Seeds of SEED: **R-SAW**: New Side Channels Exploiting Read Asymmetry in MLC Phase Change Memories

Md Hafizul Islam Chowdhuryy[†], Rickard Ewetz[†], Amro Awad[§], and Fan Yao[†]
[†]University of Central Florida [§]North Carolina State University
reyad@knights.ucf.edu, rickard.ewetz@ucf.edu, ajawad@ncsu.edu, fan.yao@ucf.edu

Abstract—**Phase Change Memory (PCM) is a promising contender for future main memory solutions. While many architecture-level performance optimizations have been studied for PCM, *the security implications* of these designs are not well understood. This work demonstrates *the first investigation* of information leakage threats in PCM-based main memories. Notably, we find state-of-the-art read techniques leveraging *access latency asymmetry* in Multi-level Cell (MLC) PCM introduce new timing variations. To understand the severity of the vulnerability, we present *R-SAW*, a novel side channel attack that aims to exfiltrate secrets from a victim process via passively observing execution timings that are correlated with secret-dependent PCM accesses. We demonstrate the attack on a real-world cryptographic algorithm–AES encryption in OpenSSL. Our evaluation shows that R-SAW is able to completely recover the encryption keys. Furthermore, our experiments reveal that R-SAW exhibits superior noise resilience compared to the widely-studied cache-based side channels. Our work highlights the importance of understanding security in systems integrated with emerging memory technologies and motivates the need to architect secure-by-design PCM main memories in the future.**

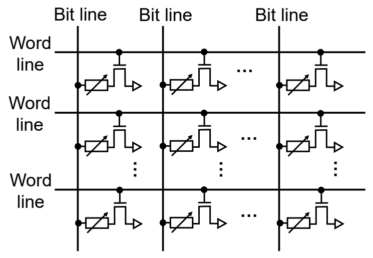
I. Introduction

The rapid advances in high performance and data-intensive computing have significantly pushed the demand for efficient and scalable memory systems. Non-volatile memories (NVMs) that offer high density, superior power efficiency, and persistent storage are increasingly regarded as the major building blocks for future memory systems. PCM is highly promising due to its maturity and DRAM-comparable performance [1]–[3]. Since PCM cells have a wide resistance range, they can be operated in MLC mode, which offers higher densities by encoding multiple bits per cell. While demonstrating several appealing advantages, PCM read latency in MLC mode can be significantly higher than single-level cell (SLC) mode [4]. In particular, the bitwise iterative sensing for MLC read operations leads to asymmetry of read latency among the bits. Since read operations are in the critical path, techniques optimizing read performance for PCM cells are widely studied [5]–[8].

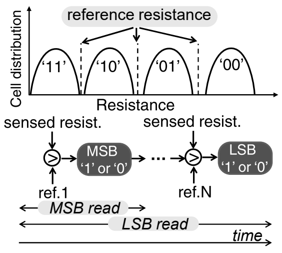
Although performance optimization has been the driving force for motivating new architecture designs, the burgeoning of hardware and microarchitecture attacks [9]–[15] in recent years show that performance-enhancing techniques without careful considerations of security can potentially open new venues for security breaches. Therefore, understanding security properties of emerging hardware in the early design stage and ensuring secure-by-design architecture is imperative. In this work, we aim to answer the following question: *Are emerging PCM-based main memory systems vulnerable to information leakage?* We focus on investigating the potential security vulnerabilities of performance-optimized access techniques for MLC PCM main memories. Our key observation is that existing widely-recognized PCM read techniques typically leverage data striping mechanisms that decouple the access to MLC bits with varying speed grades [5]–[7]. While these techniques undoubtedly bring performance benefits, they can potentially allow adversaries to exfiltrate secretive data by exploiting PCM read latency timings.

This paper demonstrates the first study on side channel threats in future systems equipped with PCM as main memory. We perform a systematic characterization of PCM access timings on the state-of-the-art inter-line striping scheme for read performance optimization [5]. Our investigation reveals that the read asymmetry allows highly visible time-varying program executions (even under the same execution path) due to distinctions in accessing fast and slow regions in MLC PCM. Such a characteristic allows the adversary to compromise a victim process's secrets if they could be inferred from PCM main memory accesses. We implement R-SAW, a novel side channel attack that exfiltrates AES keys in OpenSSL by correlating *PCM memory access patterns* with *program execution times*. Our evaluation shows that R-SAW can recover entire keys with 98.5% accuracy at runtime. While prior works have shown recovery of crypto keys through passive side channels via caches (i.e., [10], [16]), our results reveal that R-SAW exhibits considerably higher noise resilience where it manages to recover 78% of the key bytes under an extremely noisy environment where the prior cache-based side channel fails. Our work on PCM-based side channel highlights the new source of leakage in emerging memory technologies. Findings in this paper can provide computer architects with new insights of information security for future memory systems. In summary, the major contributions of this paper are:

- We make *the first investigation* of side channel vulnerabilities in MLC PCM originating from the read asymmetry due to architectural-level performance optimizations.
- We present R-SAW, a novel side channel attack that can completely exfiltrate AES keys by exploiting timing variances due to read asymmetry in MLC PCM.
- We perform a quantitative analysis of the impact of system noises on R-SAW. Our results show that R-SAW exhibits superior noise-resilience.
- We discuss several potential defensive techniques to mitigate PCM-based side channels. Our work highlights the importance of designing side channel resistant MLC PCM for future memory systems.



