The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses

Stefan Saroiu, Alec Wolman, Lucian Cojocar
Microsoft

***Abstract*—DRAM vendors today choose to keep the internal topologies of DRAM devices secret. This decision introduces significant practical challenges for designers of memory controllers who wish to provide their own forms of Rowhammer defenses. This paper describes three such challenges and the vulnerabilities, inefficiencies, and overhead they introduce. Defenses implemented at the memory controller could reduce (if not eliminate) these deficiencies should internal DRAM topology be made available.**

A high rate of accesses to the same set of DRAM rows can disturb nearby DRAM cells and affect their electrical charges. A malicious process can leverage this disturbance anomaly to induce bit flips in DRAM causing data corruption and security exploits, a phenomenon known as *Rowhammer attacks* [22], [29]–[31]. Unfortunately, with each shrinking of the DRAM fabrication process, the number of accesses until bits start to flip in DRAM is falling at a precipitous rate. Figure 1 shows the *hammer count* of DDR4 DRAM modules until the first bit flip occurs across three different lithographies: fabrication nodes 1x, 1y, and 1z. For each lithography, we randomly test three DIMMs, one from each of the three major DRAM vendors. These falling trends have been observed by others [21].

DRAM vendors have started to deploy internal-to-DRAM Rowhammer mitigations that track row accesses and remedy disturbance effects. While helpful, these mitigations have been shown to be incomplete – they can be bypassed by sophisticated attack patterns involving many rows co-located in the same bank [7], [8], [13], [14], [32]. Two recent whitepapers published by JEDEC, the industry consortium that develops DRAM standards, mention that in-DRAM mitigations cannot eliminate all forms of Rowhammer attacks [18], [19]. As a result, DRAM customers, such as cloud providers and SoC vendors, remain nervous about the possibility of unpleasant surprises – the discovery of new Rowhammer attack patterns that can bypass DRAM's internal mitigations. This concern is warranted because, in the past, DRAM vendors made claims about the security and safety of their DRAM chips that were later shown to be incorrect [11], [25].

A more reassuring alternative for DRAM customers is to implement their own Rowhammer defenses. For example, the research community has proposed many schemes that track row activations [1]–[4], [10], [12], [20], [23], [24], [26]–[29], [33]–[36], [38]–[41]. These approaches can be implemented external to the DRAM – in a memory controller, in a Register Clock Driver (RCD) chip [15] often found on server-grade

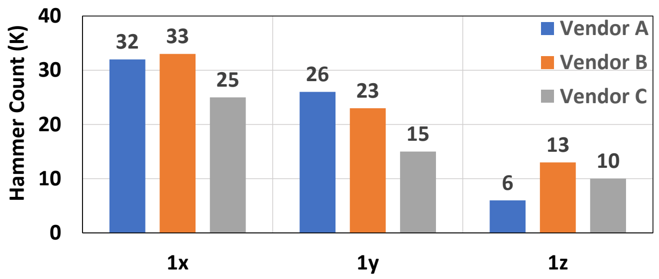

Fig. 1: Hammer count until the first bit flips (with in-DRAM internal mitigations disabled). This data comes from testing nine DDR4 DIMMs sourced from the three major DRAM vendors corresponding to three fabrication nodes: 1x, 1y, and 1z. For each DIMM we tested 2000 rows. Each hammer test used two aggressor rows (double-sided). The total number of row activations is twice the hammer count.

DIMMs, or in software. When a row has too many activations, it is deemed an *aggressor row*, and most of these schemes refresh nearby DRAM rows called *victim rows*. Refreshing potential victim rows before any of their bits flip undoes the disturbance effects. Thus, data stored in the victim rows remains protected. For the reminder of this paper, we will assume the memory controller is the place where external Rowhammer mitigation is taking place for simplicity.

