Understanding and Characterizing Side Channels Exploiting Phase-Change Memories

Md Hafizul Islam Chowdhuryy and Rickard Ewetz, *Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 32816, USA*

Amro Awad, *North Carolina State University at Raleigh, Morrisville, NC, 27560, USA*

Fan Yao, *Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 32816, USA*

Recent advances in nonvolatile memory (NVM), together with their performance-optimized architectural schemes, position NVMs as promising building blocks for future main memory. However, the security of such techniques has not been explored. This article performs the first study on information leakage threats in phase-change memories (PCM). We propose an attack framework, read-saw (R-SAW), that systematically investigates side channel vulnerabilities in representative read techniques under interline and intraline interleaving for multilevel cells. Our evaluation shows that the new side channels can accurately leak program secrets (e.g., crypto keys) and are extremely robust to noise. Our work highlights the need to understand microarchitecture security for emerging memory devices.

Recent developments in microarchitecture attacks have raised significant concerns for information security. Particularly, a burgeoning of side channels has been demonstrated in a plethora of processor hardware components.[1,2] These exploitations highlight the fact that hardware performance optimizations without proper consideration of security often open new venues for information leakage. As new hardware components and microarchitecture optimizations are more rapidly integrated into modern computing systems, understanding their security impacts is critical to ensure secure-by-design solutions.

Emerging memory technologies have become major contenders for main memory with their advantage in nonvolatility, outstanding capacity, and superior energy efficiency.[3] Phase-change memory (PCM) is a promising class of nonvolatile memories (NVMs) due to its maturity and dynamic random-access memory (DRAM)-comparable performance.[4] To enable the efficient integration of PCM in computing systems, many architectural schemes for optimizing PCM main memory have been proposed in recent years.[4,5,6] While tremendous efforts have been put into studying the microarchitecture security of on-chip resources, information leakage vulnerability in architectural schemes for PCM has not been well understood.

MAINSTREAM ARCHITECTURAL SCHEMES FOR PCM READ COMMONLY LEVERAGE THE READ ASYMMETRY IN MLC CELLS FOR PERFORMANCE OPTIMIZATION.

This article demonstrates the first work on investigating side channels in future systems equipped with PCM. We systematically surveyed the state-of-the-art *read techniques* for PCM operating under the multilevel cell (MLC) mode, a widely utilized configuration that increases memory capacity. We identify that mainstream

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3238894
Date of publication 23 January 2023; date of current version 28 August 2023.

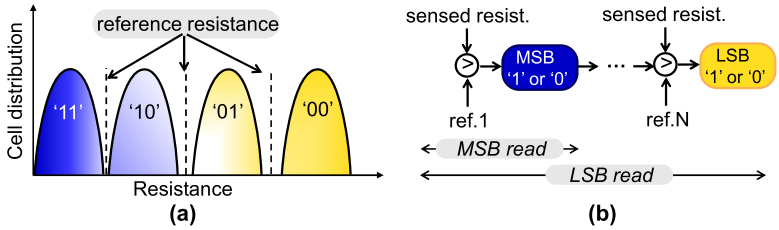

FIGURE 1. MLC PCM cell resistance range (left) and the MLC read technique (right).

architectural schemes for PCM read commonly leverage the *read asymmetry* in MLC cells for performance optimization, which allows highly variable program executions due to access to fast/slow data regions. Accordingly, we propose a novel side channel framework—R-SAW—that aims to exfiltrate program secrets by correlating victims' execution times with the PCM access patterns. We present two variants of side channel attacks: 1) R-SAW-I that targets memory read technique with PCM *interline data striping*, and 2) R-SAW-IA exploiting PCM accesses under *intraline data striping*. Our evaluation demonstrates that the newly discovered side channels are particularly dangerous: first, such attack can observe timing variance for victim's execution even under the same execution path [e.g., inferring advanced encryption standard (AES) keys]; second, R-SAW is able to carry out information leakage based on the *sub cache line* access granularity, making existing mitigations against cache line level exploits ineffective. Our work provides novel insights for future research in securing emerging NVM-based systems against side channels. In contrast to our previous work,[7] the major contributions of this article are as follows.