(a) PCM cell array



(b) PCM read operation

Fig. 1: Illustration of PCM array and MLC sensing technique.

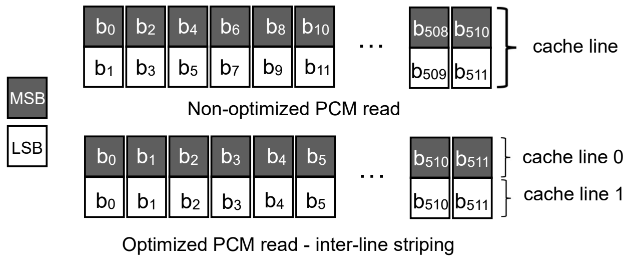


Fig. 2: PCM-optimized bit organization.

II. Background

Basics of Multi-level Cell PCM. PCM takes advantage of the phase change property of Chalcogenide Alloy (e.g., GST) that switches between a high resistance *amorphous* state and a low resistance *crystalline* state. PCM cells are typically organized as memory arrays that are accessible through word lines and bit lines (Figure 1a). As the resistance level in the amorphous state is several orders of magnitude higher than the crystalline state, the resistance range can be safely divided into several non-overlapping bands where multiple bits can be encoded in a single PCM memory cell (i.e., MLC mode) [4], [17]. Figure 1b shows the mapping of resistive states to bit symbols in 2-bit MLC cells. While MLC mode offers higher capacity, it inevitably complicates the read sensing operation. Specifically, reading bits in a PCM cell involves iterative sensing. In each iteration, the logic compares the resistance with a reference value to derive one bit at a time (i.e., from MSB to LSB) as shown in Figure 1b. We assume a 2-bit MLC for main discussion in this paper. Note that the same principle also applies to higher-density MLC PCM.

Performance-oriented PCM Read Schemes. Due to the iterative sensing procedure, reading the LSBs is about 2× slower than the MSBs. In conventional data mapping scheme where consecutive bits in a memory line are mapped to consecutive bits in MLC cells, the read latency is determined by *LSB sensing*, which could considerably degrade the overall performance compared to systems with SLC-based PCM. To retain a reasonable read performance, it is necessary to expose the read asymmetry to architecture level and *decouple the fast MSB accesses from the slow LSB accesses*. In fact, state-of-the-art read techniques generally adopt *bit striping schemes* that map certain memory data regions (e.g., a memory line) exclusively to MSBs and others to LSBs in multi-level cells, effectively accelerating reads for MSB-mapped regions without deteriorating read latency for LSB-mapped ones. Prior studies have prototyped such performance-oriented read techniques at various striping granularities including intra-line [7], inter-line [5] and inter-page striping [6]. Figure 2 illustrates the representative design where consecutive data lines are mapped to MSBs and LSBs alternatively such that reading of *odd lines* benefits from much shorter latency than *even lines* [5].

Microarchitecture Timing Channel Attacks. Microarchitecture timing channel attack is a form of information leakage attacks where a spy exfiltrates secretive information from a victim through observing or modulating the hardware access timings [9], [13], [18], [19]. These attacks are particularly worrisome as they can evade existing system-level defenses and do not leave any physical traces after exploitation. The root cause of microarchitectural timing-based leakage is *the variations of access latencies* in underlying hardware that create *measurable slow or fast executions* either directly by the spy [13] or indirectly from the victim [16]. Note that existing works have unveiled timing channel attacks targeting many performance-oriented microarchitecture designs built in processors over the years (e.g., caching and speculation).

III. Threat Model

We focus on side channel attacks on systems equipped with MLC PCM as main memories. The victim process runs services that operate on certain secrets on the targeted machine. We assume the spy either sits remotely or runs as a non-privileged process co-located with the victim. Similar to prior *passive side channels* on caches [16], [20], [21], we assume the spy passively monitors externally observable information non-intrusively and attempts to decipher the victim's secrets through correlating the secret values with timing observations. The attacker can also perform certain profiling on a local machine with a similar setup to the target machine.

IV. MLC PCM Read: A New Source of Leakage?

In this section, we will investigate whether these read-enhancement schemes could expose new timing source that can potentially lead to information leakage.

