SoK: Rowhammer in the Post-TRR Era

Pradyumna Shome
Georgia Institute of Technology
Atlanta, GA, USA
pradyumna.shome@gatech.edu

***Abstract*—Rowhammer is a reliability and security issue that has plagued DRAM modules ever since its introduction in 2014. Close packing of DRAM cells within a rank means that when rows are repeatedly accessed, cells in neighboring rows encounter bit flips beyond what error-correcting code (ECC) memory can recover. Over the past years, many attacks have been proposed leveraging this as a write primitive that circumvents process isolation through virtual memory.**

As a countermeasure, DDR4 and LPDDR4 chips included a mechanism known as target row refresh (TRR), which refreshes neighboring rows when thresholds are exceeded. However, starting with the disclosure of TRRESPASS, a fuzzing tool that produces memory access patterns overcoming TRR, there has been a deluge of new attacks and countermeasures that have forced computer architects to rethink their solutions. In this work, we perform a systematic review of new developments in the space and discuss the broader implications for hardware designers.

***Index Terms*—rowhammer, DRAM, attacks, defenses, hardware security**

I. Introduction

Computer scientists have long acknowledged the reliability limitations of hardware. Cosmic rays and arbitrary faults have been known to cause bit flips in DRAM, whose effect is particularly significant at datacenter scale in the cloud computing era.

Rowhammer, discovered in 2014 [10], has been one burgeoning problem that has garnered much interest in academia and industry. It is a disturbance effect whereby many accesses to a row of DRAM deplete the charge of cells in adjacent rows, thus causing bit flips [10]. For a given row, the likelihood and number of bit flips is magnified if both adjacent rows are repeatedly accessed. Follow-up work identified the devastating security implications, by triggering the effect as a write primitive, bypassing isolation mechanisms such as virtual memory, and writing to privileged memory. Rowhammer-based attacks have been used to mount attacks on native systems, mobile platforms [20], virtual machines, web browsers [5], trusted execution environments [6], and across the network [14] in addition to poisoning deep neural networks [23] and stealing encryption keys from kernel memory [12].

While ECC-RAM can correct a single bit, induced flips often affect multiple bits in a given word. After a spate of attacks on a variety of platforms, a slew of detect-and-refresh based countermeasures were proposed in the computer architecture literature [15, 13, 19, 9]. Most of these schemes use counters to track memory accesses to groups of addresses. When the access count to a given row exceeds a given threshold, they issue a refresh (activation) on adjacent rows, which recharges them and prevents them from being flipped. Eventually, target row refresh (TRR) was commoditized in DDR4 and LPDDR4 chips, which we discuss in II-B

Unfortunately, this was woefully insufficient and proved to be easy to exploit. The 2020 release ofTRRESPASS[4] showed that TRR was not the endgame. It is a software program that produces memory access patterns that cause bit flips even with TRR. This marked the beginning of a deluge of more creative attacks that overcame TRR, what we refer to as the "post-TRR" era.

In this paper, we elucidate broad themes underlying recent attacks, and new defenses that break attack requirements from first principles. Finally, we discuss broader implications of this DRAM cat-and-mouse game on microarchitects and hardware vendors.

II. Background

A. DRAM Organization

Dynamic Random Access Memory (DRAM) is a memory technology where a bit of information is stored in a *cell* that contains a capacitor and transistor to represent data words. As can be seen in Figure 1, many cells are organized into a *row*, and many rows are associated with a *rank*. Rows of DRAM need to be refreshed every 32ms to 64ms to maintain charge, by performing what is known as an *activation*.

To access memory, the voltage of a word line in a row needs to be raised. This change is detected and amplified by a row buffer, thus activating the row. Based on the commonly used open row policy, as long as memory is being read from the same row, the row buffer remains activated, which results in a *row hit*. If a word from a different row needs to be read, the former row needs to be closed and the latter row be activated, which is slower and referred to as a *row conflict*. To reduce the frequency of row conflicts and correspondingly reduce memory access latency, rows in a rank are also grouped into *banks*, with each bank having an associated row buffer. Therefore, a row buffer is analogous to a cache.