Unfortunately, such schemes cannot be yet deployed in practice due to DRAM vendors' insistence on keeping their internal DRAM topologies secret. The absence of this topology information makes it impossible for a memory controller to identify and refresh the victim rows affected by a tracked aggressor row. As an alternative, researchers have proposed the addition of a new DRAM command called Nearby Row Refresh (NRR) [24], [33]. When it detects an aggressor row, the memory controller issues NRR to report the aggressor's row address to the DRAM device which then allows the device to refresh the relevant victim rows. NRR appears to be an ideal compromise solution: DRAM vendors can continue to keep their internal DRAM topologies secret, while customers can implement DRAM protections in their memory controllers by leveraging the new NRR command.

In this paper, we argue that keeping internal DRAM topologies secret hurts DRAM customers in several ways. We start by showing that NRR is far from an ideal solution and, in fact, can itself be weaponized and turned into a Rowhammer attack vector. Second, in the absence of DRAM topology information, Rowhammer defenses must choose conservative values for their parameters (e.g., Rowhammer thresholds)

Aggressor row	A row activated repeatedly with the goal of inducing bit flips on adjacent (or nearby) rows in a bank. In some older terminology, aggressor row is sometime called target row.
Victim row	A row exhibiting bit flips that is located nearby an aggressor row.
Single-sided attack	An attack where a row is activated repeatedly with the goal of inducing bit flips on adjacent (or nearby) rows in a bank.
Double-sided attack	An attack with two aggressor rows located one row apart. The row located between the aggressors is a victim row.
Hammer count	The minimum number of times to activate each set of aggressor rows until the first bit flip occurs.
Blast radius	The physical distance (i.e., the number of rows apart) between an aggressor and a victim row. A blast radius of 1 corresponds to the case when the aggressor and victim rows are adjacent. Distant rows correspond to a blast radius greater than 1.
Rowhammer threshold	The maximum number of activations a row can sustain until a Rowhammer mitigation action must be performed.
Attenuation factor	A factor representing the reduction of disturbance errors as the blast radius increases. This factor is assumed to directly correlate with the increase in the number of activations an aggressor row requires to flip bits in victim rows located farther away. For example, an attenuation factor of 10 means that a victim row requires 10 times more activates to flip bits in a victim row located two rows away than an adjacent victim.

TABLE I: Rowhammer terminology used in the literature.

incurring significant overheads. Third, the NRR command must also behave conservatively and refresh *all* victim rows potentially affected by an aggressor row. The lack of security, the inefficiencies, and the overheads could be reduced (if not eliminated) should internal DRAM topologies be made available.

The remainder of this paper describes each of these three shortcomings in depth. In each case, we quantify their overheads using DRAM configurations with different *hammer counts* and *blast radii*. The hammer count value represents the minimum number of accesses to an aggressor row until the first bit flips on a victim row. DRAM with newer lithographies has lower hammer counts than DRAM with older ones (Figure 1). The blast radius of a Rowhammer attack has a value of one when an aggressor row affects only two victim rows (the adjacent ones). The blast radius value is two if an aggressor row affects four victims, and so on. Table I lists a brief primer of Rowhammer terminology used in the literature.

1. NRR-based transitive attacks. A large-class of proposals for Rowhammer defenses rely on *tracking* DRAM row activations. When the number of activations of a DRAM row exceeds a threshold within a fixed time interval, the defense issues an NRR command to the DRAM device. NRR reports the address of the highly activated row (known as the aggressor

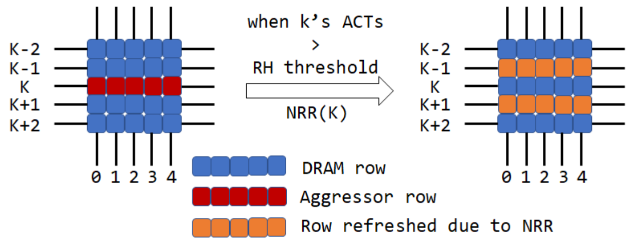

Fig. 2: NRR command is issued when row K exceeds the Rowhammer threshold. As a result, relevant victim rows (in our example rows $K-1$ and $K+1$ are refreshed.