Hardware	**Configurations**
Processor	Quad-core x86 CPU, Out-of-order execution
L1 I/D-Cache	Private, 32KB, 2-way, 1-cycle hit
L2 Cache	Private, 4MB, 16-way, 10-cycle hit
DRAM Cache	Shared, 32MB, 16-way, 50-cycle hit
Mem. Controller	64 RD & WT queue, FR-FCFS scheduling, open-row policy
PCM Memory	8GB, single channel, 2 ranks/channel (Local) 16GB, dual channel, 2 ranks/channel (Target)
PCM Timing	2-bit MLC, *MSB read:* 28ns, *LSB read:* 48ns [1], [5]

TABLE I: Architecture configurations.

A. MLC PCM System Modeling

We build a comprehensive architecture model for PCM-based systems [1] that integrates the state-of-the-art MLC PCM read mechanism using *inter-line striping* among bits in MLC cell [5], [22]. This is a highly efficient technique as it can effectively harness the read asymmetry for performance gain

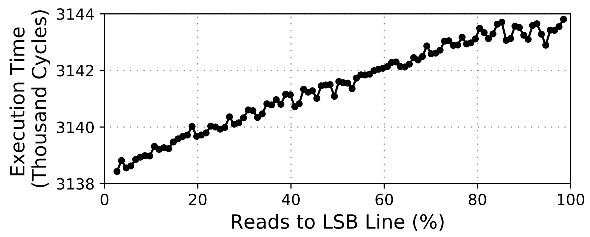


Fig. 3: Program execution time as the percentage of LSB line reads changes.

with low hardware cost. Under such a scheme, the memory controller maps consecutive memory blocks to MSBs and LSBs in an alternating manner (as shown in Figure 2). As such, odd lines are allocated with MSBs (MSB lines), and even lines are located in LSBs (or vice versa). The memory controller's read service time to odd lines only considers the timing for one iteration sensing while even lines take two iterations. We use gem5, a cycle-level simulator, to model an x86 based system with MLC PCM. Memory parameters are derived from existing studies [5]. The detailed architecture configuration is listed in Table I.

B. Impact of PCM Access Patterns on Execution Time

Since MSB and LSB lines have different read latencies, for a program that issues a certain number of memory read requests, its execution time may have a strong dependency on the MLC PCM access pattern–the breakdown of MSB and LSB lines loaded. Specifically, the program execution time shall increase as more memory reads fall in the LSB lines. To characterize the impact of PCM access patterns on program execution, we develop and run a microbenchmark that reads from arbitrary memory locations for a pre-determined number of times. We vary the percentage of LSB (or MSB) lines accessed and measure the corresponding execution times. Figure 3 shows the average execution latency for different PCM access patterns. As we can see, while there are occasional local small spikes on the execution time samples, the program execution time exhibits a clear *linear relationship* with the reads to LSB lines. Based on this result, we make the key observation that *differentiation in PCM access patterns can induce externally observable slow and fast executions*. We note that such timing variations do not exist on non-optimized MLC PCM main memories where memory array access latency is fixed.

C. Side Channel Attacks on MLC PCM

We note that there is potential information leakage if a victim process's PCM memory access patterns depend on secretive values. We present a high-level attack framework targeting MLC in Figure 4. Specifically, a spy process can interact with a benign victim process by sending requests through software interfaces ①. The victim services the requests based on certain secrets. Depending on the value of the secrets, the victim may exhibit different PCM access patterns, such as x_1 (MSB) and x_2 (LSB) in ② vs. x_3 (LSB) and x_4 (LSB) in ③. After the victim's operation is completed, it sends back the response to the spy ④. The spy not only

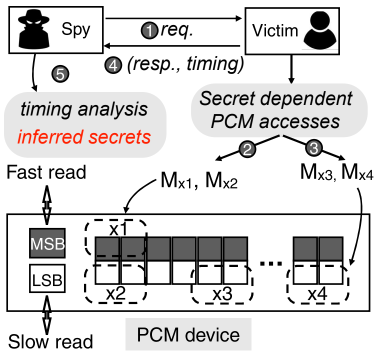


Fig. 4: Side Channel Attack Framework on MLC PCM.

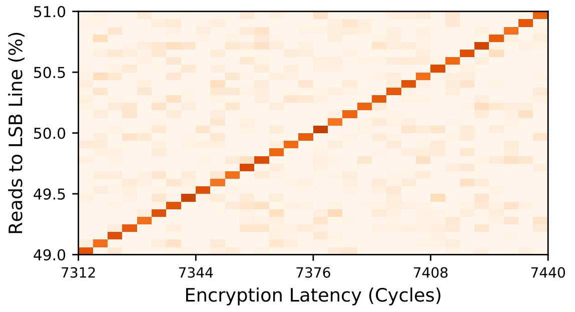


Fig. 5: Distribution of AES encryption latency over PCM access pattern.

