies to uphold memory consistency generally involve ordering of writes and fencing so that data updates atomically, and there are several models that present differing methodologies of doing so. [11, 13, 7] Persistent memory also requires that the NVMM's data be mapped into the address space of the program using it, especially in the case where it is being used independently from a file system.

2.2 Memory Vulnerabilities

2.2.1 Passive Vulnerabilities

While memory vulnerabilities are among some of the oldest attack vectors, they still pose an issue for persistent memory. The non-volatility of persistent memory means that the NVDIMMs are vulnerable to cold boot attacks more than standard DIMMs. [12] Cold boot attacks traditionally involve an attacker with physical access to the hardware cooling the DRAM in the machine so that the data contained in it will remain for a few minutes after shutdown. In those few minutes, attackers plug the DIMMS into their machine and stream out any remaining data. In the case of NVDIMMs, cooling is no longer required to maintain the persistence of the data contained. Adequate physical security is necessary to prevent this scenario, but it cannot be the only line of defense. Encryption of the data in NVMM is thus a popular consideration, especially given that encryption has been demonstrated as effective and performant in secure processor systems. [15]

However, encryption is not totally invulnerable, much like any security measure. Encryption can be vulnerable to replay attacks, in which an attacker uses previously sent messages during the execution of a program to break encryption. [1] Solutions to this issue has involved tagging messages with types and implicit typing through functions in general cases, [1] but persistent solutions tend to involve verifying the integrity of data before it arrives on-chip. [6]

2.2.2 Active Vulnerabilities

Persistent memory is not only vulnerable to data remanence attacks, however. Persistent memory is also vulnerable to unauthorized memory disclosure and corruption. While it is vulnerable to these issues in the way any program is if it is not properly protected by the operating system's access control, it is also uniquely vulnerable due to its constant address in physical memory. The standard solutions to ensure authorized access and anonymity involve operating system level access control and intra process isolation. [18]

3 Passive Vulnerability Mitigation

The papers addressing data remanence attacks include encryption at the architecture or hardware level [3, 5], with the exception of the paper written by Pan et al. [12]

3.1 Architecture-Level Encryption

The solutions involving architecture-level encryption, i-NVMM [5] and SuperMem [21] have a few approaches to encrypting the data in NVMM. i-NVMM utilizes incremental encryption, meaning that pages are encrypted and decrypted as a program is executing. A program's resident set is encrypted by a memory-side encryption engine, while the working set is left decrypted. [5] SuperMem, in contrast, utilizes counter mode encryption embedded in each write to memory. The encryption, done in the memory controller, utilizes an AES algorithm that takes a memory line number and a per-line counter in order to produce a unique one-time pad (OTP) for each line written to memory. [21]

Due to their very different natures, both approaches have unique cruxes to their approach. The primary determination of performance in i-NVMM relies on how often its built-in inert page predictor mispredicts that a page is inert (and thus can be encrypted). Frequent misprediction results in significant overheads, as the program must wait for the part of the page it is requesting to be decrypted. While it takes less time for only part of the page to be decrypted, several accesses to this page will cause the performance overheads to accrue. [5] There is a method for mispredicted pages to be fully decrypted, but it can take upwards of hundreds of thousands of accesses before a page is considered active and deserving of decryption. How often misprediction occurs can be somewhat controlled through the variables associated with the memory scan frequency and the inactivity threshold before marking a page inert. However, too low of a value for either of these variables results in low misprediction at the cost of a small percentage of the memory being encrypted. [5] When set at the moderate levels suggested by the authors, however, the misprediction rate roughly hovers around 21% with 81% memory coverage. [5] This generally results in acceptable performance overheads, given that much of the data that a program is accessing is within its working set and thus requires no additional instructions to read and utilize the data.