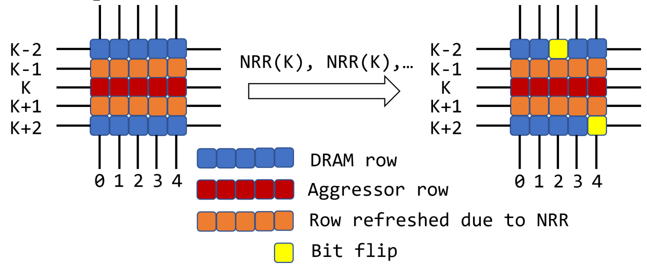

Fig. 3: NRR command is issued when row K exceeds the Rowhammer threshold. As a result, relevant victim rows (in our example rows $K-1$ and $K+1$ are refreshed.

row), and the DRAM device refreshes *all* corresponding victim rows. Figure 2 illustrates an example in which the aggressor row K exceeds a pre-defined Rowhammer threshold. In this case, the memory controller issues NRR(K) and the DRAM devices refreshes adjacent rows $K-1$ and $K+1$.

The row addresses shown in Figure 2 (and all figures that follow) are internal topological DRAM addresses. The memory controller instead uses DDR bus-based row addresses without knowing which addresses are adjacent or nearby. Two consecutive DDR bus-based row addresses, say $A1$ and $A2$ are not guaranteed to be adjacent; $A1$ might map to K and $A2$ might map to $K+8$. Even worse, a single DDR bus-based row address could have different portions map to *different* internal DRAM rows [5], [21].

NRR-based refreshes behave like row activations and, unfortunately, these additional activations cannot be tracked by the memory controller. The Rowhammer defenses do not know the addresses of the rows refreshed by NRR because this topology information is kept internal to the DRAM device. This creates the potential for a *transitive* (or indirect) form of hammering caused by the NRR commands. Figure 3 shows a DRAM device that receives a high rate of NRR commands due to row K reaching its threshold repeatedly. These NRR commands repeatedly refresh rows $K-1$ and $K+1$, thereby *transitively* turning them into aggressor rows. These transitive aggressor rows are now hammering rows $K-2$ and $K+2$ freely and can flip bits on them without the memory controller being able to count those activations.

The number of untracked row activations due to NRR commands can be quite high for DRAM configurations with low hammer counts. Table II shows the maximum number of untracked row activations within a refresh window due to NRR for different hammer counts and blast radii.

Hammer Count (K)	Max # of untracked ACTs (blast radius = 1)	(blast radius = 2)
30	166	182
20	238	272
10	498	548
5	996	1096
3	1660	1828
1	4980	5484

TABLE II: Maximum number of untracked ACTs for different DDR5 DRAM configurations. We assumed a single-sided attack and the following DDR5 timings [17]: tREF = 32 ms, tRC = 46 ns, tRFC = 410 ns. To compute the Rowhammer threshold, we used the formula from Graphene [33] with an attenuation factor of 10.

The root of this problem stems from the secrecy of the internal DRAM topology. By issuing an NRR command, the memory controller issues row activations (i.e., refreshes) without being able to track the addresses of the rows being activated. Indirectly, NRR lets an attacker issue row activations without being tracked. The Rowhammer defense is now left in a significantly weaker position and cannot offer strong protection guarantees.

Instead, access to the internal DRAM topology information would eliminate all such forms of transitive attacks. Equipped with topology information, a memory controller could track all row activations and provide strong protection guarantees against multi-row forms of attacks [7], [8], [13], [14], [32]. The NRR command would be no longer needed. When the number of row activations exceeds a threshold, the memory controller simply uses regular row activations to refresh (and remedy) the relevant victim rows. The Rowhammer defense tracks these additional indirect row activations. Finally, as an added benefit, the DRAM device would be simpler because it does not need the extra circuitry logic to implement and test the NRR command.

2. Defenses must set their Rowhammer thresholds conservatively. The most basic form of Rowhammer attack is *single-sided*: when an attacker repeatedly activates a single aggressor row. When the number of row activations exceeds a threshold T, cells located in victim rows are disturbed and may lead to bit flips. A slightly more sophisticated attack is *double-sided*: when an attacker activates two aggressor rows located one row apart. The "sandwiched" victim row is now disturbed by both aggressors. When each of the aggressor rows are activated $\frac{T}{2}$ times, they induce a similar degree of disturbance to the victim row as a single-sided attack with a threshold T. Figure 4 illustrates single-sided and double-sided attacks. Even more sophisticated forms of Rowhammer attacks involving multiple aggressor rows are possible [7], [8], [13], [14], [32].