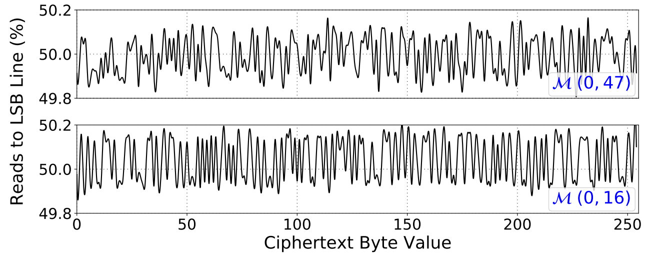
receives the victim's response but also measures the latency of the victim's execution. The spy then performs timing analysis with the attempt to infer secrets of the victim process ⑤.

V. AES Side Channels through MLC PCM

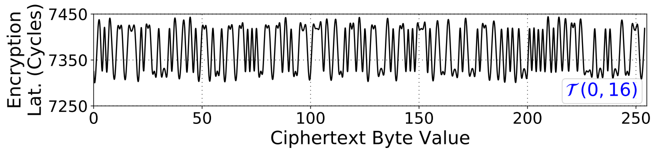
In this section, we demonstrate a novel information leakage attack–*R-SAW*, that exfiltrates crypto keys via exploiting PCM-based timing channels in AES encryption.

The AES software implementation typically takes advantage of pre-computed values to replace computation with table lookups. We target the AES-128 implementation in OpenSSL [23] that uses 5 T-tables for encrypting 16-byte data block. AES-128 uses one 128-bit encryption key and performs 10 rounds of transformation. The first 4 tables (e.g., T_0, T_1, T_2, and T_3) are used in the first 9 rounds where the last round only accesses the 5^{th} table T_4. Each round involves 16 memory accesses (i.e., table lookups) whose indices partially depend on the prior round keys. We make the same assumption as prior attacks [10], [21] that the memory blocks containing T-tables are not cached on chip prior to each encryption run.

Since the total number of table lookup in AES encryption is fixed, we expect that the overall encryption latency also exhibits a strong relation with the percentage of LSB lines. To validate this assumption, we run one million random AES encryptions and collect the execution times, as well as the number of LSB/MSB lines read from the T-tables. Figure 5 shows the distributions of encryption latencies for every LSB line percentage. We can see that despite a narrow range of LSB line percentage values (0.49 to 0.51), its linearity relation with the execution time is still observed. As *AES table access addresses are dependent on the round keys*, we hypothesize



(a) Memory-pattern vector for $k_0^{10} = 47$ (top) and $k_0^{10} = 16$ (bottom)



(b) Encryption-timing vector with $k_0^{10} = 16$

Fig. 6: Attack vector for (a) $\mathcal{M}(0,47)$ and $\mathcal{M}(0,16)$ respectively and (b) $\mathcal{T}(0,16)$.

that there could be a *deterministic PCM access pattern for a particular value of a key byte* when encrypting plaintexts in AES. In this case, then it is possible for an attacker to exfiltrate individual key values by performing a correlation analysis between a guessed key's PCM access patterns and the victim's execution times. R-SAW is designed to target the last round key, which, if compromised, could lead to exfiltration of all keys. R-SAW incorporates the following attack steps:

PCM Access Pattern Profiling on AES. In the profiling phase, the attacker aims to build *memory-pattern vectors* (MPVs) that are used later to infer individual key bytes in the victim. To do so, the attacker first instruments the AES program to identify MSB/LSB line accesses (i.e., odd/even blocks) so that the PCM access traces could be generated. The attacker then performs a sufficient number of AES encryptions on a local machine. Each encryption uses a *randomly generated* plaintext and a user key (that determines all round keys). For each run, a sample point $S = (C, K^{10}, p)$ is recorded, where C is the 16-byte ciphertext, K^{10} is the last round key (random) and p is the percentage of LSB lines read in this execution. We then organize the sample points based on every unique value combination of k_i^{10} and C_i for each i. Specifically, for every possible value of the i^{th} key byte, we get all the sample points whose $K_i^{10} = u$ and $C_i = w$ (u and $w \in [0, 255]$) as a group $\mathcal{S}(i, u, w)$. $\mathcal{S}(i, u, w)$ incorporates information about the PCM access pattern under specific values of the i^{th} key byte and the i^{th} ciphertext byte. We compute the *average* percentage of LSB lines among all the sample points in $\mathcal{S}(i, u, w)$ as $\overline{P}_{(i,u)}^{w}$ and define the memory-pattern vector for the i^{th} byte as the following:

$$\mathcal{M}(i, u) = \{\overline{P}_{(i,u)}^{0}, \overline{P}_{(i,u)}^{1}, ..., \overline{P}_{(i,u)}^{255}\} \quad (1)$$