- We systematically model the architectural read technique under PCM intraline interleaving scheme and identify a new R-SAW side channel (R-SAW-IA) exploiting timing variations due to sub memory block access in PCM.
- We present possible code patterns that are resistant to side channels observing at the memory block granularity while still exploitable via R-SAW-IA. We evaluate the attack with the prototyped victim [based on Rivest–Shamir–Adleman (RSA)] and show that R-SAW-IA can accurately unveil secretive data from the victim.
- We perform additional characterizations for both R-SAW-I and R-SAW-IA and show that not only the proposed side channels are independent of other on-chip structures that contributes to timing observation (i.e., caches), but they are also more robust to noises. We further extend the discussion on the security of PCM with side channels.

BACKGROUND AND RELATED WORKS

Phase-Change Memories

PCM devices are built with phase-change materials that can switch between high-resistance *amorphous* state and low-resistance *crystalline* state. Due to the fact that the programmable resistance range is considerably large, it is possible to store multiple bits by encoding more than two resistance levels in a single PCM cell (i.e., MLC), which significantly increases the device capacity (see Figure 1). Accessing PCM in MLC mode, however, brings additional complexity to the cell sensing operation. Particularly, the state-of-the-art MLC sensing technique (for reads) leverages an iterative process where the resistance in one cell is compared with multiple reference values (one at a time) to decode each individual bit from the order of most significant bit (MSB) to least significant bit (LSB) (see Figure 1).

R-SAW AIMS TO EXFILTRATE PROGRAM SECRETS BY CORRELATING VICTIMS' EXECUTION TIMES WITH THE PCM ACCESS PATTERNS

With iterative sensing, it generally takes longer to derive the lower bits in an MLC cell than the higher ones—*read asymmetry*. For instance, in 2-bit MLCs, reading from LSBs is about 2× slower than that from MSB. As memory load is in the critical path, it is desirable to enable the *decoupling* of the MSB accesses with shorter latencies from the LSB accesses with longer latencies. Toward this end, the architecture community has proposed several data striping schemes and necessary *architecture support* to utilize the PCM read asymmetry for performance optimizations.[5,6] Figure 2 shows the representative data interleaving designs, including 1) bit interleaving where consecutive cache line bits are mapped to MLC cell bits

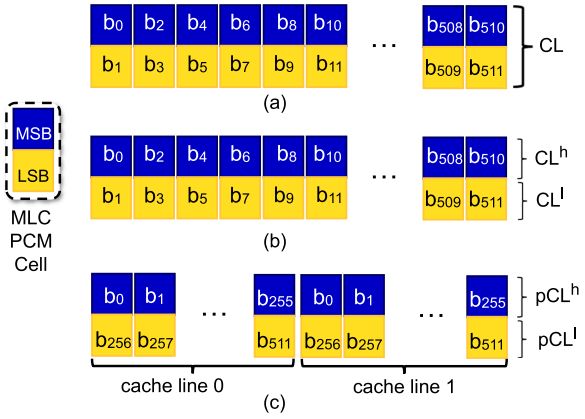

FIGURE 2. Memory block bits layout with PCM. (a) Default bit organization in a cache line (CL). (b) Interline striping with MSB-only CL (CL^h) and LSB-only CL (CL^l). (c) Intraline striping with *partial* cache line (i.e., first half) in MSBs (pCL^h) and the other half in LSBs (pCL^l).

sequentially (nonoptimized scheme); 2) interline interleaving with consecutive odd and even lines stored in MSB bits and LSB bits (speeding up odd line accesses), respectively[5]; and 3) intraline interleaving in which one half of the line maps to MSBs and the other half to LSBs (speeding up access of half of a memory block).[6]