The lack of internal DRAM topology leaves the memory controller unable to distinguish between these forms of attacks. The memory controller cannot determine whether two aggressor rows are one-row apart or many-rows apart. Figure 5 shows two different scenarios each involving two aggressor rows. In one case, the attack is double-sided, whereas in the other case, there are two ongoing single-sided attacks. The memory controller cannot distinguish between these two scenarios without knowledge of DRAM internal topology.

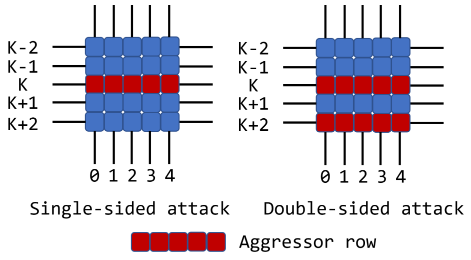

Fig. 4: Single-sided and double-sided Rowhammer attacks. On the right, row $K+1$ is "sandwiched" between the aggressor rows.

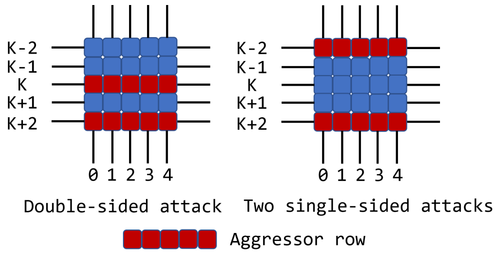

Fig. 5: A double-sided attack and two single-sided attacks. The memory controller cannot distinguish between the two.

Tracking Rowhammer defenses must set their thresholds to very conservative values to handle all possible attack patterns. For example, to handle double-sided attacks the Rowhammer threshold must be set to $\frac{T}{2}$. These conservative settings are wasteful and incur overheads. Our example lowers the Rowhammer threshold by a factor of 2, a significant reduction. By lowering the Rowhammer threshold, the memory controller will perform remedies much more aggressively even when the DRAM is in no danger of bit flips. Normal, non-adversarial workloads will be increasingly likely to trigger the Rowhammer defenses due to their conservative thresholds.

The conservative setting of thresholds will lead to redundant refreshes of row victims: the same row victim is repeatedly refreshed due to different aggressor rows reaching their thresholds. Table III shows the maximum number of redundant row refreshes for a double-sided Rowhammer attack.

Access to the DRAM topology information would enable the memory controller to more efficiently track multi-row forms of Rowhammer attacks. Rowhammer defenses could repurpose their counters to track victims instead of aggressors, and the counter values would represent the degree of potential row disturbance to victim rows. When the row disturbance value reaches a threshold, the memory controller refreshes that row. Such an approach would no longer need to set the threshold conservatively. Instead, the counters would be increased proportionally to the disturbance induced by each aggressor row. In Figure 4, row $K+1$ suffers the same degree of disturbance in both single-sided and double-sided attacks. In contrast, in Figure 5, row $K+1$'s disturbance counter increases more rapidly in the double-sided attack than in case of two single-sided attacks.

Hammer Count (K)	Max # of redundant row refreshes
30	83
20	124
10	249
5	498
3	840
1	2490

TABLE III: Maximum number of redundant row refreshes for different DDR5 DRAM configurations. We assumed a double-sided attack and the following DDR5 timings [17]: tREF = 32 ms, tRC = 46 ns, tRFC = 410 ns. To compute the Rowhammer threshold, we used the formula from Graphene [33] with an attenuation factor of 10.

Hammer Count (K)	Max # of unnecessary distant row refreshes
30	149
20	223
10	448
5	896
3	1494
1	4482

TABLE IV: Maximum number of unnecessary distant row refreshes for different DDR5 DRAM configurations. We assumed a single-sided attack and the following DDR5 timings [17]: tREF = 32 ms, tRC = 46 ns, tRFC = 410 ns. To compute the Rowhammer threshold, we used the formula from Graphene [33] with an attenuation factor of 10.