Essentially, for the i^{th} key byte, the memory-pattern vector contains 256 elements, each representing the average LSB line read percentage among encryption samples that have a unique C_i (i.e., the w value). As a result, we will generate *256 $MPVs$* for each key byte. In total, 4096 vectors are computed for all

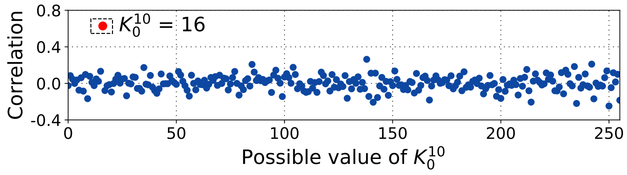


Fig. 7: Correlation between memory-pattern vectors and the victim's encryption-timing vector for K_0^{10}.

16 key bytes in the last round. These vectors are regarded as the *PCM access signature* for each key instance. Note that the profiling process only has to be done once as an offline step.

Victim's Execution Time Monitoring. After the profiling step, the attacker will start the online monitoring step where it triggers AES encryptions from the victim process with random plaintexts and measures the encryption timings. A sample point in this stage can be denoted as $S = (C, l)$ where C is the ciphertext and l is the execution latency. The attacker organizes the sample points similar to the profiling phase. Particularly, for each of the i^{th} byte in ciphertext, sample points that share the common C_i (again denoted as w) are grouped together as $\mathcal{S}(i, \mathbf{x}, w)$ where $\mathbf{x} = K_i^{10}$ is *fixed but unknown yet*. Subsequently, the attacker calculates the *average latency* among all sample points for each $\mathcal{S}(i, \mathbf{x}, w)$ as $\overline{L}_{(i,\mathbf{x})}^{w}$. The *encryption-timing vector* (ETV) is generated for each byte of the victim's last round key as follows:

$$\mathcal{T}(i, \mathbf{x}) = \{\overline{L}_{(i,\mathbf{x})}^{0}, \overline{L}_{(i,\mathbf{x})}^{1}, ..., \overline{L}_{(i,\mathbf{x})}^{255}\} \quad (2)$$

Note that $\mathcal{T}(i, \mathbf{x})$ captures the statistical timing patterns corresponding to an unknown value of the i^{th} key byte.

AES Key Recovery through Correlation Analysis. Now that both the memory-pattern vectors ($\mathcal{M}(i, u)$) and encryption-timing vectors ($\mathcal{T}(i, \mathbf{x})$) have been collected, the attacker can attempt to infer the secret key values based on correlation analysis. The key motivation is that $\mathcal{M}(i, u)$ carries information on the PCM access pattern for key byte i with value u. Since PCM access patterns (denoted using the percentage of LSB lines reads) have a strong correlation with the encryption timings (shown in Figure 5), if $\mathcal{M}(i, u)$ is sufficiently distinct for each value of u, we expect that there would be *an outstandingly higher correlation between $\mathcal{M}(i, u)$ and $\mathcal{T}(i, \mathbf{x})$, given $\mathbf{x} = u$.* As a result, the i^{th} key byte can be deciphered using the following procedure:

$$K_i^{10} = \arg\max_{u} R(\mathcal{M}(i, u), \mathcal{T}(i, \mathbf{x})) \quad (3)$$

R calculates the correlation between $\mathcal{M}$ and $\mathcal{T}$. In other words, the value of i^{th} key byte is equal to u that leads to the maximum correlation between the corresponding memory-pattern vector and encryption-timing vector. Once all the final round key bytes are obtained, the attacker can easily recover the original user key [10]. It is worth noting that MPVs are inherently dependent only on the program-level behavior and implementation. Therefore, *MPV traces will be the same regardless of the configuration of local machines*. As a result, R-SAW *does not* require the same hardware setup between the local machine and the remote machine hosting the victim.

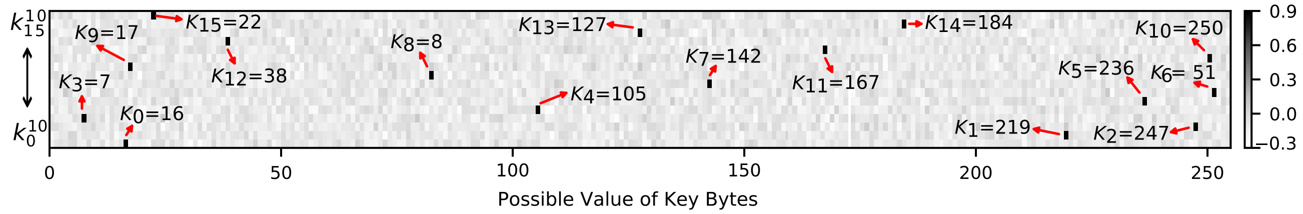


Fig. 8: A complete recovery of the final round key. Each row denotes the correlation value distribution for one key byte.

