Verifying DRAM Addressing in Software

Martin Heckel[1,2], Florian Adamsky[1], Jonas Juffinger[2], Fabian Rauscher[2], and Daniel Gruss[2]

[1] Hof University of Applied Sciences, Germany
[2] Graz University of Technology, Austria

Abstract. In this paper, we introduce a novel approach to reliably verifying DRAM addressing functions and function components from software. We perform the first systematic analysis of 5 DRAM function reverse-engineering tools on 2 different DDR3, 4 DDR4, and 4 DDR5 system configurations, revealing a significant variance in the success rate of these tools, from 0 % to 92.9 %. We discover the previously unknown rank selection side channel and reverse engineer its function on two DDR4 and two DDR5 systems. These results enable novel DDR5 row-conflict side-channel attacks, which we demonstrate in two scenarios: First, we evaluate the DDR5 row-conflict side channel in a covert channel with 1.39 Mbit/s. Second, we evaluate the channel in a website fingerprinting attack with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

1 Introduction

Software-level security often assumes that hardware is functioning correctly and without side effects. However, physical effects can undermine system security with side-channel and fault attacks. One attack target is DRAM, the main memory in modern computers. There are side channels [27, 41], fault attacks [14, 5], and slowdown attacks [24] on DRAM, undermining a system's confidentiality, integrity, and availability. DRAM addressing functions can be used in performance optimization, such as application-aware memory channel partitioning [25] or variable page sizes [38] for more efficient row-buffer usage. However, DRAM side-channel and fault attacks also often use these functions: Pessl et al. [27] presented the first DRAM side-channel attacks exploiting reverse-engineered addressing functions. Seaborn [34, 33] used DRAM addressing functions for the first practical *Rowhammer* exploit. Rambleed [17] exploits bit flips in attacker memory, depending on inaccessible victim data bits and addressing functions.

DRAM addressing functions are defined by the CPU's memory controller, i. e., specific to the CPU model and the memory configuration. Consequently, it is necessary to reverse-engineer the functions **anew** for every system. Prior works utilize DRAM access timings [27, 40, 4, 8]. Helm et al. [9] reverse-engineered the DRAM functions using performance counters. Most works did not verify the functions systematically but only tested whether an attack succeeded. However, Rowhammer bit flips can also occur with incorrect or even without addressing functions, e. g., One-Location Rowhammer [5]. Only Pessl et al. [27] and Jattke et al. [12] verified addressing functions using high-bandwidth oscilloscopes.

In this paper, we present a novel methodology to verify the correctness of DRAM addressing functions purely in software. Our approach is based on the fact that the theoretical success rate of DRAM side channel tests changes with each function component. Consequently, we can verify the correctness of even single output bit of the DRAM addressing functions purely from software. Thus, we can identify incorrect function components and combine the outputs of multiple reverse-engineering tools to a complete and correct a set of functions. We evaluate our approach on 10 systems and show that the maximum deviation from theoretical values for correct functions is 0.76 % on DDR3, 0.52 % on DDR4, and 0.49 % on DDR5, indicating the high precision of our approach.

Based on our novel verification methodology, we present the first systematic analysis of DRAM addressing function reverse-engineering approaches. We observe success rates of 92.9 % on DDR3, 85.6 % on DDR4, and 87.3 % on DDR5 for the full DRAM addressing functions with the best respective reverse-engineering tool [27, 40, 4, 8, 12]. We show that 3 tools yield good results (i. e., $\approx$ 90 % correct) on our DDR3 systems (Intel), 4 tools yield good results (i. e., $\approx$ 80 % correct) on our DDR4 systems (Intel), and only 1 tool yields moderate results (i. e., $\approx$ 60 % correct) on DDR4 (AMD) and good results (i. e., $\approx$ 85 % correct) on DDR5 (Intel). No tool yields good results on DDR5 (AMD).

Using our approach, we found an additional layer in DRAM addressing: There is a measurable timing difference between addresses of the same rank and those of different ranks. We strongly suspect this is due to the rank select commands sent between accesses to different ranks. We are the first to reverse-engineer *rank addressing functions* and show that using such functions increases the success rate by 18.36 % to 36.11 % on different systems.