Microarchitecture Side Channels

Microarchitecture side channel is a form of information leakage attack where illicit communication is built by an adversary through modulating microarchitecture states that influence the *timings of instruction executions* (i.e., latency) either observed through an attacker or the victim process. A variety of on-chip hardware components have been shown to be vulnerable to side channel exploitation.[2,8,9] Timing channels can be categorized into two classes: 1) *active* timing channel where an adversary intrusively perturbs states of hardware components (e.g., evicting victim's cache line); and 2) *passive* timing channel where the attacker only needs to passively observe the execution times of the victim process. While existing protection mechanisms (i.e., randomized cache[1] and resource partitioning) can defeat many of these attacks, new avenues of exploitation open up.[7]

THREAT MODEL AND ASSUMPTIONS

We assume that a victim is running processes on machines equipped with PCM as the main memory. Architectural optimizations are integrated to enhance PCM memory performance by supporting various data-interleaving schemes. The adversary can either corun a *userspace* process or interact with the victim's process through software interfaces (e.g., serving client's requests). Our investigation focuses on passive timing channels where the attacker monitors externally observable execution time of the victim and attempts to infer the secretive information. To analyze the vulnerabilities, we model a system with PCM main memories with key architecture parameters in the gem5 simulator, as given in Table 1.

SIDE CHANNELS IN INTERLINE STRIPED PCMs

Architecture Support for PCM With Interline Interleaving

We thoroughly model PCM-based systems that integrate the state-of-the-art interline striping bit arrangements in MLC PCM.[5] In this scheme, the memory controller maps consecutive memory blocks in MSBs (CL^h) and LSBs (CL^l) alternatively. Hence, in one pair of memory blocks, the first block is mapped entirely in the MSB and the second block is mapped to LSB of the same group of PCM cells. Using this bit organization, the CL^h reads are serviced quickly by the memory controller. In addition, when servicing reads to CL^l blocks, the controller searches for the paired CL^h block in cache, and if hit, the CL^l read is performed in one iteration of sensing as well.

Case Study: Attacking AES

We first demonstrate R-SAW-I, an attack that can recover keys from AES cryptographic system by exploiting interline read latency variations. Specifically, OpenSSL's implementation of AES-128 performs 10 rounds of transformation using five T-tables (T_{0-4}). The specific

TABLE 1. Architecture configurations.

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

entry accessed in tables depends on the corresponding round key byte and the intermediate input byte.

As the total number of T-table accesses in AES is fixed during each encryption run, a direct correlation between CL^l access ratio and encryption latency may exist. Since T-table access addresses are reliant on the round key, we conjecture that when performing encryptions, each particular value of key bytes will result in deterministic PCM access patterns. Based on this conjecture, an attacker can extract the exact value of key bytes by performing correlation analysis with encryption latencies for all possible 256 values of the key byte. R-SAW-I comprises the following steps.

PCM access pattern profiling on AES: During this stage, the attacker compiles memory-pattern vectors (MPVs) that are later used to determine specific key bytes in victim. The attacker first instruments the AES program to detect CL^h and CL^l line accesses during encryption. Then, it performs a sufficient number of encryptions on a local machine, using random keys and plaintexts, to generate PCM access traces. For each encryption run, a sample point $S = (C, K^{10}, p)$ is collected, recording the corresponding ciphertext C, last round key K^{10}, and the percentage of CL^l access p. These samples are then categorized based on every unique combination of k_i^{10} and C_i for each i. Particularly, we arrange all sample points with $K_i^{10} = u$ and $C_i = w$ (u and $w \in [0, 255]$) as a group $\mathcal{S}(i, u, w)$ for each value of ith key byte. This $\mathcal{S}(i, u, w)$ encodes the statistical PCM access pattern for specific values of ith key byte and ciphertext byte. Finally, we calculate the $\overline{P}^w_{(i,u)}$ by taking average of CL^l access percentage in $\mathcal{S}(i, u, w)$. The MPV for the ith byte is subsequently defined as follows:

