Sudoku: Decomposing DRAM Address Mapping into Component Functions

Minbok Wi† Seungmin Baek† Seonyong Park† Mattan Erez‡ Jung Ho Ahn†

Seoul National University† *The University of Texas at Austin*‡

Decomposing DRAM address mappings into component-level functions is critical for understanding memory behavior and enabling precise RowHammer attacks, yet existing reverse-engineering methods fall short. We introduce novel timing-based techniques leveraging DRAM refresh intervals and consecutive access latencies to infer component-specific functions. Based on this, we present Sudoku, the first software-based tool to automatically decompose full DRAM address mappings into channel, rank, bank group, and bank functions while identifying row and column bits. We validate Sudoku's effectiveness, successfully decomposing mappings on recent Intel and AMD processors.

1. Introduction

DRAM address mapping is a key feature of memory controllers (MCs), which translates memory requests into physical locations in DRAM to maximize parallelism and minimize contention in the memory system. Simultaneously, the growing threat of DRAM-related attacks increases the importance of precise and detailed knowledge of the DRAM address mapping [1, 5, 8, 9, 12, 18, 19]. However, despite its importance for both performance and security, DRAM address mappings remain undocumented by processor manufacturers, necessitating efficient reverse-engineering methods.

Prior reverse-engineering methods [9, 16, 18, 22] primarily use the well-known row-buffer conflict timing channel to recover bank addressing functions; however, they require physical probing or fail to fully identify all row and column bits. Furthermore, prior methods are unable to decompose DRAM address mapping into component functions,[1] which becomes more important considering recent RowHammer attacks that exploit specific DRAM internal components [1, 12, 15, 17].

In this paper, we revisit timing channels in DRAM-based memory systems to enable component-level decomposition of DRAM address mapping. First, we use DRAM refresh intervals as an indicator for inferring the granularity of refresh operations, known as refresh groups. We can determine whether two addresses are mapped to the same refresh group by alternating two memory addresses, detecting refresh-induced latency spikes, and measuring their intervals. Second, we analyze the latency of consecutive memory accesses and utilize this to infer the functions of the DRAM component. Memory access patterns affect the latency of consecutive memory accesses, depending on the target DRAM components, and provide information for identifying bank group and address functions. Lastly, we briefly cover the non-uniqueness of the derived system of the DRAM addressing functions.

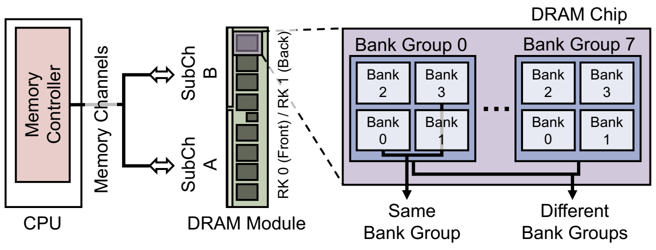

Figure 1: DRAM-based memory system organization. DDR5 introduces sub-channels and doubles the number of bank groups compared to DDR4.

Based on our exploration, we develop Sudoku, a software-based tool that identifies row and column bits and decomposes DRAM address mappings into component functions, even without physical access to systems. Sudoku successfully finds row and column bits and decomposes DRAM address mapping into component functions across recent Intel and AMD processors under various memory configurations. We also verify our results based on the processor's MC-related registers and recent physical probing-based results of DRAM address mappings [7, 9]. Lastly, we open-source our code to aid future research on reverse-engineering the DRAM address mappings.[2]

This paper makes the following key contributions:

- We analyze timing channels in modern DRAM-based memory systems based on refresh intervals and consecutive memory accesses.
- We develop Sudoku, a software-based tool that discovers full DRAM address mappings and decomposes these mappings into DRAM internal component functions, based on our timing channel analyses.
- We identify row and column bits without requiring physical access to the system, and we verify the results by confirming the injectivity of the derived mapping.
- We find DRAM component functions in recent Intel and AMD processors with various memory configurations.