VI. Evaluation

A. Key Recovery Results with R-SAW

We evaluate the key recovery effectiveness on AES using the methods described in Section V. In the profiling phase, the number of encryptions has to be large enough so that sufficient sample points are collected for every possible key byte and ciphertext byte value combination. We empirically find that running 30 Million AES encryptions with random plaintext and user key is adequate. On our simulated platform, this step takes less than 2 hours in total. Note that *profiling is only performed once offline*. In the runtime monitoring phase, the attacker initiates 128K AES encryptions in the victim to generate the encryption-timing vector. We observe that the encryptions needed on the victim's side are significantly less than the profiling phase as the victim uses only one fixed key.

Our profiling results show that for each key byte, the memory-pattern vectors ($\mathcal{M}(i, u)$) corresponding to its different values are clearly distinguishable. Figure 6a presents the memory-pattern vectors for K_0^{10} under two key byte values: 47 and 16 (i.e., $\mathcal{M}(0, 47)$ and $\mathcal{M}(0, 16)$). Figure 6b illustrates the encryption-timing vector collected from the victim AES program with K_0^{10} set to 16 (i.e., $\mathcal{T}(0, 16)$). It can be seen that the $\mathcal{M}(0, 16)$ vector indeed closely resembles T(0,16), indicating a high correlation between these two vectors that share the same key byte value. In the final key recovery step, the attacker computes the Pearson correlation between the encryption-timing vector and the memory-pattern vector under a guessed key value. Figure 7 shows the correlation coefficient between the $MPVs$ and the target ETV for the first key byte (i.e., K_0^{10}). We can see that among the 256 $MPVs$, a point clearly stands out with the highest correlation at key value 16 (i.e., when $\mathcal{M}(0, 16)$ is used). Therefore, the attacker can correctly guess that K_0^{10} has value 16. Figure 8 presents the correlation analysis for each of the victim's 16 key bytes in a grey-scale heat map. We observe that for each key byte, an obvious dark point exists for each row (i.e., outstanding correlation), indicating the correct key value. R-SAW is able to completely reveal all the key bytes values for this key instance. Finally, we have run 4000 iterations of R-SAW attack where the victim's AES program uses a different key for each iteration. Our result shows that R-SAW can recover all keys with 98.5% accuracy.

B. Characterizations of R-SAW Attack

In this section, we aim to characterize the effectiveness of R-SAW by performing a comparative study with the state-of-the-art passive timing channels on AES [10], [16]. Note that

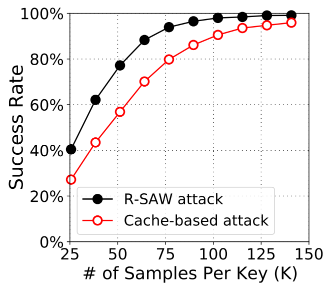


Fig. 9: Success rates with the change of sample size.

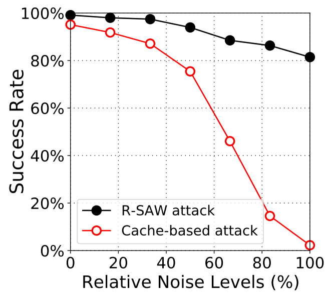


Fig. 10: Success rates with the change of system noise.

we do not perform evaluations on other active side channels exploits such as Prime+Probe [24] and Flush+Reload [18] as these attacks require explicit perturbations to the victim's cache accesses, which is different from R-SAW. Particularly, the work from Liu et al. [16] has revealed that in the last round encryption, if two table lookups (i^{th} and j^{th}) to T_4 map to the same entry, it will hold true that $C_i \oplus C_j = K_i^{10} \oplus K_j^{10}$. The cache attack works by first assuming a particular XOR value between two key bytes: $K_i^{10} \oplus K_j^{10}$, and then compute the average latency among sample points for which $C_i \oplus C_j$ equals to the assumed value. The correct $K_i^{10} \oplus K_j^{10}$ is equal to the assumed value whose corresponding average latency is the shortest. We implement the cache-based attack in [16] and compare its attack effectiveness with R-SAW. To perform a quantitative analysis, we define the *Success Rate* (SR) metric. For N iterations of attack (each with a different AES key), SR for R-SAW is the average percentage of key bytes correctly recovered; SR for cache-based attack is the average percentage of XOR values accurately recovered.