When a processor issues a memory request that misses in all levels of the cache, the translated physical address is run through a bit permutation, after which it is passed to a memory controller for data to be fetched *f*rom DRAM. Here, consecutive ranges of bits in the permuted address are used to determine a channel ID, rank ID, bank ID, row ID, and column ID (to uniquely identify a cell within the row). Channels allow multiple memory request to be served from

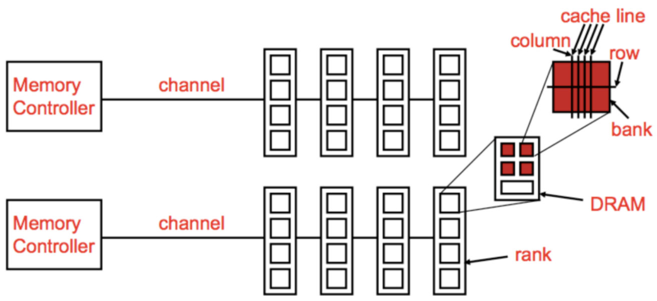

Fig. 1. The components in modern DRAM. Image from [16]

.

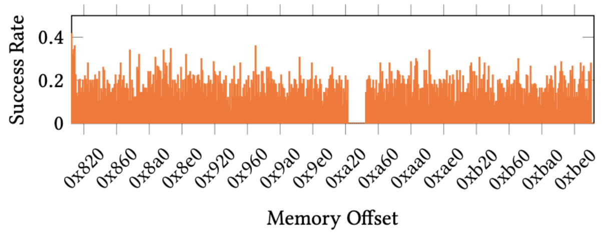

Fig. 2. Using the low success probability of rowhammer to correctly guess the start of the .ktext section (despite the presence of KASLR) in HammerScope [3].

potentially different locations in DRAM (e.g. different modules) simultaneously, thus increasing the memory bandwidth.

B. *Target Row Refresh*

A mitigation deployed on DDR4 and LPDDR4 DRAM, Target Row Refresh has the memory controller (which services all requests to DRAM) track victim rows and issue refreshes when the number of accesses exceeds a threshold (one that is below the rowhammer threshold R_H, which is the number of activations needed in a time interval to induce a flip). Pietro Frigo et al., the authors of TRRESPASS [4], reverse engineered several DIMMs and found that there was not a standard implementation of the mitigation, implying that vendors relied at least a bit on security through obscurity. There are several known versions of TRR in production, including Intel's pseudoTRR (whose implementation is not public, and was reverse engineered in TRRESPASS) and In-DRAM TRR which embeds tracking in DRAM circuitry.

III. ATTACKS

TRRESPASS [4], HAMMERSCOPE [3], Half Double [11], and Blacksmith [7] are recent rowhammer attacks that attempt to circumvent specific assumptions that TRR makes: (1) that specific patterns of victim and aggressors are needed, (2) that issuing refreshes to selected victim rows can only reduce flips, and (3) that only integrity is affected but not confidentiality. In aggregate, these attacks underscore a broader insight widely known in classical software security: kludges and point defenses will eventually be overcome by creative attack techniques. Architects attempting to develop robust countermeasures need to use a more principled approach that comprehensively ensure confidentiality, integrity, and availability.

A. *TRRespass*

In the software testing community, fuzzing is a popular approach which runs a program with a large number of randomly generated inputs to trigger bugs. TRRESPASS is a fuzzer that produces a pattern of memory accesses that can cause flips for a given victim row. The big idea explored in this work is that of *many-sided Rowhammer*. Since the number of aggressor rows that can be tracked is limited by the *sampler size* (in practice, around 4 to 6) and can issue only a single refresh every interval, many-sided Rowhammer seeks to overwhelm the tracking mechanism of the mitigation by spreading out accesses across 20+ aggressor rows with less than 50,000 activations, such that neighboring aggressor rows are separated by a victim row. The authors perform a case study of a DIMM manufactured by one of the three major DRAM vendors (Micron, SK Hynix, and Samsung) in which they reverse engineer the TRR implementation, finding that the mitigation is applied on every refresh command, can sample multiple victim rows, but can refresh only one.