2. Background

2.1. DRAM Organization, Operations, and Timings

DRAM-based main memory systems employ a hierarchical structure—typically comprising multiple *components*, such as channels, DIMMs, ranks, bank groups, and banks—to achieve high parallelism (see Figure 1). Modern standards such as DDR5 [11] further increase complexity by introducing sub-channels and doubling the number of bank groups compared to DDR4 [10]. Accessing data requires a Memory Controller

[1]We denote the memory system elements—such as the channel, DIMM, and rank—and the logical elements of a DRAM chip—such as the bank group and bank—collectively as *components*. We use "components" and "internal components" interchangeably.

[2]https://github.com/scale-snu/sudoku.git

(MC) to issue commands following specific timing constraints. An `ACT` (activate) command opens a target row within a bank. After a delay (`tRCD`), `RD` (read) or `WR` (write) commands access specific columns in the open, activated row. Accessing a different row within the same bank necessitates a `PRE` (precharge) command and incurs an additional delay (`tRP`) prior to the next activation. This sequence results in higher latency for same-bank accesses compared to row hits, creating the well-known row-buffer conflict timing channel.

Beyond row-buffer conflicts, other DRAM operations create observable timing variations [5]. DRAM cells require periodic refresh operations (`REF`) to retain data, which temporarily block memory accesses for `tRFC` cycles and occur at an average interval of `tREFI`. Modern DRAM devices support various types of refresh operations, such as all-bank, fine-grained, or same-bank refreshes, to reduce performance overhead from refresh operations [10, 11]. MCs may implement those various refresh schemes, potentially affecting access latency differently depending on the target addresses.

Crucially for our work, DRAM employs mandatory timing parameters for consecutive accesses, such as `tRRD` (row-to-row delay) and `tCCD` (column-to-column delay), which vary depending on whether the accesses target the same or different bank groups [10, 11]. However, rather than directly using these DRAM timing parameters, MCs use their own set of timing parameters—such as `tRDRD`, `tRDWR`, `tWRRD`, and `tWRWR`—based on the sequence and type of memory accesses [2–4]. These timing parameters are more complex, as they depend not only on the bank group and bank, but also on the rank and DIMM. Moreover, they are non-SPD related timings whose values vary based on the system platform and configuration, and are typically determined during DDR training for optimal performance. While often small, these timing differences between consecutive accesses represent another potential channel for inferring address mapping details.

2.2. XOR-Based Hash Functions

Modern MCs typically use XOR-based hash functions to translate physical addresses into DRAM component indices (*e.g.*, channel, rank, bank group, bank, row, and column) [6, 20, 21]. Each XOR-based hash function generates a single-bit hash value by XORing a selected subset of physical address bits. Thus, each function can be represented as a bitmask, where each set bit indicates that the corresponding address bit is involved in the hash function [20]. The complete mapping can be represented as a system of linear equations over $GF(2)$, often implemented as a binary matrix, designed to distribute memory accesses evenly [6, 20, 21].

Reverse-engineering these undocumented functions relies on observing system behavior. Prior methods employ brute-force [9, 16, 18] or educated-guessing [6, 21, 22] techniques, primarily analyzing row-buffer conflicts to deduce relationships between physical address bits and bank/row mapping. They generate random address pairs, observe conflicts (or lack thereof), and attempt to solve the underlying linear system with given input-output pairs.

While useful for identifying bank and some row/column bits, these conflict-based approaches generally cannot decompose the mapping into functions corresponding to memory systems' architectural components (*e.g.*, channel, rank, and bank group). Determining this full component-level decomposition is essential for precisely modeling memory behavior and enabling advanced security analyses, motivating the exploration of additional timing channels in this paper.

3. Timing Channels for Component Function Identification

In this section, we analyze DRAM timing channels beyond simple row conflicts, focusing on refresh operations and consecutive accesses to infer component-level mapping details.