SuperMem, in contrast, finds most of its cruxes embedded within its usage of a counter write-through cache. The main drawbacks include that each time a minor counter (which encrypts one line in a page) overflows, the major counter (which encrypts the entire page) must be incremented by one and the entire page must be re-encrypted with these new counters. [21] This can introduce latency in rare workloads that may be writing to one

area of memory frequently. However, SuperMem counteracts these possible latencies by reducing the amount of writes to NVM by counter write coalescing and speeding up writes with parallelizing cross-bank storage. [21] Counter write coalescing holds writes to the same memory line in NVMM to take place at once, reducing the amount of writes that need to take place. Using cross-bank storage removes the bottleneck of all the counters being written to the same bank or counters and their data being written to the same bank. [21] With these two methods, the overall latency is reduced and any subsequent rare case where re-encryption of pages is frequent will see some performance benefit over other encrypted NVMM approaches.

i-NVMM and SuperMem represent a legacy and recent entry into hardware-level encryption, respectively. i-NVMM reduces overall latency by leaving actively used pages decrypted, but these pages are left vulnerable upon shutdown, making it not as secure as SuperMem. SuperMem also results in a better shutdown for systems equipped with NVMM, as i-NVMM requires some time to encrypt the pages that were within the working set of the running programs before shutdown. Both approaches are transparent to the developer, making them easy to adopt on systems desiring the extra protection, but SuperMem makes a better case with its ability to more completely protect the NVMM.

3.2 Software-Level Mitigation

The other approach to making NVMM resistent to cold boot attacks comes in the paper written by Pan et al. [12] The authors propose a software-based solution that involves a patch to the linux kernel. This patch allows for memory blocks to be marked as sensitive, forcing them to reside in encrypted main memory. This prevents the data from residing in insecure NVM caches. It is important to note that this solution focuses on protecting the caches of NVMM rather than its main memory, as the authors acknowledge encryption as a satisfactory approach for main memory. [12] Securing caches pose a significant technical challenge, as encrypting them can incur significant overhead and is thus difficult to implement.

This solution is primarily implemented with a software API that will declare blocks in memory as secret and thus forbidden to be stored in NVM caches. The example usage given by the paper involves storing secret keys for disk encryption, but the same concepts apply to any secret data. [12] The software solution proposed has up to a 45% overhead without any cryptographic hardware acceleration on the ARM processors tested, but only 2% when there is hardware support. [12] The overhead is almost directly correlated with the amount of accesses to the uncacheable secret data (in the case of this paper, encryption keys). The authors propose that a performance optimization can be implemented by allowing secret data to be cached when an authenticated user is logged in, as cold boot attacks are unlikely to be staged while a user is utilizing the system.

This solution is geared towards protecting encryption keys, so it is uncertain how effective this solution would be for other use-cases. It may pose severe overheads for computations that rely on quick cache access to some subset of secret data, as the program would have to load this data from slower encrypted main memory each time the data is needed. This may also compromise some data upon system failure, as these secrets may be necessary to resume the last executed program upon restart.

3.3 Performance Improvements

Many of the papers that make improvements in passive vulnerability mitigation often contribute a significant amount of material and more completely solve the issue of data remanence attacks, making the contribution a new solution rather than an improvement on existing solutions. The paper written by Awad et al. [2] is one of the few that seems best classified as an improvement rather than a standalone solution to data remanence attacks. The paper introduces a secure NVMM controller, Silent Shredder, reduces the amount of writes that occurs in counter mode encryption by completely eliminating the writes that "zero" out a page before it is reallocated to another process. Instead of physically zeroing the page, Silent Shredder modifies the encryption key used for the particular page in encryption in order to make the data enclosed completely unintelligible to a new process utilizing it. [2] It mostly accomplishes this with initialization vectors, which handles both making the page unintelligible and providing the cache with a zero-filled block so that it properly recognizes its zeroed state. This solution not only reduces the amount of writes to NVMM, but also improves initialization read speed due to the minor counter making shredded blocks easy to identify. A zero filled block is returned to the cache, eliminating the need to actually read the invalid data. [2]

4 Active Vulnerability Mitigation

The other common class of papers address traditional memory vulnerabilities, such as those that arise from an attacker snooping on an executing program. These attacks often involve unauthorized memory reads and writes, which can lead to unauthorized memory disclosure or corruption. These threats are especially disastrous to NVMM due the longevity of data within them. The proposed solutions include (but are not limited to) using Merkle Trees to protect the integrity of the data being written [14], reducing the amount of time memory is attached to a program's address space [17], and providing intra-process isolation for persistent memory objects. [18]