3. Many row refreshes due to NRR are wasteful. Any "optimizations" to NRR are likely to introduce vulnerabilities. Upon receiving an NRR command, the DRAM device must refresh all potential victim rows. Unfortunately, in many cases, some of these refreshes are unnecessary (or redundant). As an example, consider a double-sided attack, such as the one shown on the left in Figure 5. This attack causes the memory controller to issue two NRR commands reporting aggressor rows K and $K+2$. The DRAM device will refresh $K-1$, $K+1$ and $K+1$, $K+3$ (not shown in the Figure), respectively. This examples illustrates the inefficiency – row $k+1$ ends up being refreshed twice redundantly.

Additional inefficiencies arise in DRAM devices in which an aggressor row disturbs several victims, not just two. In practice, disturbance has been shown to affect multiple rows located nearby an aggressor row, although the degree of disturbance decreases with distance [5], [22]. The two rows adjacent to an aggressor are disturbed the most (say their disturbance is d), then the two rows located one-row apart suffer less disturbance (say $\mu \times d$) than the two rows located two-rows apart ($\mu \times \mu \times d$), and so on [33].

Figure 6 illustrates a single-sided attack in a DRAM device with a blast radius of 2. For such a device, the NRR command will refresh four rows. This is inefficient because the distant rows (i.e., located farther away from the aggressor) $K-2$, $K+2$ do not need to be refreshed as often as the adjacent rows $K-1$, $K+1$. Table IV shows the maximum number of additional refreshes of distant rows that are unnecessary for different DRAM configurations.

One could imagine an "optimized" NRR command that

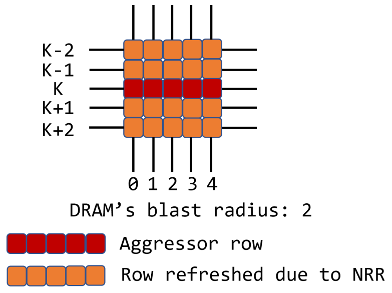

Fig. 6: Single-sided attack in a DRAM device with a blast radius of 2.

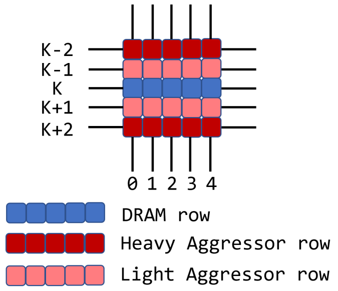

Fig. 7: Multi-sided attack involving heavy and light aggressor rows in a DRAM device with a blast radius of 2.

refreshes victim rows at different rates depending on their distance from the aggressor row. For example, in Figure 6, NRR could refresh the adjacent rows but only a random sample of the distant rows. Such a behavior would refresh the distant rows less aggressively than the adjacent ones.

Unfortunately, such changes are dangerous because they introduce vulnerabilities in the Rowhammer defenses. As an example, consider a new multi-row attack involving *heavy* and *light* aggressor rows. A heavy aggressor is one that reaches the threshold repeatedly causing the memory controller to issue NRR commands. A light aggressor receives enough row activations to almost, but not quite, reach the threshold.

Figure 7 illustrates such an attack that leaves row K unprotected. The light aggressor rows mount a double-sided attack. However, the light aggressor rows do not trigger NRR commands because they stay just below the Rowhammer threshold. Instead, the memory controller issues NRR commands for the heavy aggressor rows $K-2, K+2$. However, the DRAM device might end up not sampling row K frequently enough to undo its disturbance.

Implementation Challenges

While providing DRAM topology information to the memory controller offers many benefits as described above, it could also add significant implementation complexity to the memory controller.

One challenge stems from the diversity of memory types and DRAM topologies. Memory controllers must be able to discover the internal topology of each DRAM device and represent it in a compact way. The discovery could be done by querying the Serial Presence Detect (SPD) [16] chip. While there could be many different internal topologies, we expect each topology to follow a regular pattern. Memory controllers could encode and represent each pattern efficiently and in a compact manner.