3.1. Understanding How Systems Configure Memory via System Registers

Accurate analysis requires understanding system-specific memory configurations and timings. We identify these by examining BIOS settings and reading processor-specific MC-related registers [2–4]. For example, we obtain precise values for key timing parameters such as `tRFC`, `tREFI`, and various `tRDRD` timings, which are essential for our subsequent analyses. We also examine the MC-related registers that are no longer documented in recent processors but were disclosed in the previous datasheets, as in [13], to obtain several configurations related to DRAM address mapping [2, 3].

Knowing the system-configured refresh interval (`tREFI`) is particularly important, as it depends on the type of refresh operations and DRAM chip density [10, 11]. We also note that on the tested Intel processors with DDR5 (Table 1), the MC treats DDR5 sub-channels as two independent memory channels; a single-rank DDR5 module is perceived as two channels, and populating two physical channels results in the MC managing four logical channels. This understanding informs our timing analyses on systems with DDR5.

Modern processors also provide limited information about DRAM address mapping through MC-related registers [2, 3], enabling verification of specific DRAM component mapping functions. For example, Intel Core processors reveal a channel hash mask and the bit used for bank group selection through their MC-related registers [2, 3]. Also, in Linux, the EDAC (Error Detection and Correction) subsystem utilizes these MC-related registers to accurately diagnose the DRAM error locations, thereby revealing portions of DRAM address mappings in the server processors [14, 15]. We leverage this DRAM address mapping information to validate Sudoku's results (Section 5).

Finally, while accessing system registers aids our timing channel analysis, our Sudoku tool does not require it.

3.2. Refresh Interval

Beyond simple detection, we exploit refresh-induced latency spikes to infer refresh group—whether two addresses share the same refresh resources. By alternately accessing a pair of randomly generated memory addresses and monitoring the interval between periodic latency spikes [5, 9], we observe distinct

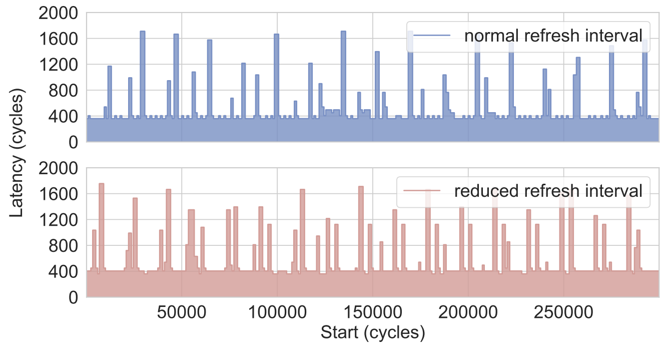

Figure 2: Measured refresh-induced latency spikes in the AMD Ryzen 9 7950X processor with DDR5 operating at 4.5GHz. We iteratively access two memory addresses, measure their latencies, and compute the average spike interval. Two types of refresh intervals are shown: a normal refresh interval (top) and a reduced refresh interval (bottom).

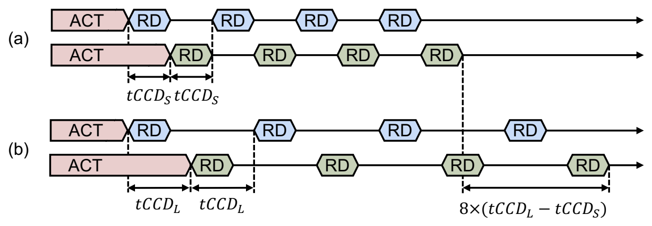

Figure 3: Theoretical timing differences between two consecutive memory reads. Consecutive reads exhibit different timing characteristics depending on their mappings. (a) Reads to different bank groups ($tCCD_S$) incur a shorter interval than (b) reads to the same bank group ($tCCD_L$).

patterns. Addresses mapped to the same refresh group yield spikes at the standard tREFI interval. By contrast, addresses mapped to different refresh groups produce two interleaved spike trains, resulting in a reduced effective interval between observed latency spikes. Figure 2 shows measured memory access latencies over time in the AMD Ryzen 9 7950X processor with DDR5. For the normal refresh interval, we observe monolithic and periodic latency spikes. In contrast, for the reduced refresh interval, there are two distinct periodic spikes with a certain offset. This directly reveals whether the address pair differs in the address bits determining the refresh group.