We built the first row-conflict covert channel on DDR5 with a true capacity of up to 2.23 Mbit/s on DDR3, 0.66 Mbit/s on DDR4, and 1.39 Mbit/s on DDR5. Additionally, we evaluate our approach in a row-conflict side-channel website-fingerprinting attack, identifying a single website out of 100 with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

In summary, in this paper, we make the following contributions:

1. We present a new approach for DRAM address function verification in software[3] based on computing deviations from the theoretical behavior.
2. We systematically evaluate 5 reverse-engineering tools on 2 DDR3, 4 DDR4, and 4 DDR5 systems and show that none produce good results on all systems.
3. We discover a novel rank selection timing channel and reverse-engineer the corresponding rank function on two DDR4 and two DDR5 systems.
4. We present the first row-conflict covert channel on DDR5[4]. with 1.39 Mbit/s.
5. We demonstrate a novel row-conflict-based side-channel attack on DDR5, allowing to distinguish 100 websites with an F_1 score of 74 %.

Outline. Section 2 provides background. Section 3 describes our setup. Section 4 presents our new function verification approach. Section 5 presents our DRAM

[3] https://github.com/iisys-sns/DramaVerify
[4] https://github.com/iisys-sns/DramaNg

rank addressing insights. Section 6 presents our covert channel on DDR5, and Section 7 our website-fingerprinting attack. Section 8 concludes.

2 Background and Related Work

This section discusses DRAM and DRAM addressing, covert channels, website fingerprinting, and related work.

DRAM. DRAM cells consist of capacitors and transistors organized in rows and columns, forming DRAM banks grouped into ranks on a DIMM [27]. DIMMs are connected to the CPU's memory controller via channels. Activating cells (reading the data into a row buffer) is destructive, so content must be written back before activating another row. Since capacitors lose charge over time, DRAM needs periodic recharging, e. g., every 64 ms.

DRAM Addressing Functions. Addressing functions map consecutive memory into different banks, ranks, and channels to minimize bank conflicts. The memory controller translates physical addresses via addressing functions into DRAM components, i. e., channels, DIMMs, ranks, banks, rows, and columns.

Linear Addressing functions are used on systems where the number of all DRAM components is a power of two. They are represented as a hexadecimal bitmask, indicating which bits need to be XORed. For example, the addressing function $0x88000$ corresponds to 10001000000000000000_2, indicating that the 19th and the 15th bit of the physical address needs to be XORed. Each function distinguishes two states, meaning there are $log_2(n_{\text{banks}})$ addressing functions on a system with n_{banks} DRAM banks. Reverse-engineering non-linear addressing functions (which use operations other than XOR) remains an open problem.

Covert Channels. Various microarchitectural elements have been used in covert communication channels [18], e. g., via CPU load [26], CPU caches [42, 45, 46, 22, 21, 23, 28, 7, 31], and the memory bus [44, 43]. Covert channels have become a best practice to evaluate the capacity of side channels. Semal et al. [35] present a DRAMA covert channel on DDR3 and DDR4 systems with up to 729 bit/s. Wang et al. [41] show that this channel also affects Intel SGX. Van der Veen et al. [39] amplify the DRAMA channel by making all memory uncacheable. In a simulation, Kushwaha et al. [16] showed certain *secure* cache designs can also amplify the DRAMA channel from a simulated range of 2.73 Mbit/s to 4.61 Mbit/s to 4.53 Mbit/s to 6.82 Mbit/s. The fastest DRAM covert channel to date [27] achieves a capacity of up to 2 Mbit/s on DDR4.

Website-Fingerprinting Side Channels. A common scenario for side-channel evaluation with low spatial resolution is website fingerprinting. Fingerprinting attacks exploited Android data usage [37], browser memory usage [10], the power side channel [29], cache occupancy [36], interrupt timing [3, 47, 30], SSD contention [13], and timing side channels in the operating system [19], often achieving high F1 scores in open- and closed-world scenarios on the top 100 websites.

Related Work. Several Rowhammer exploits do not use DRAM bank addressing functions [14, 32, 5]. However, newer and sophisticated Rowhammer

exploits [32, 6, 4, 15, 11] use DRAM addressing functions to increase the number of bit flips with more targeted hammering.

Pessl et al. [27] reverse-engineered DRAM bank addressing functions from software and used physical memory-bus probing for verification. Barenghi et al. [2] and Marazzi et al. [20] also use the timing-based approach to reverse-engineer DRAM addressing functions. Prior work often determines the correctness through success rates of resulting attacks, e. g., the overall success rate of the full set of addressing functions. In contrast, we demonstrate that the *theoretical influence of correct functions can be leveraged as ground truth to verify the correctness of single reverse-engineered functions*. While we also discover the rank selection side channel that was previously unknown, we focus our verification on the known row-conflict side channel with addressing functions from prior work (which did not include rank selection functions).