Another challenge is row remapping. DRAM manufacturers sometimes internally remap certain rows to other locations as a work-around for faults in the manufacturing process. Also, certain DRAM devices support post-package repair [9] that remaps rows in the field to available spare rows. These remappings are a form of *exceptions* to the topology's regular pattern and representing their new locations in a compact manner could be challenging. This is a case where we feel DRAM vendors are in a position to help by ensuring the area where rows are remapped is resilient to DRAM disturbance. For example, they can place remapped rows sufficiently apart (leaving empty rows acting as in-between guards). Another possibility is to re-purpose internal-to-DRAM Rowhammer mitigations to protect remapped rows only, a much easier task than offering whole DRAM device protection.

Towards a Rowhammer-free Future

DRAM vendors have long kept details about DRAM's internal topology secret. We argue the time to reconsider this decision has arrived. Memory controllers could provide stronger and more efficient Rowhammer defenses than what is possible today. Rowhammer defenses could be more targeted and consume less DRAM bandwidth and power. Memory controllers could refresh victim rows using traditional row activation commands without the need for a new DRAM command, saving on implementation and testing costs.

Finally, DRAM vendors' need for secrecy originates from two causes. First, there is a belief that revealing internal topology could disclose IP information to their competitors. Second, vendors are concerned that DRAM customers would use these internal details to evaluate devices arguing that DRAM devices from vendor A are "better" than those from vendor B.

Neither reason stands scrutiny. A DRAM tester [37], a device with a price of several thousands of dollars, can reverse engineer DRAM internal topology. Additionally, our previous work describes building a low-budget fault injector that can perform similar reverse engineering experiments [6]. DRAM vendors evidently have the budgets and the know-how to reverse engineer each others' parts and learn about their internal IP secrets.

Finally, DRAM customers will be unlikely to use internal DRAM topology information to decide which vendor's devices are best. Some DRAM devices already report timings and other information through their Serial Presence Detect (SPD) [16] chips. Yet, there is no evidence that SPD-based information dictates DRAM customer's purchases. Arguably, a DRAM vendor who shares internal DRAM topology in an effort to provide increased security and reliability could appear more appealing to customers than one who decides to keep this information secret.

Acknowledgments. We would like to thank Flavio Griggio, Yan Liu, Kevin Loughlin, and Jay Sarkar for their feedback on previous drafts of our paper.

References

[1] K. S. Bains and J. B. Halbert, "Distributed Row Hammer Tracking," Patent App. US 13/631,781, 2014.

[2] K. S. Bains and J. B. Halbert, "Row Hammer Monitoring Based on Stored Row Hammer Threshold Value," Patent No. US 2015/9,032,141, 2015.

[3] K. S. Bains, J. B. Halbert, C. P. Mozak, T. Z. Schoenborn, and Z. Greenfield, "Row Hammer Refresh Command," Patent No. US 2014/0006703 A1, 2014.

[4] A. Chakraborty, M. Alam, and D. Mukhopadhyay, "Deep Learning based Diagnostics for Rowhammer Protection of DRAM Chips," in *ATS*, 2019.

[5] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *IEEE S&P*, 2020.

[6] L. Cojocar, K. Loughlin, S. Saroiu, B. Kasikci, and A. Wolman, "mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors," *Technical Report – Microsoft Research*, vol. MSR-TR-2021-25, 2021.

[7] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript," in *USENIX Security*, 2021.

[8] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.

[9] FuturePlus Systems, "JEDEC DDR4 Revision B Spec: What's different?" https://www.futureplus.com/jedec-ddr4-revision-b-spec-whats-different/, 2017.

[10] M. Ghasempour, M. Lujan, and J. Garside, "ARMOR: A Run-Time Memory Hot-Row Detector," http://apt.cs.manchester.ac.uk/projects/ARMOR/RowHammer/armor.html, 2015.