Applying this method, we observed all-bank refresh on an Intel 12th Gen processor with DDR4, fine-grained refresh on Intel 12th (Alder Lake) and 14th (Raptor Lake Refresh) Gen processors with DDR5, and all-bank refresh operations on an AMD Ryzen Zen 4 processor with DDR5.[3] Furthermore, since XOR-based hash functions tend to partition the address space evenly, the proportion of random address pairs exhibiting reduced intervals provides an estimate of the number of address bits (and thus, XOR functions) involved in determining the refresh group.

3.3. Consecutive Memory Accesses

MC's memory timing parameters, such as the variants of tRDRD, specify different minimum intervals for consecutive memory accesses depending on whether they target the same bank group, different bank groups, different ranks, or different DIMMs in both Intel and AMD processors [2–4]. While these timing differences encode component mapping information, they are typically small (a few nanoseconds) and easily obscured by MC request scheduling and system noise.

[3]Measurements on AMD systems required careful thresholding due to limited granularity of measuring timings and various noise factors.

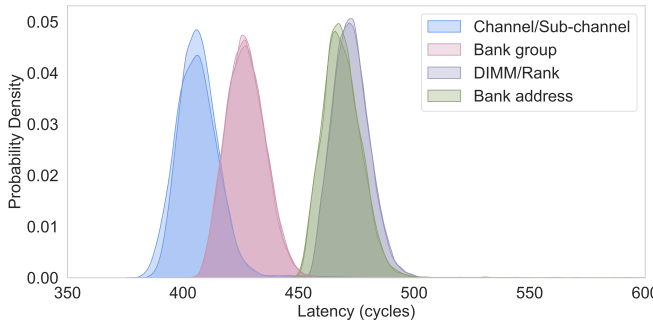

Figure 4: The distribution of consecutive memory access latencies across memory address pairs that differ in only one of the nine DRAM addressing functions.

To overcome this limitation, we amplify these subtle differences. Our method creates and alternately accesses two *memory streams*, each carefully constructed to ensure internal row-buffer hits. This interleaving magnifies the base timing differences dictated by the relevant tRDRD variant between the two streams, as illustrated conceptually in Figure 3.

Measuring the latency of these alternating stream accesses reveals distinct latency distributions (shown for an Intel Core i9-14900K in Figure 4). The peaks of these distributions correspond to the timing differences associated with accessing different component levels. By comparing the measured peak latencies to the system-configured tRDRD values (obtained as per Section 3.1), we can group the underlying XOR functions based on the component they likely map to (*e.g.*, channel, sub-channel, bank group, DIMM/rank, and bank address). Combined, refresh interval analysis and consecutive access timing allow us to group address mapping functions by component type. However, the current method has limitations; system-configured MC timing parameters highly affect the latency of consecutive memory accesses. For example, distinguishing rank and module functions could be ambiguous if their respective tRDRD timings are configured identically by the system.

4. Sudoku

We develop Sudoku,[4] a software tool for reverse-engineering DRAM address mapping in commercial computer systems, requiring no physical access. Sudoku extends existing tools (*e.g.*, DRAMA [18] and ZenHammer [9]) that reverse-engineer DRAM addressing functions using row-buffer conflicts. Specifically, Sudoku takes the DRAM addressing functions produced by these tools, automatically identifies row and column bits, and decomposes DRAM address mappings into component functions. As depicted in Figure 5, Sudoku integrates conflict-based testing [18] with timing channel analyses from Section 3.

[4]The tool is named Sudoku because decomposing DRAM addressing functions resembles solving a Sudoku puzzle, where each element must adhere to specific constraints.