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

Victim's execution time monitoring: The attacker triggers AES encryption on victim system using random plaintexts and records $S = (C, l)$, where C is the ciphertext, and l is the execution latency. Similar to the profile step, the collected samples are organized such that for each ith ciphertext byte, the S records for the same C_i are grouped as $\mathcal{S}(i, \mathbf{x}, w)$, where $\mathbf{x} = K_i^{10}$ is the unknown key (fixed). Subsequently, the attacker builds an encryption-timing vector (ETV) for each byte of last round key by calculating $\overline{L}^w_{(i,\mathbf{x})}$ based on *average latency* for each $\mathcal{S}(i, \mathbf{x}, w)$. The ETV captures the statistical encryption latency pattern for the unknown value of the ith key byte, and is denoted as follows:

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

AES key recovery through correlation analysis: After the ETV collection [$\mathcal{T}(i, \mathbf{x})$] is completed, the attacker performs correlation analysis of ETV with MPVs to infer the secret key value. We expect that an outstandingly higher correlation between $\mathcal{M}(i, u)$ and $\mathcal{T}(i, \mathbf{x})$ will exist for $\mathbf{x} = u$. We represent this procedure as follows:

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

Based on this, each last round key byte can be inferred by finding the u that results in the highest correlation with $\mathbf{x}$ for that specific byte. Once all bytes of last round key are inferred, the original key can be recovered.[8]

Evaluation

We first generate MPVs using 30 million encryptions in local system. In addition, we perform 128,000 encryptions in the victim to generate the ETV. From our profiling result, we observe that MPV for each key byte [$\mathcal{M}(i, u)$] corresponding to different values is differentiable. Figure 3 shows the correlation analysis of ETVs for each of the 16 key bytes. We observe that for each key byte, there exists an obvious outlier representing strong and highest correlation, which is the correct key byte value. We run this for 4,000 different victim key settings and observe that R-SAW-I achieves 98.5% accuracy.

SIDE CHANNELS IN INTRALINE STRIPED PCMs

Architecture Support for PCM With Intraline Interleaving

We model PCM intraline optimization as proposed by Arjomand et al.[6] Specifically, bits in each memory block

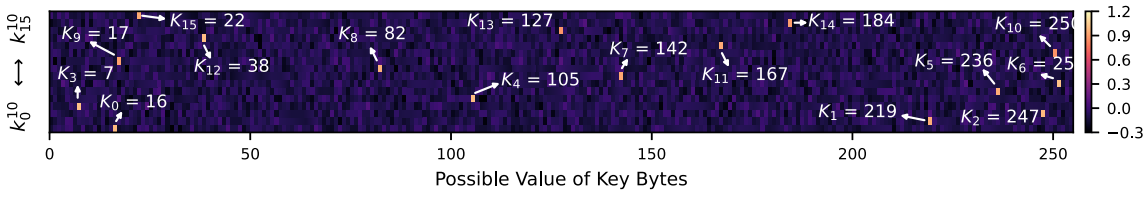

FIGURE 3. Complete recovery of the final round key. Each row denotes the correlation value distribution for one key byte.

are organized in such a way that the first half of the block [$p\text{CL}^h$ in Figure 2(c)] is stored in only the MSB of PCM cells, and the second half ($p\text{CL}^l$) in the LSB of the same PCM cells. When the processor loads a memory block from main memory, the $p\text{CL}^h$ read finishes faster and can be ready before the $p\text{CL}^l$ (see the "Background and Related Works" section). To optimize performance, the memory controller utilizes *early forwarding* of $p\text{CL}^h$ to the requesting core while only marking the outstanding cache miss as completed when $p\text{CL}^l$ arrives. This optimization improves program execution time by opportunistically hiding the LSB read latency.

Potential Side Channel Vulnerabilities

Since intraline optimization can forward $p\text{CL}^h$ earlier than $p\text{CL}^l$, executions requiring $p\text{CL}^h$ can progress while waiting for $p\text{CL}^l$. At runtime, software can issue memory access that either only maps to $p\text{CL}^h$ or $p\text{CL}^l$ of a memory block, leading to variable timings. To understand the potential impact of intraline access pattern, we design a microbenchmark that reads from arbitrary halves of memory blocks. By controlling $p\text{CL}^h$ access ratio, we observe program execution time increases linearly with $p\text{CL}^l$. We also observe that this timing correlation only exists with the presence of the intraline striping optimization. We illustrate two representative program patterns that are potentially exploitable, as shown in Figure 4.