[11] M. Greenberg, "Row Hammering: What it is, and how hackers could use it to gain access to your system," https://blogs.synopsys.com/committedtomemory/2015/03/09/row-hammering-what-it-is-and-how-hackers-could-use-it-to-gain-access-to-your-system/, 2015.

[12] Z. Greenfield, J. B. Halbert, and K. S. Bains, "Method, Apparatus and System for Determining a Count of Accesses to a Row of Memory," Patent App. US 13/626,479, 2014.

[13] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021.

[14] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *IEEE S&P*, 2022.

[15] JEDEC, *DDR4 Register Clock Driver Definition (DDR4RCD02)*, 2019.

[16] JEDEC, *Serial Presence Detect (SPD), General Standard No. 21C*, 2019.

[17] JEDEC, *Double Data Rate 5 (DDR5) SDRAM Standard*, 2020.

[18] JEDEC, *Near-Term DRAM Level Rowhammer Mitigation (JEP300-1)*, 2021.

[19] JEDEC, *System Level Rowhammer Mitigation (JEP301-1)*, 2021.

[20] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *CAL*, 2015.

[21] J. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.

[22] 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*, 2014.

[23] E. Lee, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Time Window Counter Based Row Refresh to Prevent Row-Hammering," *CAL*, 2018.

[24] E. Lee, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Preventing Row-hammering by Exploiting Time Window Counters," in *ISCA*, 2019.

[25] J.-B. Lee, "Green Memory Solution," http://aod.teletogether.com/sec/20140519/SAMSUNG_Investors_Forum_2014_session_1.pdf, 2014.

[26] C. Li and J.-L. Gaudiot, "Detecting Malicious Attacks Exploiting Hardware Vulnerabilities Using Performance Counters," in *COMPSAC*, 2019.

[27] J. Lin and M. Garrett, "Handling Maximum Activation Count Limit and Target Row Refresh in DDR4 SDRAM," Patent No. US 2015/0200002 A1, 2015.

[28] S. Mandava, B. S. Morris, S. Sah, R. M. Stevens, T. Rossin, M. W. Stefaniw, and J. H. Crawford, “Techniques for Determining Victim Row Addresses in a Volatile Memory,” Patent No. US 9,269,436 B2, 2016.

[29] O. Mutlu, “The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser,” in *DATE*, 2017.

[30] O. Mutlu, “RowHammer and Beyond,” in *COSADE*, 2019.

[31] O. Mutlu and J. S. Kim, “RowHammer: A Retrospective,” *TCAD*, vol. Special Issue on Top Picks in Hardware and Embedded Security, 2019.

[32] L. Orosa, A. G. Yaglikci, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, “A Deeper Look into RowHammer’s Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses,” in *MICRO*, 2021.

[33] Y. Park, W. Kwon, E. Lee, T. J. Han, J. H. Ahn, and J. W. Lee, “Graphene: Strong yet Lightweight Row Hammer Protection,” in *MICRO*, 2020.

[34] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, “Counter-based Tree Structure for Row Hammering Mitigation in DRAM,” *CAL*, 2017.

[35] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, “Mitigating Wordline Crosstalk Using Adaptive Trees of Counters,” in *ISCA*, 2018.

[36] M. Son, H. Park, J. Ahn, and S. Yoo, “Making DRAM Stronger Against Row Hammering,” in *DAC*, 2017.

[37] TurboCATS, “The TurboCATS III-2667ST Memory Test System,” http://www.turbocats.com.hk/, 2019.

[38] Y. Wang, Y. Liu, P. Wu, and Z. Zhang, “Detect DRAM Disturbance Error by Using Disturbance Bin Counters,” in *CAL*, 2019.

[39] Y. Wang, Y. Liu, P. Wu, and Z. Zhang, “Reinforce Memory Error Protection by Breaking DRAM Disturbance Correlation Within ECC Words,” in *ICCD*, 2019.

[40] A. G. Yağlıkçı, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, “BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows,” in *HPCA*, 2021.

[41] J. M. You and J.-S. Yang, “MRLoc: Mitigating Row-hammering based on memory Locality,” in *DAC*, 2019.