Are Crossbar Memories Secure? New Security Vulnerabilities in Crossbar Memories

Vamsee Reddy Kommareddy, Baogang Zhang, Fan Yao, Rickard Ewetz and Amro Awad

Abstract—Memristors are emerging Non-Volatile Memories (NVMs) that are promising for building future memory systems. Unlike DRAM, memristors are non-volatile, i.e., they can retain data after power loss. In contrast to DRAM where each cell is associated with a pass transistor, memristor cells can be implemented without such transistor, and hence enable high density ReRAM systems. Moreover, memristors leverage a unique crossbar architecture to improve the density of memory modules. Memristors have been considered to build future data centers with both energy-efficiency and high memory capacity goals. Surprisingly, we observe that using memristors in multi-tenant environments, e.g., cloud systems, entails new security vulnerabilities. In particular, the crossbar contents can severely affect the write latency of any data cells within the same crossbar. With various memory interleaving options (to optimize performance), a single crossbar might be shared among several applications/users from different security domains. Therefore, such content-dependent latency can open new source of information leakage. In this article, we describe the information leakage problem in memristor crossbar arrays (MCAs), discuss how they can be potentially exploited from application level. Our work highlights the need for future research to mitigate (and potentially eliminate) information leakage in crossbar memories in future computing systems.

Index Terms—Crossbar memory, ReRAM, security.

1 Introduction

With poor scalability and energy inefficiency of DRAM devices, the search for alternative memory technologies has been a major research interest in the past decade. Fortunately, emerging Non-Volatile Memories (NVMs) promise a new set of memory architectures that can potentially replace/augment DRAM in future computing systems. Several emerging NVMs, such as Phase-Change Memories (PCM) and Memristors, offer very high densities and better power efficiency. Moreover, such technologies can potentially retain data after power loss, and hence enable persistent applications, e.g., file-system hosting and data check-pointing. Among different classes of NVMs, memristor-based memories are particularly promising due to their low latency, high write-endurance, density, and low idle power. Many system vendors envision future data centers with memristors as the main memory (and storage). For instance, HPE Labs' The Machine project has its system architected with memristor-based memory as it's shared global memory [1].

As memristor-based memories are expected to build the memory systems of future computing systems, understanding its potential security vulnerabilities is of utmost importance. Common across all NVMs, data remanence has been considered traditionally as the major security challenge [4]. However, *the inherent design-based characteristics* of NVM devices can open new security vulnerabilities. The lesson we learned from the recent rowhammer attacks [6] in DRAM memories is the importance of understanding the potential implications of hardware design choices on security. For rowhammer attacks, the potential cross-talk effect between physically adjacent DRAM rows has been overlooked at the design stage. The findings of the rowhammer vulnerabilities and its burgeoning system exploitation (e.g., [8]) are alerting computer architects to consider the security of emerging memory technologies as the first-order design constraint. Unfortunately, there are very limited studies that explore potential vulnerabilities in NVMs and in particular memristor-based memories.

In this article, we identify a major vulnerability in memristor-based memory which can be potentially exploited to construct information leakage attacks. Our key observation is that the write latency of a memory cell in a crossbar memory is *highly dependent* on the state of their neighbouring memory cells in the same crossbar, due to sneak currents in ReRAM [11]. Consequently, the timing differences in write operations in crossbar memories can be exploited by adversaries to exfiltrate sensitive information that belongs to other security domains. To the best of our knowledge, this is the first work that presents a major source of information leakage in memristor-based crossbar architectures. To validate our observation and understand the extent of write latency difference associated with different content in crossbar memories, we build a detailed memristor-based memory model and integrate it into a cycle-accurate architectural simulator to measure the observed latencies from user space. Surprisingly, the latencies observed at the application level can reflect the state of other cells in the same crossbar, which can be possibly exploited to construct either covert or side channels. With such a new vulnerability, we urge the research community to carefully study both hardware and system security solutions in order to mitigate and hopefully disable such information leakage channels for future computing systems. In summary, major contributions of this article are:

- We make the key observation about the deterministic relationship between the write latency of a ReRAM memory cell and the content of its neighboring cells in the same crossbar. We focus on security implications of such timing property for future ReRAM-based computing systems.
- We explore and illustrate how attackers could leverage the content-based access timing characteristics to construct a new class of timing channel technique—*WRITE+TIME*.
- We build a memristor-based memory model using circuit-level simulations and incorporate it into the cycle-accurate

- *V.R.Kommareddy, B.Zhang, F.Yao, R.Ewetz and A.Awad are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida. Email: {vamseereddy8, baogang.zhang}@knights.ucf.edu, {fan.yao, rickard.ewetz, amro.awad} @ucf.edu.*

Manuscript submitted: 08-Sep-2019. Manuscript accepted: 24-Sep-2019. Final manuscript received: 01-Oct-2019

gem5 simulator. Our evaluation shows that adversaries can successfully observe distinguishable latency patterns from user space, which can be leveraged to carry out information leakage attacks.