DRAMDig [40] is a knowledge-assisted tool to determine DRAM address mappings and then run a double-sided rowhammer test for verification. Zhang et al. [48] extended DRAMDig to a Rowhammer testing tool using reverse-engineered addressing functions. Frigo et al. [4] trigger bit flips in TRR-enabled DIMMs using a many-sided pattern with many aggressor rows, relying on reverse-engineering DRAM addressing functions. Helm et al. [9] use performance-counter-based reverse-engineering, focusing on Intel Haswell, Broadwell, and Skylake. Since we also cover AMD, we cannot use their approach in our measurements.

Fewer works focused on AMD. AMD published functions on older CPUs [1] but not for newer ones, where Heckel et al. [8] and Jattke et al. [12] recently reverse-engineered DRAM addressing functions. They also reverse-engineered DRAM addressing functions on DDR5 utilizing the row-conflict side-channel we utilize for the attacks presented in this paper. Jattke et al. [12] verify the function correctness with an oscilloscope, like Pessl et al. [27].

3 Experimental Setup

Our experimental setup consists of 2 systems with DDR3, 4 with DDR4, and 4 with DDR5 DRAM. Each system has a unique ID that follows the format: `S⟨DDR Version⟩⟨Counter⟩`. All systems run a current version of Arch Linux (6.8.7-arch1-1). Since rank addressing functions are a significant part of the experimental evaluation, we use DIMMs with one or two ranks. If we have multiple similar systems, we use a DIMM with one rank in one system and a DIMM with two ranks in the other (see Table 1 for details). We use `/proc/self/pagemap` to get physical to virtual addresses mappings.

4 Verification of DRAM Addressing Functions

This section shows how we verify DRAM bank addressing functions based on Drama [27]. Rather than exploiting the varying timings between row hits and conflicts to create a covert channel, we utilize this side channel to confirm the

Table 1: Systems used for experimental evaluation. The number of banks (n_{bnk}) is the number of all banks in the system, e. g., there are $\frac{n_{\text{banks}}}{n_{\text{rnk}}}$ banks per rank.

	System CPU	Memory	n_{bnk}	n_{rnk}
DDR3	*S301 i5-3320M*	4 GiB	8	1
DDR3	*S302 i7-4800MQ*	4 GiB	16	2
DDR4	*S401 i9-10900K*	8 GiB	16	1
DDR4	*S402 i9-10900K*	8 GiB	32	2
DDR4	*S403 5950X*	8 GiB	16	1
DDR4	*S404 5950X*	8 GiB	32	2
DDR5	*S501 i7-13700*	16 GiB	64	1
DDR5	*S502 i7-13700*	16 GiB	64	1
DDR5	*S503 7700X*	16 GiB	64	1
DDR5	*S504 7700X*	16 GiB	64	1

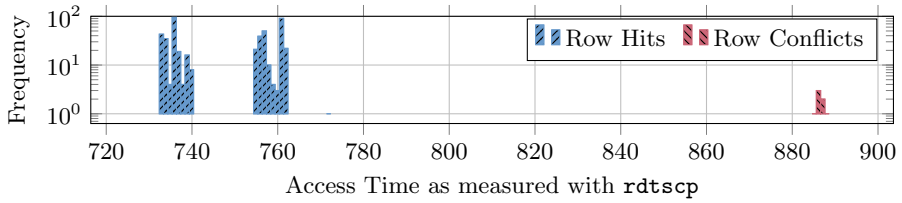


Fig. 1: Histogram of access times when accessing randomly selected pairs of addresses in a flush and reload loop. The gap around 825 TSC cycles hints at the threshold between row hits and misses on *S404*.

accuracy of the bank addressing functions. Although we use the same side channel from DRAMA, we utilize it *differently*, as discussed in Section 4.1 We verify our approach using DRAM bank addressing functions reverse-engineered by five existing tools [27, 40, 4, 8, 12]. Our criterion for selecting the tools was that they were published in the last ten years and do not require features available only on one architecture. We omitted Helm et al. [9] since their approach uses performance counters exclusive to Intel CPUs from Haswell to Skylake.

4.1 Verification Steps

This section describes the steps to verify reverse-engineered DRAM bank addressing functions. First, we measure the threshold between a row hit and a conflict before allocating 1 GiB of memory. We resolve the physical addresses with elevated privileges, group the allocated memory based on the given addressing functions, and run the DRAMA side channel. We compute a success rate for the addressing function from the number of row hits and conflicts.