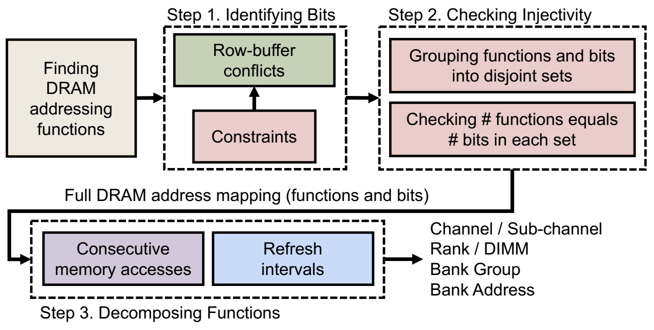
Step 1. Identifying Bits
Step 2. Checking Injectivity
Finding DRAM addressing functions
Row-buffer conflicts
Constraints
Grouping functions and bits into disjoint sets
Checking # functions equals # bits in each set
Full DRAM address mapping (functions and bits)
Consecutive memory accesses
Refresh intervals
Channel / Sub-channel
Rank / DIMM
Bank Group
Bank Address
Step 3. Decomposing Functions

Figure 5: An overview of Sudoku. Sudoku i) leverages row-buffer conflicts with constraints to identify row and column bits, ii) validates the DRAM address mapping by checking injectivity, and iii) exploits refresh intervals and consecutive memory accesses to decompose DRAM address mapping.

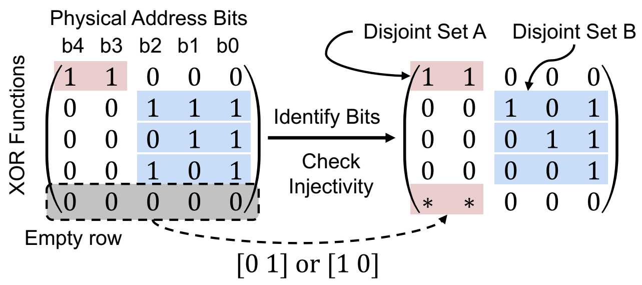

Figure 6: The process of verifying injectivity and finding an additional function. Disjoint set A (subsystem) doesn't satisfy the injectivity, meaning that an additional function or bit is required.

4.1. Generating Desired Memory Addresses

To effectively isolate the timing effects of specific DRAM components (for decomposition) or row/column bits (for identification), Sudoku requires carefully crafted memory access patterns. Instead of random testing or brute-forcing, Sudoku solves the (partially) known mapping system to generate address pairs that differ only in the output of specific target functions, while keeping other function outputs identical. This targeted approach is crucial for correctly attributing observed timing variations.

Sudoku *identifies row and column bits* using row-buffer conflicts as an oracle, guided by an educated-guessing approach similar to prior work to reduce the search space [6, 22]. The key constraint is generating address pairs mapping to the same bank (*i.e.*, having identical outputs for all bank-related hash functions). Sudoku systematically tests the role of each physical address bit by generating bitmasks. It considers masks derived from the known functions and combinations involving previously unused address bits to ensure comprehensive coverage. By generating address pairs based on these masks (XORing the mask with a base address) and observing whether accesses cause row-buffer conflicts or hits, Sudoku classifies the corresponding physical address bits in the mask as contributing to row or column indexing, respectively.

4.2. Validating the System of Hash Functions

A valid DRAM address mapping must be *injective* (one-to-one) to ensure correctness. Sudoku verifies this property by representing the derived functions and involved bits as a linear system over $GF(2)$ and applying the rank-nullity theorem [20]. Sudoku decomposes the system into disjoint subsystems and checks if, for each subsystem, the number of linearly independent functions (rank) equals the number of involved physical address bits.

However, this conflict-based testing has inherent limitations due to the non-uniqueness of XOR-based hash systems [6, 20] (as illustrated in Figure 6). It may not be possible to uniquely determine every function if multiple function/bit combinations produce the same conflict behavior. To resolve such ambiguities, Sudoku follows common assumptions (*e.g.*, assigning high-order bits to rows and low-order bits to columns [9, 18, 22]), ensuring the final mapping preserves injectivity and functional correctness, even if alternative valid representations might exist [6, 20]. Lastly, we avoid excessive function reduction (*e.g.*, reducing the system to a minimal basis) that could alter the original conflict characteristics relevant for analysis when considering both DRAM address mapping functions and row/column bits.