2 Threat Model

In this article, we study information leakage threats in resource-sharing environments (e.g., the cloud) that are equipped with memristor-based memory devices. Our attack model assumes that there is a benign victim or a malicious trojan process with access to sensitive information. At the same time, there exists a spy who is trying to steal or infer the secrets. Both parties run unprivileged processes. The trojan/victim and the spy processes belong to different security domains, and thus they are not supposed to communicate with each other directly. We further assume that the operating system is trusted and secure.

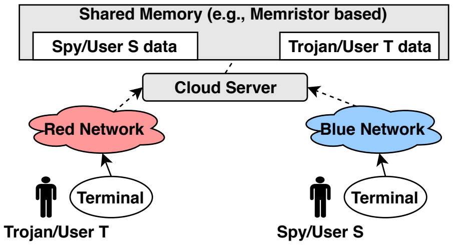


Fig. 1: Threat model of memristor-based memory systems

To better understand the attack model, we demonstrate an adversary scenario as depicted in Figure 1. We assume User T (trojan) is malicious and trying to send out certain restricted information, e.g., a cryptographic key, to User S (spy), however, without leaving any trace. Note that User T would not be able to do this through the legitimate system interfaces as it is not allowed to send restricted data outside the red network per the security policy. If User T can somehow send information to User S within the server, then User S can read it directly and push it through the blue network. Particularly, since all users share the memory modules in the cloud server, if User T and User S can communicate through exploiting the value dependent ReRAM access latency, then eventually User T can get the data and send it out. Moreover, even if User T is not malicious, if there is a way that User S can read User T's data indirectly, e.g., side-channel attacks, then the data can be sent through the unrestricted network.

3 Background

Resistive RAM Architecture. Resistive RAM (ReRAM) refers to the concept of using resistance to store information. ReRAM cells have a low resistance state (LRS) and a high resistance state (HRS), which are used to represent the logical bit '1' and '0', respectively. ReRAM memory cells are read by applying a small voltage across the devices and measuring the output current. The state of a ReRAM cell is switched by applying a write voltage with a specific polarity, magnitude, and duration. The switching of a ReRAM cell from HRS to LRS is called SET and the switching of a ReRAM cell from LRS to HRS is called RESET.

ReRAM cells are integrated into dense memory structures to build denser memory. The most common memory structure is the crossbar architecture with selector accesses devices, i.e., there is selector device connected in series with each ReRAM cell. Compared with crossbar architectures with access transistors, each selector based accessed memory cell can fabricate the theoretical minimum area of $4F^2$, i.e., where F is the minimum feature size of the technology. Different from crossbar architectures without access devices, the selector devices enable the crossbar structures to be scaled to larger dimensions, which reduces overhead (current leakage) and cost.

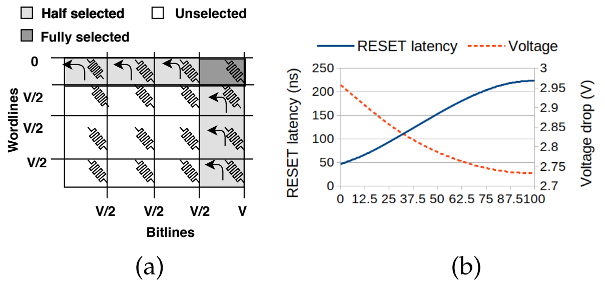
Hardware-based Information Leakage Attacks. Covert and side channel are a form of information leakage attack where illicit communication is constructed between two parties via system resources that are not intended for transmitting information [3]. In side channel attacks, a benign application (victim) unknowingly leaks sensitive data to a malicious spy. In the scenario of covert channels, there exists a malicious insider trojan process who collude with a spy process to reveal secrets illegitimately. Both covert channel and side channel can manifest as timing channel where the timing of accesses to certain shared hardware resources are manipulated to exfiltrate secrets [9]. Besides timing, location in the storage devices can also be utilized to build illegal communications through writing of bits by one program and reading of those bits by another using a stealthy encoding scheme.