Merkle trees (MTs) have been used in the past to verify the integrity of secure processors [10, 15], making it an attractive consideration for NVMM. MTs for integrity often involve the usage of message authentication codes (MACs), where MACs are associated with nodes of the MT. To verify integrity, the MACs are computed from the node written to the root MAC. The root of the MT, and thus its associated MAC, are always kept on chip. [15, 16] It is important to note that while MTs are effective at ensuring the integrity of data being written into memory, they also require several extra reads and writes to maintain. This places strain on the already reduced write capacity of NVMM. [14] The paper written by Rakshit et al. attempts to address this with **A**uthentication **S**cheme for **S**ec**URE** energy efficient NVMs (ASSURE). [14] This solution leverages multi-root MTs (MMTs) to reduce the amount of hashes that have to be recomputed in a MT on write, which in turn reduces writes to NVMM. To reduce the amount of memory needed to maintain the multiple roots processor-side, only a few roots are dynamically predicted to be active. In conjunction with the MMTs the authors also use smart MACs (SMACs), which leverage that DEUCE NVM encryption only re-encrypts modified words when consecutive write-backs to the same area

are made. [14] SMACs leverage that feature by only recomputing the parts of the MAC corresponding to the word(s) of the cache line that are re-encrypted on write.

The next approach displayed in literature involves reducing the amount of time that memory is exposed in a program's address space. [17] This solution uses a combination of address space layout randomization (ASLR) and page table entries (PTEs) embedded into the persistent memory objects (PMOs). When memory is needed within a program, the persistent memory object is attached to the program's address space by pointing a page table entry to the physical memory where the PMO is located. [17] A PTE tree is present in the case where the PMO is larger than a word in order to avoid the prohibitively large cost of initializing multiple PTEs. By creating several PTEs within the PMO, we avoid both the cost of creation of these PTEs and the cost of a TLB shootdown that would occur each time that a PMO is mapped to the memory of a program. [17] At each attach, the address of the PMO is also randomized so that an attacker cannot simply observe memory being reattached in the same location several times during execution. By changing where it is attached upon each attach and completely detaching the memory when not needed, attackers are unable to discern where the memory is located or stage attacks against it while unattached. [17]

The last approach is similar to Intel's MPK. The solution, detailed in the paper written by Xu et al., [18] involves the same process of splitting memory into several protection domains where access in and out is controlled by protection keys. When this methodology is applied to NVMM, a PMO is placed into one of several protection domains upon attach with an associated protection key detailing its access control policy. Subsequent loads and stores are checked against the domain's access policy as well as the the page access policy in the TLB or page table. [18] By protecting PMOs this way, badly behaving threads cannot compromise a PMO being utilized in another program that it does not have access to. MPK is not well suited to PMOs out of the box due to its limitation of 16 protection keys and domains, so a couple solutions are proposed to counteract this limitation. The first approach is similar to an existing solution, libmpk, which maps 16 protection keys to an unlimited amount of domains. The proposed approach differs from libmpk with its extra architecture to reduce overhead. This approach is called Hardware MPK Virtualization. [18] The second approach, called Hardware Domain Virtualization, completely abandons the need for limited keys by managing access control on the domain itself. This not only offers individual protection to each PMO, but also completely removes the need for TLB shootdowns when an entry is evicted or changed. [18]

Given how varied these approaches are, they are more difficult to compare directly. Many of these solutions could be used in tandem, especially ASSURE [14] and Hardware Domain Virtualization [18], as ASSURE's integrity verification schemes could be used as a backup in cases where data is wrongfully able to be modified in cases of human error in setting domain permissions. This is also the case for the solution that involves attaching and detaching PMOs[17], as it is primarily protecting the data from modification rather than assuring the integrity of the data. Most of the weaknesses of these papers originate in the fact they still incur worst-case overhead percentages roughly in the double digits [18, 17] or still have an impact on the longevity of the NVMM. [14] However, overhead and shortened memory life span penalties may be offset enough by the relative speed and recoverability of NVMM to make these solutions worthwhile over volatile memory and disk.

4.1 Performance Improvements

The papers that improve upon these solutions are mostly focused on providing better security metadata recovery. Given that a major promise of NVMM is quick recovery upon failure, assuring that the substantial amount of security metadata is also recovered without adding significant overhead is essential. Anubis [20] and Triad-NVM [3] both attempt to solve this issue. Anubis is a memory controller design that makes security metadata recovery faster and more consistent by keeping track of which addresses are not up to date upon a crash and restoring the parts of the tree that are affected. For SGX-style Merkle trees, Anubis also keeps a shadow copy of the cache in persistent memory. [20] This shadow copy is stored in a shadow table with a small Merkle tree to verify its integrity. Only the root of the tree needs to be kept securely, as verification of the shadow copy is only done during recovery. By only recomputing counter blocks and Merkle tree nodes that were marked as lost, Anubis eliminates the need to completely reconstruct Merkle trees.