4.3. Decomposing DRAM Address Mapping

In its final step, Sudoku decomposes the validated set of DRAM addressing functions, assigning each function to its corresponding physical component (*e.g.*, channel, rank, bank group, and bank address). This decomposition leverages the distinct timing signatures identified in Section 3, measured using the targeted memory access patterns generated as described in Section 4.1. Sudoku systematically tests each relevant XOR function (or group of functions potentially related to a component) using the following timing analyses: refresh intervals and consecutive memory accesses.

Refresh Intervals: Sudoku generates pairs of addresses that differ only in the output bits produced by the target function(s). By measuring the interval between refresh-induced latency spikes when accessing these pairs (as detailed in Section 4.1), Sudoku identifies the functions that control the refresh group. Functions causing a “reduced” refresh interval pattern are marked as related to the refresh scope defined by the system configuration (*e.g.*, channel/sub-channel, rank/DIMM or potentially finer granularities, depending on the refresh type employed by the system).

Consecutive Memory Accesses: To further differentiate functions, especially those not distinguished by refresh or those necessitating finer classification (like bank group vs. bank address), Sudoku employs the consecutive access timing analysis (Section 3.3). It generates alternating memory streams where the underlying addresses differ only based on the target function(s). The resulting latency distribution is analyzed; the location of the primary latency peak is compared against the known `tRDRD` timing parameters of the system. A match allows Sudoku to associate the target function(s) with the cor-

Table 1: Tested system configurations.

System	Processor (microcode)	Motherboard	Memory Devices
Intel-A	Intel Core i9-12900K (`0x38`)	ASUS Z690-A	32 GB DDR4-3200 2R×8 UDIMM
Intel-B	Intel Core i9-12900K (`0x38`)	ASUS Z690-F	32 GB DDR5-4800 2R×8 UDIMM
Intel-C	Intel Core i9-14900K (`0x12C`)	MSI B760M	32 GB DDR5-4800 2R×8 UDIMM
AMD-A	AMD Ryzen 9 7950X (`0xA601206`)	ASRock X670E	32 GB DDR5-4800 2R×8 UDIMM

responding component level (*e.g.*, mapping a function to bank group if its activation leads to latency that is consistent with `tRDRD` timing for different bank groups).

By systematically applying these two timing analyses to the reverse-engineered functions using precisely controlled address generation, Sudoku labels each function with its inferred component role. This process yields the decomposed DRAM address mapping, detailing how different sets of physical address bits map to channel, sub-channel, rank/DIMM, bank group, and bank address indices.

The accuracy of this decomposition depends on the system's behavior and configuration, as noted previously. Factors such as the MC's refresh strategy, request scheduling policies, and whether timing parameters for different component interactions are distinct and measurable can *limit* the ability to differentiate certain functions (*e.g.*, channel vs. sub-channel, or rank vs. DIMM).

5. Results

Experimental setup: We evaluated Sudoku on the recent Intel and AMD systems detailed in Table 1. We used memory access latencies in CPU cycles, measured using the `rdtscp` instruction with core dynamic frequency scaling disabled to ensure stable measurements [9, 18]. For channel/sub-channel functions, which are often directly specified or easily inferred from MC-related registers (MCHBAR), we used these known values directly to constrain the reverse-engineering process, without affecting the overall mapping functionality. Lastly, we regarded recent physical probing-based DRAM address mapping results [7, 9] as ground truth and verify that Sudoku generates consistent DRAM address mapping results. Full results, including the derived XOR masks, are presented in Table 2.

Intel Core Processors: Sudoku successfully decomposed the DRAM address mapping for the tested Intel Core processors with the given DRAM addressing functions derived using prior tool [18]. Consistent with trends observed in AMD processors [9] and differing from some older Intel architectures [18, 22], these recent Intel platforms utilize most identified physical address bits within their mapping functions. Notably, for DDR5 configurations, we observed discontinuities in the physical address bits used for row and column indexing, influenced by multi-channel and multi-DPC (DIMM per channel) setups. Our timing analyses also confirmed the use of fine-grained all-bank refresh with DDR5, contrasting with the standard all-bank refresh observed with DDR4 on the same platform, demonstrating Sudoku's ability to reveal such configuration details. Lastly, it is worth noting that Sudoku can identify full DRAM address mapping, thus requiring no physical access to the system.