4 ReRAM Access Latency Traits

A ReRAM memory cell can be activated for a read or write operation by selecting the corresponding wordline and bitlines. In ReRAM crossbar architectures, when a cell is activated, the current will not only flow through the activated cells (called *fully-selected*) but also through all the other cells along the selected wordline and bitline(s), which are called *half-selected*. The currents through the half-selected cells are called *sneak currents*. The currents introduced during a RESET operation are illustrated in Figure 2(a). The fully-selected cell in the figure is RESET by setting the corresponding wordline and bitlines to 0 and V, respectively. All other wordlines and bitlines are set to $V/2$ to minimize the sneak currents. When voltages are applied to wordlines and bitlines of a crossbar, a voltage drop will occur across the memristor at each intersection, which can be converted to latency by Equation 1, where t is the switching time, V_d is the voltage drop, k and C are constants [11].

$$t * e^{kV_d} = C \tag{1}$$

In a prior work [10], it was observed that the latency of the RESET operation was dependent on the number of LRS cells along the selected wordline. To fully understand the latency profiles for RESET (write) operation, We performed a characterization study using circuit-level simulations (See Section 7 for more details). Figure 2(b) shows how the RESET latency and voltage drop vary as the percentage of bit '1's (i.e., LRS) is set from 0% to 100% at row 0. From the figure, it is observed that as the number of cells increases, the RESET latency varies from 46ns to 224ns, and the IR drop varies from 2.96V to 2.73V. Clearly, the required latency for the RESET operation is variable and dependent on the location and content of neighbouring memory cells. By storing the number of LRS cells on each wordline using a counter, the latency of the RESET operation can be reduced with up to $4.8\times$. Without the use of the LRS counter, the worst case latency is required to be used for every RESET operation, which will detrimentally degrade the performance in both latency and power. Note that such latency variations are specifically observed for RESET operations, since switching time for memristors to SET a cell is typically very short.



(a) (b)

Fig. 2: (a) ReRAM 4x4 crossbar architecture with sneak paths due to half selected cells. (b) RESET latency and Voltage drop variation at Row 0 with different LRS percentage (%)

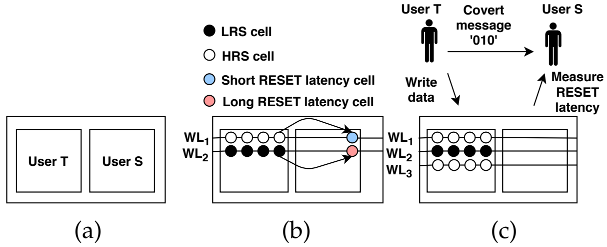
5 Vulnerability to Information Leakage

In this section, we demonstrate how the variable write latency can be used to perform a covert channel attack. We continue the motivational example in Figure 1, where Users T and S have access to parts of a shared ReRAM based memory. Figure 3 shows how User T and User S manage to manipulate the timings of write access to ReRAM cells to build a stealthy communication channel.

In order to carry out a covert channel attack, User S and User T first attempt to map memory space such that their memory allocations achieve the setup as shown in Figure 3a. That is, User S and User T co-locate their data on several ReRAM rows in the same crossbar. Meanwhile, in order for User T to manipulate the observed access latency for User S, User S and User T place their data on the right half and left half of the crossbar, respectively. It is worth noting that such memory setup can be easily realized using memory massaging techniques that have been demonstrated in DRAMs [7]. Once the memory setup is completed, User T can pass data to User S by exploiting the latency of the RESET operation. We call this technique **WRITE+TIME**, which includes the following two major steps:

1) For a co-located row in certain crossbar, the trojan first writes a specific data pattern to the ReRAM cells in its assigned half of the row (left). For example, to send bit '1', the trojan can set its memory cells to all "1's", and to transmit bit '0', it sets all "0's" to the left side of the row.
2) The spy then issues *write* operations to one memory cell in the right half of the row, and *times* the latency of the write operation. Based on the communication protocol, the spy learns a bit '1' if it sees a longer latency and bit '0' if the latency is short. The spy then keeps idle for a short period of time for the trojan to prepare the next bit.