Sensitivity to Sample Size. We first analyze SR by varying the number of sample points. To do that, we generate 100 random keys and perform both attacks with these keys. We change the number of samples per key from $25K$ to $140K$. Figure 9 illustrates the trend of SR as sample size changes. We can see that for both attacks, SR increases as the sample sizes increase. However, the gain in SR diminishes as the sample size reaches 100K. Furthermore, for given sample size, R-SAW consistently achieves a higher success rate compared to cache-based attacks (up to 20%). We expect that the higher attack efficiency in terms of sample size is because the timing variations due to PCM access pattern is more prominent than the cache hit patterns.

Resiliency to System Noise. In realistic settings, there might be noises from background processes. Noises can obfuscate timing measurements, which potentially undermines the effectiveness of timing channels. To understand the impact of noise, we run a multi-threaded noise injection process on the target AES system that performs randomized memory loads. The highest level of noise corresponds to accessing caches/memories in a non-stop manner. We then scale the noise levels by slowing down the memory access frequency accordingly in each thread. Under each noise level, we generate 100 AES keys, and 128K sample points are collected for each attack. Figure 10 shows the success rates of R-SAW and cache-based attacks with different noise levels. We can see that the cache-based attack is extremely sensitive to system noise. In particular, a sharp decrease in SR is observed when the noise level is above 40%. In contrast, R-SAW exhibits superior resilience: It can still successfully recover 81% of the key bytes at the highest noise level where the effectiveness of the cache attack is degraded to random guess. This is because the cache-based attack relies on cache hits for its statistical measurement. With the spike in system noises, memory lines brought to the cache by one access (e.g., i^{th} table lookup) may be quickly evicted by the background activities. As a result, the subsequent j^{th} table lookup to the same block may experience cache miss instead of hit, leading to obfuscated timing observation. Differently, R-SAW's exploitation depends on PCM memory access timing. Even with more main memory accesses as a result of memory access contention, the memory-access pattern (i.e., percentage of LSB line) can still be quite stable as it is dependent on the key value. As a result, R-SAW represents a new timing source independent of prior exploits. R-SAW is highly resilient, and thus it may pose an even higher risk in future systems.

VII. Discussion

A. Applicability of R-SAW on Different MLC PCM Systems

While our main study focuses on inter-line data striping scheme in MLC PCM, we note that the new side channel threat can be potentially manifested in other implementations. Specifically, with intra-line striping [7], read asymmetry is exposed among sub-blocks of a memory line. This may enable finer-grained timing observations, making the potential exploits even more devastating. On the other hand, prior work has shown certain applications involve secret-dependent page-level memory activities [12], these applications can be vulnerable to MLC PCM employed with inter-page striping.

B. Mitigations

We discuss two mitigation methodologies for R-SAW and the corresponding challenges with these mechanisms.

Randomized PCM Data Mapping. One plausible mitigation technique is to integrate architectural support in memory controller that randomizes memory line mapping. Particularly, instead of mapping odd/even lines to MSBs and LSBs deterministically, two logically consecutive memory lines can be remapped randomly to either MSB or LSB lines on the same page using permutation seed generated at runtime. This way, the memory-access pattern is randomized and its correlation with the execution timing would be obfuscated. However, such approach could incur extra storage overhead for permutation metadata maintenance. Also, the permutation seed might need to be changed frequently to secure long-lived pages [25].

Software Hardening. Existing techniques have studied software rewriting (e.g., removing branches in security sensitive path) to ensure information security against microarchitecture attacks [26], [27]. We note that similar software hardening techniques could be utilized to mitigate R-SAW. Specifically, security-sensitive software can map memory addresses tainted with secretive information to the regions with the same speed grade to avoid secret-dependent PCM access timings. However, implementing PCM security-aware data mapping in software can require non-trivial modifications, which could pose considerable burden on program developers. Another challenge with this approach is that such secure implementation is only specific to certain MLC PCM systems, which may not be generic enough to offer protection on systems employed with different PCM access techniques.

VIII. Related work

Recent studies have shown successful exfiltration of cryptographic keys (e.g., AES and RSA) through many microarchitectural components such as caches [10], [13], [16], and branch predictors [14], [28]. Existing protection mechanisms against cache timing channel include randomization (e.g., randomized cache indexing) [25] and resource partitioning [29]–[31]. We note that our work demonstrates a new side channel threat and those existing defense schemes do not mitigate R-SAW. Another generic approach for alleviating side channel is to obfuscate the precision of timing observation through noise injections [32]. However, our study in Section VI has shown that R-SAW exhibits much higher resilience to system-level noises. Therefore, while this approach may be effective for caches, the desired noise level to obscure R-SAW could be too high to be tolerated in systems for regular operations. Besides architectural-level defenses, existing works have proposed secure crypto algorithms using constant-time implementation [33]. We note that R-SAW can also potentially be applied to non-crypto applications in a much broader spectrum.