B. *HammerScope*

Unlike most Rowhammer-based attacks that target integrity, HammerScope compromises confidentiality. It is an attack based on the insight that the number of activations needed to flip a bit is correlated with the DIMM's power consumption. An adversary that attempts to mount a Rowhammer attack on one core can learn information about the memory activity of a victim program on another core. The variance in levels of DRAM activity allows a cross-core adversary to distinguish between different potential DIV (division) operands, instruction types, and whether an address is mapped, all within a victim program.

One attack undermines Kernel Address Space Layout Randomization (KASLR), by using the execution time of prefetch commands on a large number of addresses to determine the starting address of the kernel text segment (which stores kernel instructions), thus enabling code reuse attacks (see Figure 2. Another attack monitors DRAM memory activity when accessing elements of a Spectre probe array with each element (a guess of the secret byte value) flushed from the cache – the guess is correct if there is much lower DRAM activity, since the speculative access is served from the cache. The authors propose several possible explanations for the reduced efficacy of Rowhammer on high DRAM consumption: increased DRAM power consumption (1) increases the slew rate, (2) decreases the voltage of the aggressor word line, and (3) produces interleavings of Rowhammer access patterns with non-Rowhammer access patterns that reduce its efficacy.

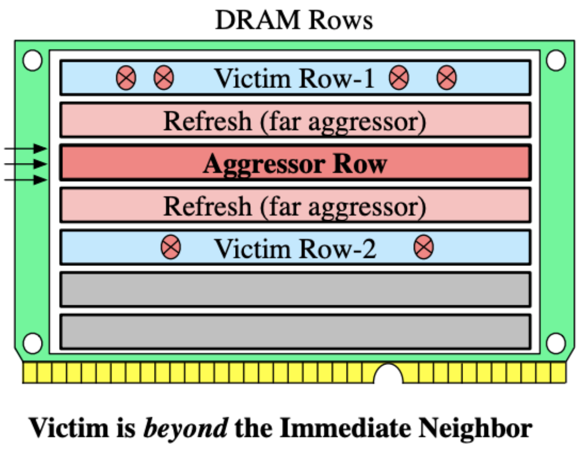

Fig. 3. Visual description of the Half-Double attack [11]. Image from Aqua [18].

C. Half Double

This attack [11] features many accesses to rows at distance 2 from the victim row (”far aggressors”, followed by a few accesses to rows at distance 1 (”near-aggressors”). The key insight is that refreshing a row, be it as a mitigation or otherwise, is *equivalent* to to activating it (i.e. writing over the row with the existing contents).

The large number of accesses to far aggressors causes TRR and related countermeasures to activate near-aggressors several times, which are not counted towards the threshold. These cause the victim row to lose charge. Finally, some more accesses to near aggressors (but below the threshold) cause more voltage drain on the victim row, which results in bit flips. This is combined with a Spectre-attack to speculatively verify that bits were flipped.

D. Blacksmith

The authors of Blacksmith [7] empirically tested a plethora of memory access patterns to understand the properties fundamental to a successful rowhammer access pattern. This led them to wonder if a uniform pattern of aggressor hammering was strictly necessary for bit flips to occur. Exploring the large search space of non-uniform access patterns made them realize the three properties essential to an attack: *order*, *regularity*, and *intensity*, which are correlated in the frequency domain to phase, frequency, and amplitude, respectively.