Note that it is possible to leverage multiple rows for the covert communication. For example, User T can write "0's" and "1's" to the ReRAM cells on wordline WL_1 and WL_2. To receive secrets from User T, User S *writes* (or RESET) data to WL_1 and WL_2 and *times* this operation. Obviously, the latency of the write operation to WL_1 will be significantly shorter than the latency of the write operation to WL_2 because there are more cells in LRS along WL_2 than WL_1 as illustrated in Figure 3(b). Therefore, with WRITE+TIME, User S can determine that User T has written "0's" or "1's". For instance, using the same communication protocol as described above, User T could pass a three-bit secret ("010") to User S using wordlines WL_1, WL_2, WL_3 (Figure 3c). While this example shows a scenario for covert channel, we note that it is straight forward to perform side-channels where a spy infers the contents of a victim's data by using the same memory setup.



(a) (b) (c)

Fig. 3: (a) shared ReRAM based memory for users. (b) the write latency for user S is dependent on the content of user T's memory. (c) User T passing a message "010".

6 Experimental Setup

We used HSPICE [5] to simulate ReRAM model and the Gem5 architecture simulator [2] to evaluate a system with ReRAM latencies.

TABLE 1. Simulated ReRAM Parameters

Metric	Description	Specification
M	input dimension	512
N	output dimension	512
n	number of bits to write simultaneously	8
R_w	wire resistance	2.5Ω
K_r	Nonlinearity of the selector	200
V	Full selected voltage	3.2V

Crossbar Configuration. The parameters used for the crossbar configuration are listed in Table 1. In our design, the worst-case data pattern is assumed to evaluate the time latency, i.e. we always assume all n-bits need to be RESET in an M × N crossbar since the RESET latency is determined by the number of bits that transforms from "1" to "0", and compared to which the SET latency is negligible [11].

ReRAM Organization. We assume that the future ReRAM DIMMs are organized similar to DRAM organization. Each DIMM has 2 ranks and each rank contains multiple banks. Multiple ranks and banks can be accessed simultaneously. Within a bank, data is stored in multiple ReRAM crossbars, [10] which can also be accessed in parallel. Each crossbar has a dedicated write driver, wordline and bitline selectors. We modeled a 512×512 crossbar organization and calculated latency in writing to the farthest cell from write driver, 512^{th} cell of the 1^{st} row, by varying the half selected cell states from 0% cell with LRS to 100% cells with LRS.

ReRAM Crossbar Simulation. A summary of the parameters used in a single ReRAM crossbar simulation are provided in Table 1. The latency of the write operation is a function of the voltage drop over the fully-selected devices. Consequently, the node voltages in the crossbar must be determined given the state of all the ReRAM cells. The voltage at each node on the crossbar can be obtained by formulating a system on linear equations using Modified Nodal Analysis (MNA), as follows:

$$Y(g,p)\begin{bmatrix} v_{xbar} \\ v_{wl} \\ v_{bl} \end{bmatrix} = \begin{bmatrix} 0 \\ v_{wl,w} \\ v_{bl,w} \end{bmatrix}, \tag{2}$$

where $Y(g,p)$ is a matrix with dimensions $(2 \cdot N \cdot M + M + N) \times (2 \cdot N \cdot M + M + N)$ that is a function of g and p. g contains the state of all the ReRAM cells (LRS/HRS); p captures the parasitic parameters of the crossbar. The selector devices are modeled by scaling the resistance values of the non-selected and half-selected cells with K_r. M and N are the number of inputs and outputs, respectively. v_{xbar}, v_{wl}, and v_{bl} are the node voltages in the crossbar, wordlines, and bitlines, respectively. $v_{wl,w}$ and $v_{bl,w}$ are the voltages applies to the wordlines and bitlines, respectively. The node voltages v_{xbar} are obtained by solving the system in Equation 2. Next, it is straightforward to calculate the write latency using Equation 1.

Gem5 Simulation. We build a ReRAM memory access latency model based on our HSPICE simulations and integrate it with gem5. The memory access timing model is implemented as a lookup table that details expected latency for write operation given the data pattern of the corresponding crossbar row. We evaluated the WRITE+TIME strategy by simulating a single out-of-order X86 core in system-call emulation (SE) mode. We set up the memory mapping as described in Section 5, and configure the data pattern (first half of the rows in a crossbar) before simulating the spy.