- *Subcache line control flow divergence:* In this pattern, a secret dependent branch (inside a $p\text{CL}^l$) can transfer the control flow to either the $p\text{CL}^h$ or $p\text{CL}^l$ in the next memory block [see Figure 4(a)]. In such cases, the taken path may skip $p\text{CL}^h$ and diverge the control flow to $p\text{CL}^l$, whereas the not-taken path will execute through $p\text{CL}^h$. Since the taken path needs to execute instructions belonging to $p\text{CL}^l$ on first access, this exposes intracache line level latency variations in program execution time. The PCM read latency variations observed in this case are typically much higher than the execution time differences due to the difference between these two paths in terms of instructions.

```
1 highCtr ← 0
2 lowCtr ← 0
3
4 if s = 0
5       lowCtr++
6 else if s = 1
7       highCtr++
```

CODE SNIPPET 1: Element counter.

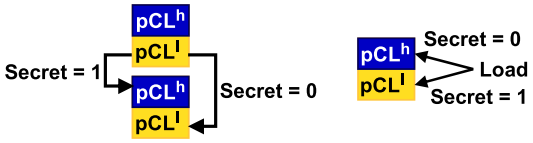

FIGURE 4. Intraline access for secret-dependent control flow (left) and data flow (right).

- *Subcache line data flow transfer:* In this vulnerable code, secret-dependent memory access can index to either a $p\text{CL}^h$ or $p\text{CL}^l$ of the same memory block [see Figure 4(b)]. Based on the execution latency, it is possible to determine which half line is indexed into, thus revealing the secret value.

R-SAW-IA VULNERABILITY CAN MANIFEST IN CASES WHERE PROGRAMS DO NOT EXHIBIT DIFFERENTIABLE ACTIVITIES AT CACHE LINE GRANULARITY.

Since the timing variance corresponds to different accesses within a memory block, such vulnerability can manifest in cases where programs do not exhibit differentiable activities at cache line granularity (e.g., classical cache attacks exploiting hit/miss[2]). Code Snippets 1 and 2 (explained later) demonstrate two representative code gadgets corresponding to the vulnerabilities in Figure 4. Specifically, Code Snippet 1 illustrates a secret-dependent data access to different halves of memory block. As shown in the gadget, if both variables highCtr and lowCtr belong to the same cache line, this gadget does not have any cache line level vulnerability. However, intracache line level vulnerability still exists in case these two data access map to a $p\text{CL}^h$ and $p\text{CL}^h$, respectively. Note that such gadgets are common in image

```
1  x ← 1
2  y ← 1
3  for i ← e − 1 to 0 {
4        x ← sqr(x)
5        x ← mod(x, m)
6        y ← mul(x, b)
7        y ← mod(y, m)
8        if e_i = 1
9              x ← y
10 }
```

CODE SNIPPET 2: Square-and-multiply-always.

processing applications (e.g., Libjpeg), where transformation of images (memory accesses) often depends on the values of neighboring pixel values.

Case Study

We present R-SAW-IA, a side channel utilizing intracache line level access pattern. At a high level, R-SAW-IA profiles the execution latency of the target application for each possible secret value (i.e., offline profiling stage). Once profiling is completed, the attacker triggers victim execution and records the execution times. Finally, a correlation analysis between the victim execution time with the attacker's profile data leaks the secretive information. To demonstrate R-SAW-IA, we analyze the modular exponentiation algorithm using *square-and-multiply-always*[10] for GnuPG's RSA implementation, as shown in Code Snippet 2. Specifically, it performs multiplication regardless of the value of the current exponent bit, and the result of multiplication operation is only kept if the exponent bit is "1." The implementation was proposed to defeat cache timing channels that identify cache line access due to the invocation of the multiplication operations in the original RSA algorithm.[2] Importantly, although there is a branch (line 8) that depends on the secret bit, the branch block is typically very small and can be placed within the same cache line as the nonsecret dependent instructions before it (i.e., line 7 and above). However, the secret dependent code can still spawn over *half of cache line* (i.e., LSB half) similar to the case in Figure 4(a). In this case, although such control flow path cannot be observed from caches (as the same cache lines would be accessed in either direction), the submemory block level observation in intraline optimization remains. The exploitation steps are as follows.