AMD Zen 4 Processor: On the AMD Zen 4 platform, considering the necessary PCI address offset as described in prior work, Sudoku produced decomposed mappings consistent with those reported by ZenHammer [9]. The tool successfully distinguished rank/DIMM and sub-channel functions using refresh intervals and bank group and bank functions via consecutive memory accesses. The AMD processors issue separate refresh commands for each sub-channel, each DIMM, and each rank, which means that it is possible to distinguish between channel and sub-channel functions.

6. Conclusion

This paper has demonstrated novel techniques for decomposing undocumented DRAM address mappings by exploiting previously underutilized timing channels. We first showed that DRAM refresh intervals reveal refresh group, and second, that amplified timing variations in consecutive memory accesses expose component-level mapping information. Leveraging these insights, we developed Sudoku, a software-based tool that automatically decomposes DRAM addressing functions into their specific component roles (*e.g.*, channel, rank, bank group, and bank). We validated its effectiveness on recent Intel and AMD processors. This work provides crucial, fine-grained mapping information necessary for advanced RowHammer analysis and vulnerability assessment on modern platforms. By open-sourcing Sudoku, we hope to enable further research into DRAM security and system behavior.

Acknowledgment

We thank Hwayong Nam and Michael Jaemin Kim at Seoul National University for their valuable feedback to the paper.

References

[1] Seungmin Baek, Minbok Wi, Seonyong Park, Hwayong Nam, Michael Jaemin Kim, Nam Sung Kim, and Jung Ho Ahn, "Marionette: A RowHammer Attack via Row Coupling," in *ASPLOS*, 2025.

[2] Intel Corporation, "12th Generation Intel Core Processor Datasheet, Volume 2 of 2," https://www.intel.com/content/www/us/en/content-details/655259/12th-generation-intel-core-processor-datasheet-volume-2-of-2.html, 2022, 655259-003, Accessed: April 1, 2025.

[3] Intel Corporation, "13th Generation Intel Core Processor, Volume 2 of 2," https://www.intel.com/content/www/us/en/content-details/743846/13th-generation-intel-core-processors-datasheet-volume-2-of-2.html, 2022, 743846-001, Accessed: April 1, 2025.

[4] Advanced Micro Devices, "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 70h-7Fh Processors," https://www.amd.com/content/dam/amd/en/documents/archived-tech-docs/programmer-references/55072_AMD_Family_15h_Models_70h-7Fh_BKDG.pdf, 2018, 55072 Rev 3.09, Accessed: April 1, 2025.

[5] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[6] Jana Hofmann, Cédric Fournet, Boris Köpf, and Stavros Volos, "Gaussian Elimination of Side-Channels: Linear Algebra for Memory Coloring," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[7] Patrick Jattke, Michele Marazzi, Flavien Solt, Max Wipfli, Stefan Gloor, and Kaveh Razavi, "MCSEE: Evaluating Advanced Rowhammer Attacks and Defenses via Automated DRAM Traffic Analysis," in *Proceedings of the 34th USENIX Security Symposium (USENIX Security 25)*, 2025.

Table 2: Reverse-engineered DRAM addressing functions and decomposed results using Sudoku. N/A denotes that there is no function for the corresponding component. Each value in the table represents a mask for an XOR-based hash function, where the parity of the set bits in the mask determines the output.