IX. Conclusion

In this paper, we investigate the information leakage vulnerabilities in PCM-integrated systems. Our work reveals that state-of-the-art PCM read techniques that leverage read asymmetry in Multi-level cell open new side channel attack vectors. We present R-SAW, a novel side channel that can fully recover AES encryption keys based on MLC PCM access-dependent timings. The evaluation result shows that R-SAW exhibit much higher attack efficiency, as well as superior noise resilience, compared to cache-based attacks. Our work motivates the need to architect secure-by-design MLC PCM main memories for future systems.

Acknowledgments

This work is supported in part by U.S. National Science Foundation under CNS-2008339 and CNS-1908471.

References

[1] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *IEEE/ACM ISCA*, 2009, pp. 2–13.

[2] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *IEEE/ACM ISCA*, 2010, pp. 153–162.

[3] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung *et al.*, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465–479, 2008.

[4] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. Chow, A. Cabrini, G. M. A. Calvi *et al.*, "A multi-level-cell bipolar-selected phase-change memory," in *IEEE ISSCC*, 2008, pp. 428–625.

[5] M. Hoseinzadeh, M. Arjomand, and H. Sarbazi-Azad, "Reducing access latency of mlc pcms through line striping," in *IEEE/ACM ISCA*, 2014, pp. 277–288.

[6] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories," *ACM TACO*, vol. 11, no. 4, pp. 1–25, 2014.

[7] M. Arjomand, A. Jadidi, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "HL-PCM: MLC PCM main memory with accelerated read," *IEEE TPDS*, vol. 28, no. 11, pp. 3188–3200, 2017.

[8] M. Jalili and H. Sarbazi-Azad, "Express read in MLC phase change memories," *ACM TODAES*, vol. 23, no. 3, pp. 1–24, 2018.

[9] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *IEEE S&P*, 2019, pp. 1–19.

[10] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *Springer CHES*, 2006, pp. 201–215.

[11] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture," in *IEEE ACSAC*, 2006, pp. 473–482.

[12] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *IEEE S&P*, 2015, pp. 640–656.

[13] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Springer CT-RSA*, 2006, pp. 1–20.

[14] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," in *ACM ASPLOS*, 2018, p. 693–707.

[15] V. R. Kommareddy, B. Zhang, F. Yao, R. Ewetz, and A. Awad, "Are crossbar memories secure? new security vulnerabilities in crossbar memories," in *IEEE CAL*, 2019, pp. 174–177.

[16] F. Liu and R. B. Lee, "Random fill cache architecture," in *IEEE MICRO*, 2014, pp. 203–215.

[17] M. Jalili, M. Arjomand, and H. S. Azad, "A reliable 3D MLC PCM architecture with resistance drift predictor," in *IEEE DSN*, 2014, pp. 204–215.

[18] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: a high resolution, low noise, l3 cache side-channel attack," in *USENIX Security*, 2014, pp. 719–732.

[19] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *IEEE HPCA*, 2018, pp. 168–179.

[20] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[21] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *IEEE HPCA*, 2016, pp. 394–405.

[22] M. Asadinia and H. Sarbazi-Azad, "Inter-line level schemes for handling hard errors in pcms," in *Elsevier Advances in Computers*, 2020, vol. 118, pp. 49–78.

[23] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS (v. 0.9.7)," 2006, www.openssl.org.

[24] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE S&P*, 2015, pp. 605–622.

[25] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *IEEE MICRO*, 2018, pp. 775–787.

[26] D. Molnar, M. Piotrowski, D. Schultz, and D. Wagner, "The program counter security model: Automatic detection and removal of control-flow side channel attacks," in *Springer IACR*, 2005, pp. 156–168.

[27] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *IEEE S&P*, 2009, pp. 45–60.

[28] M. H. I. Chowdhuryy, H. Liu, and F. Yao, "BranchSpec: Information Leakage Attacks Exploiting Speculative Branch Instruction Executions," in *IEEE ICCD*, 2020.

[29] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *IEEE MICRO*, 2018, pp. 974–987.

[30] F. Yao, H. Fang, M. Doroslovacki, and G. Venkataramani, "Cotsknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies," in *IEEE HOST*, 2019, pp. 121–130.

[31] F. Yao, H. Fang, M. Doroslovački, and G. Venkataramani, "Leveraging cache management hardware for practical defense against cache timing channel attacks," *IEEE Micro*, vol. 39, no. 4, pp. 8–16, 2019.

[32] H. Fang, S. S. Dayapule, F. Yao, M. Doroslovački, and G. Venkataramani, "Prefetch-guard: Leveraging hardware prefetches to defend against cache timing channels," in *IEEE HOST*, 2018, pp. 187–190.

[33] G. Barthe, G. Betarte, J. Campo, C. Luna, and D. Pichardie, "System-level non-interference for constant-time cryptography," in *ACM CCS*, 2014, pp. 1267–1279.