Profiling of execution latency: In this step, attacker runs RSA encryptions with different values of e, and collects the execution latency for each. The attacker then creates an execution latency profile corresponding to each number of bit "1"s in the exponent (i.e., n_e). After this stage, the attacker has an execution latency vector (ELV) that captures the execution time signature of the victim process corresponding to each n_e in e.

Collecting victim latency traces: In this step, the attacker triggers victim execution that runs RSA encryption with an unknown e. Attacker measures the execution latency corresponding to this unknown exponent.

RSA exponent secret recovery using correlation analysis: The attacker performs the correlation analysis of the victim latency traces against the profiled ELV. Since the n_e is a representative of additional $p\text{CL}^l$ access during RSA encryption, the victim program execution time is a function of the unknown n_e. For the guessed n_e whose ELV results in the highest correlation with victim traces is determined to be the number of bits "1"s in victim's exponent.

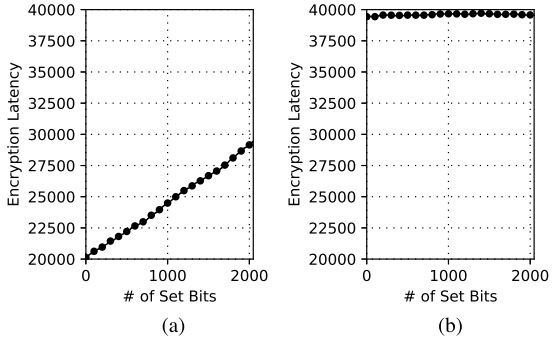

FIGURE 5. Total program execution latency as a function of # of bits "1"s in the exponent for Code Snippet 2. (a) Intraline interleaving. (b) No interleaving.

Evaluation

We evaluate R-SAW-IA by launching the attack based on Code Snippet 2. Note that as the attacker observes the entire program execution time of the victim, the timing observation collectively includes all the iterations in the loop. Figure 5(a) shows that indeed program execution time increases linearly with the increase of the n_e value. In contrast, when intraline striping optimization is not enabled, such correlation does not exist [as illustrated in Figure 5(b)]. We collect victim encryption latency traces for 1,000 different values of exponent e. R-SAW-IA can determine the n_e in each of them with 93% accuracy.

CHARACTERIZING ROBUSTNESS OF PCM SIDE CHANNELS

Characterization of R-SAW-I

Sensitivity to sample size: We evaluate the success rate (SR) by changing the number of samples taken from victim execution. Figure 6(a) shows that by increasing the number of samples per key from 25,000 to 140,000 both R-SAW-I and cache-based attacks have improvements over SR. However, we observe that given a fixed sample size, R-SAW-I consistently attains higher SR compared with the cache-based attack.

Resiliency to system noise: Along with the victim encryption process, we run a multithreaded noise injection process that continuously accesses main memory. By varying the frequency of memory accesses, we can control the noise level. We run both attacks under each

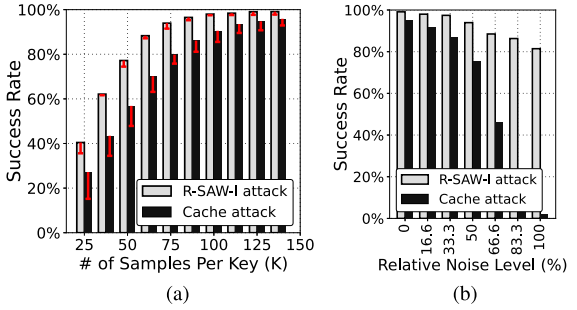

FIGURE 6. Comparative study of R-SAW-I and cache-based attack. (a) SR versus sample size. (b) SR versus system noise.