Triad-NVM, in contrast, introduces a new methodology for persisting and recovering security metadata. In Triad-NVM, persistent and non-persistent regions are treated differently to speed up recovery. The primary mechanism for recovery of the persistent regions is persisting a given number of levels of the Merkle Tree from the leaf up. Persisting only the low levels of the tree can provide some assistance in isolating issues with corrupted counters. However, persisting more of the tree shortens recovery time, as there are less nodes to compute when reconstructing the tree. [3] In the non-persistent region of memory, the counters and Merkle tree are updated lazily by setting the intermediate parent nodes of the counters to zero. When a leaf node's counter is updated and a zero is found in it's parent, the architecture knows that this is the first write to the counter block and updates the counter value and its parent accordingly. [3] This solution does provide some recovery speedup over other solutions, but still results in multiple extra writes being made to NVMM, which is still detrimental to the hardware's longevity overall. [3]

5 Conclusion

Non-volatile main memory is still relatively new hardware, meaning that there are many aspects that require further development and instrumentation. This is especially so given that its primary users are cloud providers that require high reliability and strong security guarantees for any hardware adopted into their ecosystem. Security is especially important to these providers because their enterprise customers may be storing sensitive customer or company data. To illuminate the current research regarding the security of NVMM, this paper introduces and explains seven security papers and splits them into two categories based upon what class of vulnerabilities they defend against.

The first of these categories is passive vulnerabilities, which are generally solved with encryption. [17, 21, 12] The main areas for growth in these papers involve making the encryption require less writes to NVMM, as the extra writes would put strain on the cells. [2] A lot of improvements also center around reducing overheads and latency that the encryption causes. The most promising pathway for solutions in this category seems to be architecture-level encryption, as solutions can offer significant speed boosts through direct interaction

with the memory bus.

The other class of vulnerabilities addressed are active vulnerabilities, which have a myriad of solutions at the architecture level. [17, 18, 14] Architecture solutions provide some performance benefit and more flexibility for custom solutions that interface directly with memory. Few of these solutions cover all aspects of protecting memory; Some combination of protecting the memory from unauthorized writes and assuring data integrity is necessary to provide the most complete protection. Papers proposing improvements involve making security metadata more quickly recoverable and alternate methodologies for recovering the metadata. [20, 3] Merkle trees require many extra writes to fully persist, making them detrimental to the longevity of the memory when persisted but easily recoverable. Merkle trees also require a significant amount of extra computation to recompute upon recovery, making them detrimental to the recovery time of a machine upon failure when persisted minimally. Thus, much research has been devoted to striking the careful balance between both approaches.

Solutions for NVMM security are diverse and have several avenues for improvement. With time, NVMM is likely to become a significant candidate for main memory in machines with several diverse, long-running workflows, making their security paramount to the overall security of a significant subset of computing. By giving a general overview of the existing papers addressing NVMM security, this paper will hopefully help direct others to general research areas within NVMM security that require more attention and development.

References

[1] Tuomas Aura. "Strategies against Replay Attacks". In: *Proceedings 10th Computer Security Foundations Workshop*. 1997, pp. 59–68. DOI: 10.1109/CSFW.1997.596787.

[2] Amro Awad et al. "Silent Shredder: Zero-Cost Shredding for Secure Non-Volatile Main Memory Controllers". In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16. Atlanta, Georgia, USA: Association for Computing Machinery, 2016, pp. 263–276. ISBN: 9781450340915. DOI: 10.1145/2872362.2872377. URL: https://doi.org/10.1145/2872362.2872377.

[3] Amro Awad et al. "Triad-NVM: Persistency for Integrity-Protected and Encrypted Non-Volatile Memories". In: *Proceedings of the 46th International Symposium on Computer Architecture*. ISCA '19. Phoenix, Arizona: Association for Computing Machinery, 2019, pp. 104–115. ISBN: 9781450366694. DOI: 10.1145/3307650.3322250. URL: https://doi.org/10.1145/3307650.3322250.