They begin by extending TRRESPASS’ multi-sided patterns by adding random accesses to bypass tracking. To accelerate the search for vulnerable non-uniform patterns, they devised the notion of an aggressor tuple consisting of several aggressors. Each aggressor tuple is associated with a frequency (how often a tuple is accessed within a pattern), phase (the elapsed time from the start of a pattern before a tuple is accessed), and amplitude (how long a particular tuple is hammered consecutively). Varying some of these parameters while holding others constant allows them to create patterns

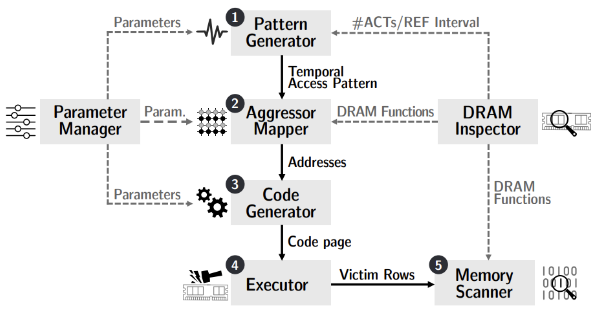

Fig. 4. Image describing the Blacksmith’s architecture from [7].

that are challenging to explore manually, which ultimately allows the tool to trigger bit flips when other known patterns fail. All 40 of the authors’ pool of DDR4 DIMMs were shown to be vulnerable.

IV. Defenses

Recent defenses have approached the problem from a variety of perspectives. Researchers have gradually reduced the assumptions they make on specific adversary behaviors, and have increasingly focused on eliminating one or more conditions necessary in any successful attack. Randomized Row Swap (RRS) [17] seeks to isolate victim and aggressor rows spatially, reducing the impact of successful bit flips. Aqua [18] improves on this by reducing the number of swaps and creates a jail for aggressor rows, and using DRAM for tracking (unlike other mechanisms that use SRAM). BlockHammer [22] throttles memory accesses to rows that it detects are being activated often, precluding rows from being hammered fast enough in the available window. Finally, CSI:Rowhammer [8] eschews any attack-specific assumptions and uses cryptographic MACs to ensure integrity of data in DRAM to protect against all kinds of faults and errors, intentional or otherwise.

A. Randomized Row Swap

Unlike all other defenses that focus on victims, this defense targets adversaries. In this mitigation, aggressor rows with frequent memory accesses are identified by a modified implementation of the Misra-Gries frequent value analysis algorithm. Since there are fewer of them compared to victim rows, aggressor rows are tracked using a *hot row tracker*, leading to a comparatively lower overhead.

Once the access counts of these rows exceed a threshold, they are swapped with other random rows, breaking the spatial locality with victim rows. Rowhammer attack patterns can cause bit-flips in rows that are not adjacent, so, instead of using activations on specific rows (which require considerable resources to individually track), this defense foils the attacker’s attempt to focus bit flips on any given row. This greatly reduces the time an adversary has to trigger bit flips, protecting against not only known adversarial memory access patterns, but also those that are unknown and more complex.

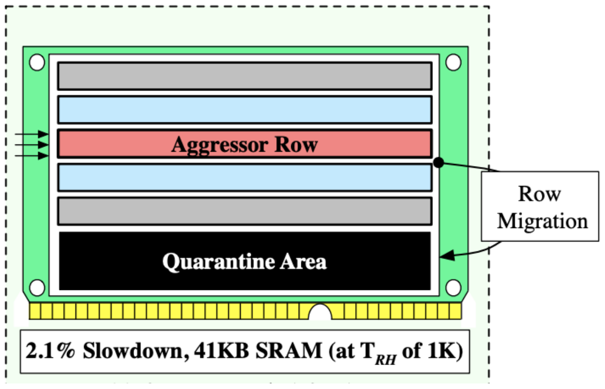

Fig. 5. Rows are migrated to the quarantine area in Aqua [18].

B. Aqua

Randomized Row-Swap (RRS) needs to swap an aggressor row with a random other row when the number of activations reaches a threshold approximately one-sixth the rowhammer threshold, which has a considerable performance overhead. By quarantining aggressor rows in a dedicated *row quarantine area* (akin to a jail), Aqua greatly reduces the number of swaps needed by increasing the threshold at which a swap is needed in the first place. This design leads to a 10x performance gain relative to RRS, and utilizes only 1% of DRAM to store aggressor row table, unlike RRS that needs 2.4MB of (more expensive) SRAM per rank at the lowest Rowhammer threshold of 1000 activations.