7 Evaluation

In this section, we study the timing of write latency for a malicious spy process running in the system with ReRAM memories. To validate our observation, it is necessary to show that the content-based variable write latency in ReRAM cells can be actually observed from user space.

We implement a set of micro-benchmarks to evaluate the timings of write operation corresponding to different data patterns. Specifically, to collect latency of write operation, we launch a latency-profiling thread that executes a load instruction immediately after the target *store* (write) instruction to the same memory address. Due to the read-after-write data dependency, the read operation has to stall before the write request is completed. We, therefore, measure the time (in cycles) it takes to execute write and read code sequence using the `rdtsc` instruction, indicated as WRITE+TIME strategy. To make sure that the data is not present in all levels of caches, before the write operation, the target cache line is flushed using the `clflush` instruction that is available in x86 architecture. In our simulations, we indicate *memory access delay* as the delay observed by the application before flushing the data from the caches till the read request responses back.

We achieve the memory layout as depicted in Figure 3 by allocating a large chunk of memory in user space. By checking the physical page number together with the memory addressing scheme, the program is able to locate memory addresses that map to the left and right side of certain crossbar rows. We use the *right-side* memory in the latency-profiling thread (i.e., the spy) for latency sensing and the *left-side* memory to set the data patterns. To avoid factors like noises from other sources, read and write queues which alter crossbar *memory access delay* for the users, we perform write-and-read operations iteratively, from 1 to 100.

Figure 4 demonstrates the latency profiles observed when the percentage of cells with LRS states in the left half of the crossbar row vary from 0 to 100%. As we can see, the latency-profiling thread does not observe much variation in the *memory access delay* when there are only a few iterations of operations (e.g., < 5). This is because with short memory access sequences, the read and write queues can hold all the pending requests. As the number of such iterations increases, we can observe that the *memory access delay* is directly proportional to the data patterns at the half selected cells (e.g., 100 iterations). Due to multiple iterations and persistent domain off the processor chip, which persists writes to the memory, the channel capacity can be maintained high and the user observed memory latency is not affected by NVM write and row buffers.

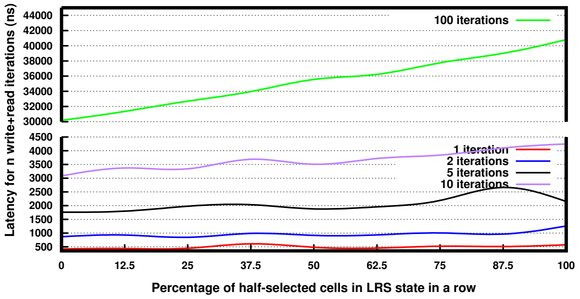


Fig. 4: Memory access latency observed by the application.

8 Conclusion

In this article, we define an emerging security threat to the data stored in the ReRAM when multiple user applications are sharing one ReRAM device. A latency tool is developed to obtain the latency map of the ReRAM crossbar for the write operation. We study the delay in accessing the memory locations based on the data patterns of half selected cells using the developed latency tool. For a simple case wherein the row of a crossbar is shared by 2 applications we show the memory access latency from the application side. This can lead to timing-based information leakage attack where an adversary manages to exfiltrate secrets of a memory-colocating process via write operation latencies. We validate the observable latency by running comprehensive simulation on gem5 and our results show that user space application can successfully both distinguishable latency corresponding to the content of memory cells in the same row.

Acknowledgments

This work was supported in part by National Science Foundation (NSF) under awards CNS-1814417, CNS-1908471 and CCF-1755825.

References

[1] "The Machine Project," 2019, https://www.labs.hpe.com/memory-driven-computing.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[3] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018.

[4] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[5] R. HSPICE, "Reference manual: Commands and control options," Version D-2010.03-SP1, June 2010. http://www. synopsys. com/Tools , Tech. Rep., 2012.

[6] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ISCA*. IEEE Press, 2014.

[7] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-cpu attacks," in *USENIX Security*. USENIX Association, 2016.

[8] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, 2015.

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

[10] W. Wen, L. Zhao, Y. Zhang, and J. Yang, "Speeding up crossbar resistive memory by exploiting in-memory data patterns," in *ICCAD*. IEEE, 2017, pp. 261–267.

[11] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015, pp. 476–488.