Measure the Threshold between Row Hit and Row Conflict. We allocate one 2 MiB Transparent Huge Page (THP) (512 single 4 KiB pages) and measure the access time using the `rdtsc` instruction of the first (offset 0) and second (offset 4 KiB) page within the THP. We repeat this for all other pages within the THP, e. g., comparing the first and the third, and so on. Between measurements, we clear the CPU cache with `clflush`.

A histogram from the measured access times is shown in Figure 1. While we expect the histogram to have two peaks, one for fast row hits and one for slow row conflicts, there are three, two with row hit timings. This can be explained by the fact that *S404* has two ranks. We also performed this experiment on *S401*,

which has only one rank and only one row-hit peak. Accesses to addresses on the same rank (*rank hits*) do not require the memory controller to issue a rank select command between them. Access to addresses on different ranks (*rank conflicts*) required the memory controller to issue rank select commands. For this reason, those accesses are slightly slower. There are two ranks on *S404*, and the addresses are equally distributed over both ranks, so there are two peaks of similar size. The histogram is analyzed to identify the threshold t_T between the *row hit* peak and the *row conflict* peak (around 825 for the histogram shown in the plot).

Grouping Addresses. We allocate 1 GiB and get the physical addresses from `/proc/self/pagemap`. With 2^{30} single addresses in 1 GiB, only a fraction (1 % by default) is used. We apply all reverse-engineered DRAM addressing functions under test to the physical addresses. The bits are XORed, and the resulting bit is considered a bit of the bank number. When n addressing functions are applied to a physical address, there is an n-bit bank number. Finally, the virtual address is added to the group to match the bank number computed before.

Verification. After address grouping, the row-conflict side channel [27] is used for group verification. Only correct addressing functions lead to correct groups, so it is sufficient to verify that the groups are correct to derive that the addressing functions are correct. There are 2^n groups for n functions. Several address pairs (a_1, a_2) (we use 5 000 by default, since this number yielded good results in a brief comparison) are randomly selected for each group. Additionally, an address (b_1) from another randomly selected group is selected.

Then, we measure the timing of alternatingly accessing (a_1, a_2), expecting a row conflict ($t_c > t_T$), and (a_1, b_1), expecting a row hit ($t_h < t_T$). Each of these timing measurements is performed a few hundred times, and the median of the measurements is used as the resulting value. If both timings are correct, $t_c > t_T$ and $t_h < t_T$, the address pair is considered to be grouped correctly.

Evaluation. We compute the share of correctly grouped pairs from all address pairs, resulting in a correctness measure for reverse-engineered DRAM bank addressing functions. In Section 4.2, we experimentally evaluate our approach.

4.2 Experimental Evaluation

We evaluate our approach with the following reverse engineering tools: DARE [12], DRAMA [27], DRAMDIG [40], TRRESPASS RE Tool [4], and AMDRE [8]. After **identifying the addressing functions with these tools**, we ran our verification method on these functions 10 times. We report the average percentage of cases where the assumed banks are correct. We show a graphical representation of all bits identified to belong to at least one DRAM bank addressing function. Therefore, it is possible to see how stable the functions were over multiple measurements. Additionally, to function correctness, we evaluate the stability of the tools as follows: **Stable** tools yield the same result upon every execution. **Mostly Stable** tools yield the same result in ≥ 70 % of runs and only 1 or 2 bits difference in the other cases, or no result in at most 3 runs. **Unstable** tools yield the same result in ≥ 70 % of runs, but other runs vary. **Completely Unstable** tools have varying results for all runs. **Failed** tools crashed or returned nothing.

Table 2: Experimentation results of multiple PoCs [27, 40, 4, 8, 12] on our DDR3 systems. 10 measurements were performed. AFn Mask shows a graphical representation of all bits belonging to at least one DRAM bank addressing function. The average percentage $\%_{\text{avg}}$ is lower than $\%_{\text{max}}$ when some runs failed and some succeeded.