noise level and compute SR for 100 AES keys (with 128,000 sample points each). Figure 6(b) shows that SR for cache-based attack reduces drastically with the increase in noise level. Particularly, there is a sharp decrease of SR when the noise level is higher than 40%. In contrast, R-SAW-I can maintain 81% accuracy under the highest noise (i.e., nonstop memory reads). This is because while cache-based attack relies on cache hit activities (i.e., if both the ith and jth key bytes use the same T_4 entry, the encryption latency is lower because of cache hit), and cache can be heavily polluted because of the additional memory reads due to noise. In contrast, R-SAW-I relies on PCM access pattern (i.e., percentage of CL^l reads) that remains unaffected by the additional memory accesses.

Impact of on-chip caching: We model systems that either: 1) do not cache memory accesses, thus only keeping PCM access-based leakage, or 2) do not integrate PCM line striping, thus only keeping cache-based leakage. In Figure 6(a), the error bar on *R-SAW-I attack* represents R-SAW-I SR due to PCM memory access pattern only (which is only 1%–4% lower than default), and the error bar on *cache attack* represents the SR due to cache activity only (which is 2%–9% lower than default). As expected, R-SAW-I attack is possible due to secret-dependent PCM access pattern, and it is not influenced by caches.

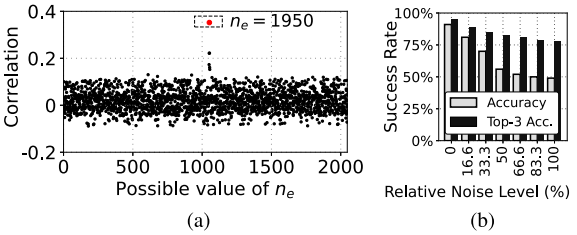

FIGURE 7. R-SAW-IA attack analysis. (a) Correlation between ELV and victim encryption latency trace. (b) Success rates versus system noise.

Characterization of R-SAW-IA

We characterize R-SAW-IA by evaluating it with the chosen plaintext attack on RSA, leaking the number of bits "1"s in the secret exponent. We choose 100 plaintexts and generate the profile ELVs from 30 million encryptions, as discussed in the "Case Study" section. Then, we collect 1,000 victim encryption latency traces for each of the 100 plaintexts. Finally, we run correlation analysis of the victim trace against the ELVs. For example, Figure 7(a) shows that the correlation value is the highest when n_e is 1,950, which represents the correct n_e in victim. This highlights the strong correlation between n_e and overall encryption latency that is directly caused by intraline optimization of PCM reads.

Impact of system noise: Similar to the R-SAW-I, we define six levels of system noises along with the noise-free configuration to quantify the noise resiliency of R-SAW-IA. Figure 7(b) shows that even with the highest degree of noise, R-SAW-IA observes a reasonable 78% top three accuracy (i.e., the correct n_e is in one of the top three correlations). As the pCL^h to pCL^l access ratio remains unaffected by the additional memory accesses, R-SAW-IA is less susceptible to noise. We note that with intraline interleaving, regardless of which half of the memory block is accessed, both pCL^h and pCL^l reads are performed in memory to return the complete memory block to the processor. Hence, the performance benefits of intraline interleaving mainly come from early execution of instructions utilizing pCL^h. In contrast, with interline interleaving, memory reads can be terminated early if CL^h is read, which results in both early execution of instructions and higher memory throughput for applications with CL^h reads. This results in R-SAW-I observing higher degree of read latency variations compared with R-SAW-IA. Nevertheless, R-SAW-IA is still capable of exploiting the intraline latency variations with high noise resiliency.