C. BlockHammer

BlockHammer seeks to provide a rowhammer defense that (1) scales with the continuously lowering Rowhammer activation threshold R_H and (2) does not require knowledge of proprietary knowledge of how vendors implement DRAM chips. The first component, RowBlocker (see 6 uses Bloom filters to track the activation rate of rows within a time interval. As can be seen in 6, a row is flagged if its activation count exceeds a threshold that is below R_H. Future memory accesses to flagged rows get proactively throttled to a rate below R_H until the end of the time interval, thus preventing flips from occurring.

Simultaneously, the other componentAttackThrottler, allocates a per-thread quota for memory accesses that is inversely correlated with the number of accesses to blacklisted rows. As a direct consequence, threads with benign applications enjoy higher memory bandwidth whereas adversaries are rate-limited, thus improving overall performance. All of this is implemented in the memory controller (a component of the CPU), which means that DRAM-specific information is not relied upon. By introducing and using a metric known as the *Rowhammer Likelihood Index*, the tool can reliably differentiate between Rowhammer attacks and other workloads.

D. CSI:Rowhammer

CSI: Rowhammer is a defense that does not rely on any specific properties of rowhammer attacks, and applies more broadly to all faults. It uses the QARMA tweakable block cipher [1] to compute a MAC of each word. When data is written to a cell, a MAC is calculated for the data word and stored in an adjacent location. This MAC is re-calculated on every access and compared (using approximate equality) to the stored MAC. If there is a difference of a single bit, a correction attempt is made in hardware. If there are more bits with flips, an Corrupted Memory exception is thrown which is handled in the operating system, resulting in the potential termination of the offending process (likely the rowhammer adversary).

To improve performance, the ISA is extended by instructions to calculate the MAC, and the software-side features a parity guided algorithm to efficiently search for the correct version of data (since MACs can only detect that an error occurred but not fix them or even identify the bits affected). The key used for the MAC is regenerated on every boot. This hardware-software co-design is evaluated on gem5 [2]'s OutOfOrder core and has an overhead lower than 0.75%. It can correct 5 bit flips in a 256-bit data word within 300 ms just using a brute-force search. This is is supplemented by optimizations, one of which is to reload pages that are backed by a disk, as opposed to attempting a search.

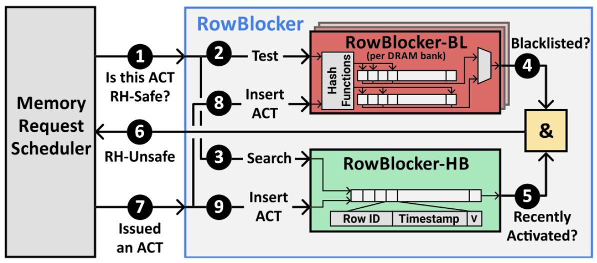

Fig. 6. The RowBlocker mechanism in BlockHammer arbitrating memory access requests. [22]

V. Discussion

The post-TRR era has seen adversaries come up with increasingly complex memory access patterns to trigger bit flips. Much work has gone into studying the characteristics of patterns that are more likely to produce flips, which have culminated in two fuzzing tools, TRRespass and Blacksmith. Meanwhile, defense researchers have had to return to the drawing board to glean what conditions are strictly necessary to pull off a successful attack and stem (as opposed to patch) issues from the root.

Hardware vendors have been caught relying on the obscurity of DRAM implementations, a strategy that needs changing. Just as developers working on proprietary software have embraced open-source and cryptographers rely on algorithm competitions, DRAM vendors should work to create publicly auditable designs which can let them provide concrete security guarantees. A good start could be to leverage attack tools to increase confidence in the implementations, akin to automated testing of software. At the same time, basic research into the physical causes of DRAM disturbance would also be beneficial. Insights gleaned from such research could enable designing DRAM that is constructed free from this and other known issues.