	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$
S301	AMDRE		83.5 %	27.4	1.4 %	92.9 %	AMDRE		90.1 %	0.1	90.0 %	90.3 %
	DRAMDig		92.7 %	0.1	92.5 %	92.8 %	DRAMDig		89.8 %	0.8	87.3 %	90.2 %
	Drama		43.9 %	34.1	8.3 %	92.7 %	Drama		42.3 %	35.4	0.0 %	90.1 %
	Dare		0.0 %	0.0	0.0 %	0.0 %	Dare		0.0 %	0.0	0.0 %	0.0 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %	TRRespass		0.0 %	0.0	0.0 %	0.0 %

Table 3: Experimentation results ([27, 40, 4, 8, 12]) on our DDR4 systems (left) and DDR5 systems (right), analogue to Table 3.

	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$
S401	AMDRE		85.4 %	0.1	85.2 %	85.5 %	AMDRE		42.9 %	42.9	0.0 %	86.0 %
	DRAMDig		84.8 %	0.2	84.4 %	85.1 %	DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		15.3 %	15.6	0.0 %	44.9 %	Drama		5.1 %	1.8	1.6 %	6.3 %
	Dare		76.8 %	25.6	0.0 %	85.5 %	Dare		6.3 %	0.0	6.3 %	6.4 %
	TRRespass		84.1 %	2.1	79.6 %	85.6 %	TRRespass		5.4 %	1.2	2.9 %	6.1 %
S402	AMDRE		77.2 %	0.2	76.9 %	77.4 %	AMDRE		87.0 %	0.2	86.7 %	87.3 %
	DRAMDig		42.3 %	0.1	42.0 %	42.5 %	DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		6.7 %	6.5	0.0 %	21.3 %	Drama		4.0 %	2.2	0.0 %	5.5 %
	Dare		29.6 %	19.4	0.0 %	42.4 %	Dare		4.6 %	1.7	0.0 %	5.4 %
	TRRespass		42.2 %	0.1	42.0 %	42.4 %	TRRespass		5.4 %	0.1	5.2 %	5.5 %
S403	AMDRE		23.2 %	13.0	0.0 %	38.8 %	AMDRE		0.0 %	0.0	0.0 %	0.0 %
	DRAMDig		0.0 %	0.0	0.0 %	0.0 %	DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		13.9 %	9.7	0.0 %	23.1 %	Drama		22.7 %	0.6	21.8 %	23.7 %
	Dare		14.4 %	6.0	0.0 %	21.6 %	Dare		1.2 %	1.2	0.0 %	2.6 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %	TRRespass		0.0 %	0.0	0.0 %	0.0 %
S404	AMDRE		62.3 %	0.5	61.8 %	63.4 %	AMDRE		0.0 %	0.0	0.0 %	0.0 %
	DRAMDig		0.0 %	0.0	0.0 %	0.0 %	DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		16.0 %	5.8	7.3 %	21.8 %	Drama		20.6 %	6.9	0.0 %	23.6 %
	Dare		18.5 %	1.6	16.1 %	20.5 %	Dare		18.9 %	9.5	0.0 %	23.7 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %	TRRespass		0.0 %	0.0	0.0 %	0.0 %

DDR3. Table 2 summarizes our results on two DDR3 systems. DRAMDig consistently identified *stable* addressing functions with success rates around 92 %. AMDRE produced *mostly stable* results but with high variance (1.4 % to 92.9 %). In contrast, Drama was *completely unstable*, ranging from 0 % to 92.7 %. Dare and TRRespass *failed* on both systems.

DDR4. As shown in Table 3, on *S401* and *S402*, AMDRE, DRAMDig, and TRRespass returned *stable* functions, Dare reported *mostly stable* functions and Drama reported *completely unstable* functions. Drama has a success rate of 0 % to 44.9 %. The maximum success rate for AMDRE, DRAMDig and Dare is around 77 % to 85 % and around 42 % to 85 % for TRRespass. The minimum success rate is approximately 85 % for AMDRE and DRAMDig, it is 79.6 % for TRRespass and 0 % to 42 % for Dare.

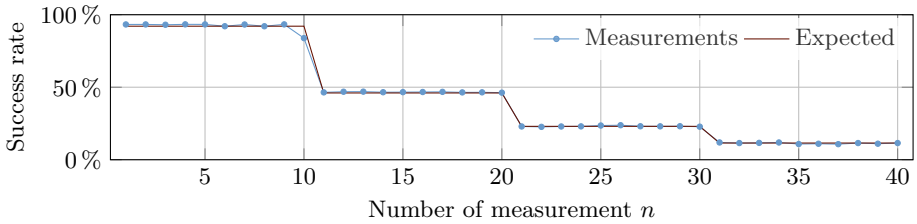


Fig. 2: Success rate reported depending on the number of addressing functions submitted. The measurement was done on *S401* with 4 addressing functions. For each number of addressing functions, 10 measurements were performed.