[4] Wentao Cai et al. "Understanding and Optimizing Persistent Memory Allocation". In: *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*. ISMM 2020. London, UK: Association for Computing Machinery, 2020, pp. 60–73. ISBN: 9781450375665. DOI: 10.1145/3381898.3397212. URL: https://doi.org/10.1145/3381898.3397212.

[5] Siddhartha Chhabra and Yan Solihin. "i-NVMM: A Secure Non-Volatile Main Memory System with Incremental Encryption". In: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2011, pp. 177–188.

[6] Alexander Freij et al. "Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Memory". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020, pp. 14–27. DOI: 10.1109/MICRO50266.2020.00015.

[7] Swapnil Haria, Mark D. Hill, and Michael M. Swift. "MOD: Minimally Ordered Durable Datastructures for Persistent Memory". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 775–788. ISBN: 9781450371025. DOI: 10.1145/3373376.3378472. URL: https://doi.org/10.1145/3373376.3378472.

[8] Intel. *"Intel Optane Persistent Memory"*. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[9] Emre Kültürsay et al. "Evaluating STT-RAM as an energy-efficient main memory alternative". In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2013, pp. 256–267. DOI: 10.1109/ISPASS.2013.6557176.

[10] Tamara Silbergleit Lehman, Andrew D Hilton, and Benjamin C Lee. "PoisonIvy: Safe speculation for secure memory". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783741.

[11] Yuanjiang Ni et al. "SSP: Eliminating Redundant Writes in Failure-Atomic NVRAMs via Shadow Sub-Paging". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 836–848. ISBN: 9781450369381. DOI: 10.1145/3352460.3358326. URL: https://doi.org/10.1145/3352460.3358326.

[12] Xiang Pan et al. "Nvcool: When non-volatile caches meet cold boot attacks". In: *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE. 2018, pp. 439–448. DOI: 10.1109/ICCD.2018.00072.

[13] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. "Memory Persistency". In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. ISCA '14. Minneapolis, Minnesota, USA: IEEE Press, 2014, pp. 265–276. ISBN: 9781479943944.

[14] Joydeep Rakshit and Kartik Mohanram. "ASSURE: Authentication Scheme for SecURE Energy Efficient Non-Volatile Memories". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. DAC '17. Austin, TX, USA: Association for Computing Machinery, 2017. ISBN: 9781450349277. DOI: 10.1145/3061639.3062205. URL: https://doi.org/10.1145/3061639.3062205.

[15] Brian Rogers et al. "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE. 2007, pp. 183–196. DOI: 10.1109/MICRO.2007.16.

[16] G. Edward Suh et al. "Efficient Memory Integrity Verification and Encryption for Secure Processors". In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. USA: IEEE Computer Society, 2003, p. 339. ISBN: 076952043X.

[17] Yuanchao Xu, Yan Solihin, and Xipeng Shen. "MERR: Improving Security of Persistent Memory Objects via Efficient Memory Exposure Reduction and Randomization". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 987–1000. ISBN: 9781450371025. DOI: 10.1145/3373376.3378492. URL: https://doi.org/10.1145/3373376.3378492.

[18] Yuanchao Xu et al. "Hardware-Based Domain Virtualization for Intra-Process Isolation of Persistent Memory Objects". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 2020, pp. 680–692. DOI: 10.1109/ISCA45697.2020.00062.

[19] Fan Yang, Fei Li, and Jiwu Shu. "Survey on Secure Persistent Memory Storage". In: *Journal of Computer Research and Development* 57.5, 912 (2020), p. 912. DOI: 10.7544/issn1000-1239.2020.20190820. URL: https://crad.ict.ac.cn/EN/abstract/article_4176.shtml.

[20] Kazi Abu Zubair and Amro Awad. "Anubis: Ultra-Low Overhead and Recovery Time for Secure Non-Volatile Memories". In: *Proceedings of the 46th International Symposium on Computer Architecture*. ISCA '19. Phoenix, Arizona: Association for Computing Machinery, 2019, pp. 157–168. ISBN: 9781450366694. DOI: 10.1145/3307650.3322252. URL: https://doi.org/10.1145/3307650.3322252.

[21] Pengfei Zuo, Yu Hua, and Yuan Xie. "SuperMem: Enabling Application-Transparent Secure Persistent Memory with Low Overheads". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 479–492. ISBN: 9781450369381. DOI: 10.1145/3352460.3358290. URL: https://doi.org/10.1145/3352460.3358290.