Prior research by Jose Rodrigo Sanchez Vicarte et al. [21] shows that there are many proposed hardware optimizations with security implications. If security were treated as a first-class design constraint just as power efficiency and performance, we as a community could break free from the cycle of feature development followed by security-related cleanup as an afterthought. As Winston Churchill, former Prime Minister of the United Kingdom said, "those who don't learn from history are doomed to repeat it". Lessons in life will be repeated until they are learned.

References

[1] Roberto Avanzi. "The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes". In: *IACR Transactions on Symmetric Cryptology* 2017.1 (Mar. 2017), pp. 4–44. DOI: 10.13154/tosc.v2017.i1.4-44. URL: https://tosc.iacr.org/index.php/ToSC/article/view/583.

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, et al. "The Gem5 Simulator". In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718.

[3] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D. Keromytis, Yossi Oren, et al. "HammerScope: Observing DRAM Power Consumption Using Rowhammer". In: CCS '22 (2022). DOI: 10.1145/3548606.3560688. URL: https://doi.org/10.1145/3548606.3560688.

[4] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, et al. "TRRespass: Exploiting the many sides of target row refresh". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 747–762.

[5] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.js: A remote software-induced fault attack in javascript". In: *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer. 2016, pp. 300–321.

[6] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. "SGX-Bomb: Locking down the processor via Rowhammer attack". In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 2017, pp. 1–6.

[7] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. "BLACKSMITH: Scalable Rowhammering in the Frequency Domain". In: *43rd IEEE Symposium on Security and Privacy (S&P)*. 2022.

[8] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Moritz Lipp, Maria Eichlseder, and Daniel Gruss. "CSI:Rowhammer - Cryptographic Security and Integrity against Rowhammer". In: *44th IEEE Symposium on Security and Privacy (S&P)*. 2023.

[9] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, et al. "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh". In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2022, pp. 1156–1169.

[10] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, et al. "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors". In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. ISCA '14. Minneapolis, Minnesota, USA: IEEE Press, 2014, pp. 361–372. ISBN: 9781479943944.

[11] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, et al. "Half-Double: Hammering From the Next Row Over". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3807–3824. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double.

[12] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. "RAMBleed: Reading Bits in Memory Without Accessing Them". In: *41st IEEE Symposium on Security and Privacy (S&P)*. 2020.

[13] Eojin Lee, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. "TWiCe: Preventing Row-hammering by Exploiting Time Window Counters". In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 2019, pp. 385–396.

[14] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, et al. "Nethammer: Inducing rowhammer faults through network requests". In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2020, pp. 710–719.

[15] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W. Lee. "Graphene: Strong yet Lightweight Row Hammer Protection". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020, pp. 1–13. DOI: 10.1109/MICRO50266.2020.00014.

[16] The Beard Sage. *DRAM nomenclature explained*. Feb. 2020. URL: http://thebeardsage.com/dram-nomenclature-explained/.

[17] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J. Nair. "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows". In: *Proceedings*

of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '22. Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 1056–1069. ISBN: 9781450392051. DOI: 10.1145/3503222.3507716. URL: https://doi.org/10.1145/3503222.3507716.

[18] Anish Saxena, Gururaj Saileshwar, Prashant Nair, and Moinuddin Qureshi. "AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime". In: *2022 ACM/IEEE 55th Annual International Symposium on Microarchitecture (MICRO)*. 2022.

[19] Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. "Counter-based tree structure for row hammering mitigation in DRAM". In: *IEEE Computer Architecture Letters* 16.1 (2016), pp. 18–21.

[20] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, et al. "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 1675–1689. ISBN: 9781450341394. DOI: 10.1145/2976749.2978406. URL: https://doi.org/10.1145/2976749.2978406.

[21] Jose Rodrigo Sanchez Vicarte, Pradyumna Shome, Nandeeka Nayak, Caroline Trippel, Adam Morrison, David Kohlbrenner, et al. "Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 347–360.

[22] A. Giray Yağlıkçi, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, et al. "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows". In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2021, pp. 345–358. DOI: 10.1109/HPCA51647.2021.00037.

[23] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. "DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1463–1480. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/yao.