DISCUSSIONS OF MITIGATION

Randomized PCM data mapping: One potential way to mitigate the attack is to randomize memory block mapping to MSBs and LSBs using architectural support in memory controller. For interline striping, MSBs and LSBs can be remapped to new locations on the same page using a permutation seed generated at runtime. For intraline striping, instead of mapping first half of a block to MSB and second half to LSB, they can be remapped to different halves in the same block randomly. This will make the memory-access pattern randomized, breaking the correlation with execution latency. However, this scheme might require frequent changes of the randomization seed to prevent potential reverse engineering of the mapping.

Software hardening: Software optimization is also be one potential mitigation. Prior research has investigated rewriting the software to ensure information safety (e.g., preventing secret dependent branching), which can be adopted to prevent R-SAW. Specifically, security-critical sections in applications can be allocated to memory locations with similar latency groups to prevent secret-dependent PCM access latency. However, adapting such PCM latency region-aware mapping techniques in software can introduce nontrivial complexity in software design.

CONCLUSION

In this article, we investigated the information leakage vulnerabilities in MLC PCM systems. We found that PCM access techniques leveraging read asymmetry in MLCs introduced new side channel attacks. We presented two variants of attack, targeting both interline and intraline interleaving optimizations. 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.

ACKNOWLEDGMENTS

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

REFERENCES

1. F. Liu and R. B. Lee, "Random fill cache architecture," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 203–215.
2. Y. Yarom and K. Falkner, "FLUSH RELOAD: A high resolution, low noise, l3 cache side-channel attack," in *Proc. USENIX Secur.*, 2014, pp. 719–732.
3. M. H. I. Chowdhuryy, M. R. H. Rashed, A. Awad, R. Ewetz, and F. Yao, "LADDER: Architecting content and location-aware writes for crossbar resistive memories," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2021, pp. 117–130.
4. B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. ACM/IEEE Int. Symp. Comput. Archit.*, 2009, pp. 2–13.
5. M. Hoseinzadeh, M. Arjomand, and H. Sarbazi-Azad, "Reducing access latency of MLC PCMs through line striping," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, 2014, pp. 277–288.
6. M. Arjomand, A. Jadidi, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "HL-PCM: MLC PCM main memory with accelerated read," *IEEE Trans. Parallel Distrib. Syst*, vol. 28, no. 11, pp. 3188–3200, Nov. 2017.
7. M. H. I. Chowdhuryy, R. Ewetz, A. Awad, and F. Yao, "Seeds of SEED:R-SAW: New side channels exploiting read asymmetry in MLC phase change memories," in *Proc. IEEE Int. Symp. Secure Private Execution Environ. Des.*, 2021, pp. 22–28.
8. J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Proc. Cryptographic Hardware Embedded Syst.*, 2006, pp. 201–215.
9. M. H. I. Chowdhuryy and F. Yao, "Leaking secrets through modern branch predictor in the speculative world," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2059–2072, Sep. 2022.
10. GPG, "Mitigate a flush reload cache attack on RSA secret exponents," 2013. Accessed: Apr. 10, 2022. [Online]. Available: https://github.com/gpg/libgcrypt/commit/e2202ff2b

MD HAFIZUL ISLAM CHOWDHURYY is a Ph.D. student at the University of Central Florida, Orlando, FL, 32816, USA. His research focuses on computer architecture with a focus on security. He is a Student Member of IEEE. He is the corresponding author of this article. Contact him at reyad@knights.ucf.edu.

RICKARD EWETZ is an associate professor in the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 32816, USA. His research interests include physical design and computer-aided design for in-memory computing using emerging technologies. He is a Member of IEEE. Contact him at Rickard.Ewetz@ucf.edu.

AMRO AWAD is an assistant professor in the Electrical and Computer Engineering Department, North Carolina State University at Raleigh, Morrisville, NC, 27560, USA. His research interests include secure hardware architectures and memory systems. He is a Member of IEEE. Contact him at ajawad@ncsu.edu.

FAN YAO is an assistant professor in the Electrical and Computer Engineering Department, University of Central Florida, Orlando, FL, 32816, USA. His research interests include computer architecture, hardware, and system security. He is a Member of IEEE. Contact him at fan.yao@ucf.edu.