System	Memory Configuration	Functions						Bits	
				Banks					
		Channel and Sub-Channel	DIMM and Rank	Bank Group		Bank Address		Row	Column
Intel-A	1Ch-1DPC	N/A	0x0000088000	0x0000002A00,	0x0124044000	0x0249910000,	0x0492620000	0x07FFFC0000	0x0000001FC0
	1Ch-2DPC	N/A	0x0000108000, 0x0000420000	0x0000002A00,	0x0924084000	0x0249210000,	0x0492840000	0x0FFFF80000	0x0000001FC0
	2Ch-1DPC	0x0000082600	0x0000110000	0x0000005400,	0x0248088000	0x0493220000,	0x0924C40000	0x0FFFF80000	0x0000001FC0
	2Ch-2DPC	0x0000082600	0x0000210000, 0x0000840000	0x0000005400,	0x1248108000	0x0492420000,	0x0925080000	0x1FFFF00000	0x0000001FC0
Intel-B, Intel-C	1Ch-1DPC	0x00000C3200	0x0000410000	0x0000081100,	0x0222104000, 0x0442080000	0x0088820000,	0x0111040000	0x07FFF80000	0x0000000FC0
	1Ch-2DPC	0x00000C3200	0x0000810000, 0x0001040000	0x0000081100,	0x0222104000, 0x0444408000	0x0114100000,	0x088A020000	0x0FFFE80000	0x0000000FC0
	2Ch-1DPC	0x0000104200, 0x0000186400	0x0000820000	0x0000102100,	0x0444208000, 0x0888410000	0x0111040000,	0x0222080000	0x0FFFF00000	0x0000001BC0
	2Ch-2DPC	0x0000104200, 0x0000186400	0x0001020000, 0x0002080000	0x0000102100,	0x0444408000, 0x0888810000	0x0228200000,	0x1114040000	0x1FFFD00000	0x0000001BC0
AMD-A	1Ch-1DPC	0x07FFF80040	0x0000040000	0x0084200100,	0x0108400200, 0x0210801000	0x0042100800,	0x0421080400	0x07FFF80000	0x000003E080
	1Ch-2DPC	0x0FFFF00040	0x0000040000, 0x0000080000	0x0108400100,	0x0210800200, 0x0421001000	0x0084200800,	0x0842100400	0x0FFFF00000	0x000003E080
	2Ch-1DPC	0x0000000100, 0x0FFFF00040	0x0000080000	0x0108400200,	0x0210800400, 0x0421002000	0x0084201000,	0x0842100800	0x0FFFF00000	0x000007C080
	2Ch-2DPC	0x0000000100, 0x1FFFE00040	0x0000080000, 0x0000100000	0x0210800200,	0x0421000400, 0x0842002000	0x0108401000,	0x1084200800	0x1FFFE00000	0x000007C080

[8] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[9] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi, "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[10] JEDEC, "DDR4 SDRAM," 2017.

[11] JEDEC, "DDR5 SDRAM," 2024.

[12] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom, "SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[13] Andreas Kogler, Daniel Weber, Martin Haubenwallner, Moritz Lipp, Daniel Gruss, and Michael Schwarz, "Finding and Exploiting CPU Features using MSR Templating," in *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[14] Linux, "Error Detection And Correction (EDAC) Devices," https://docs.kernel.org/driver-api/edac.html, accessed: April 1, 2025.

[15] Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci, "Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer," in *Proceedings of the 29th Symposium on Operating Sytems Principles (SOSP)*, 2023.

[16] Heckel Martin and Florian Adamsky, "Reverse-Engineering Bank Addressing Functions on AMD CPUs," in *3rd Workshop on DRAM Security (DRAMSec)*, 2023.

[17] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn, "DRAMScope: Uncovering DRAM Microarchitecture and Characteristics by Issuing Memory Commands," in *ISCA*, 2024.

[18] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[19] Victor van der Veen and Ben Gras, "DramaQueen: Revisiting Side Channels in DRAM," in *3rd Workshop on DRAM Security (DRAMSec)*, 2023.

[20] Hans Vandierendonck and Koenraad De Bosschere, "XOR-Based Hash Functions," *IEEE Transactions on Computers*, 2005.

[21] Stavros Volos, Cédric Fournet, Jana Hofmann, Boris Köpf, and Oleksii Oleksenko, "Principled Microarchitectural Isolation on Cloud CPUs," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[22] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal, "DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping," in *Design Automation Conference (DAC)*, 2020.