On *S403* and *S404*, DRAMDig and TRRespass *failed*. The masks of Dare are *stable* and *mostly stable*. AMDRE returns *completely unstable* and *stable* results. The success rates are generally lower than for the other two systems.

DDR5. Table 3 shows the results for our four DDR5 systems. No tool has stable results across all machines. DRAMDig *failed* on all machines and AMDRE and TRRespass on *S503* and *S504*. AMDRE is only *stable* on one machine, TRRespass is *unstable* on the two machines where it works. Drama was *unstable* or *completely unstable* across all machines. The success rates are generally lower than for DDR4, with AMDRE reaching the highest success rates of approximately 87 % on two systems. Dare has *stable* and *mostly stable* results with maximum success rates of 2.6 % to 23.7 %.

4.3 Verification of Single Addressing Functions

We verify the entire set of addressing functions in Section 4.2. However, with that, we cannot make any statement about single functions. This section extends the approach from Section 4.2 to verify single addressing functions within a set.

If a system has n_{fn} correct DRAM bank addressing functions, the number of banks is $n_{\text{banks}} = 2^{n_{\text{fn}}}$. If we remove one of these functions, the number of banks addressable halves: $2^{n_{\text{fn}}-1} = \frac{2^{n_{\text{fn}}}}{2} = \frac{n_{\text{banks}}}{2}$. No addresses can be added to half of the banks because they can not be addressed with the reduced number of functions. This results in 50 % of the DRAM banks no longer being accessible via addressing functions and reduces the success rate by 50 %, as shown in Figure 2.

In the range $0 < n \leq 10$ we use all DRAM addressing functions `0x2040`, `0x24000`, `0x48000`, and `0x90000`. The average success rate is 92.08 %. For $10 < n \leq 20$, we remove `0x90000` and the average success rate halves to 46.61 %. In the $20 < n \leq 30$ range, we remove `0x48000` resulting in an average success rate of 23.09 %. Finally, for $30 < n \leq 40$, the function `0x2040` is used, and the average success rate is 11.35 %, with an expected success rate of 11.51 %.

We conclude that all the DRAM addressing functions used above are correct. However, this approach has the disadvantage that the differences between using or not using a function get lower the more functions were removed before. For example, the expected difference for removing the first function is 46.04 %, which

Table 4: Evaluation of single addressing functions grouped by system. The value of $\%_{\text{exp}}$ is always 50 % of the measured initial success rate for the entire function set. Manual manipulation to obtain wrong functions (see the ✓ under *Mod*).

	Function	Mod.	$\%_{\text{meas}}$	$\%_{\text{exp}}$	$\%_{\text{diff}}$	Cor.		Function	Mod.	$\%_{\text{meas}}$	$\%_{\text{exp}}$	$\%_{\text{diff}}$	Cor.
S302	0x22000	✗	47.6 %	46.1 %	1.5 %	✓	*S302*	0x23000	✓	23.1 %	10.7 %	12.4 %	✗
	0x44000	✗	47.6 %	46.1 %	1.5 %	✓		0x44000	✗	10.9 %	10.7 %	0.2 %	✓
	0x88000	✗	47.8 %	46.1 %	1.7 %	✓		0x89000	✓	22.6 %	10.7 %	11.9 %	✗
	0x110000	✗	47.7 %	46.1 %	1.6 %	✓		0x110000	✗	10.9 %	10.7 %	0.2 %	✓
S401	0x2040	✗	46.1 %	46.6 %	0.5 %	✓	*S401*	0x2040	✗	10.4 %	10.4 %	0.0 %	✓
	0x24000	✗	46.4 %	46.6 %	0.2 %	✓		0x25000	✓	20.8 %	10.4 %	10.4 %	✗
	0x48000	✗	46.4 %	46.6 %	0.2 %	✓		0x48000	✗	10.5 %	10.4 %	0.1 %	✓
	0x90000	✗	46.3 %	46.6 %	0.3 %	✓		0x110000	✓	22.6 %	10.4 %	12.2 %	✗
S501	0x6300	✗	43.6 %	43.3 %	0.3 %	✓	*S501*	0x6100	✓	9.43 %	3.11 %	6.32 %	✗
	0x10000	✗	43.6 %	43.3 %	0.3 %	✓		0x10000	✗	3.07 %	3.11 %	0.04 %	✓
	0x20000	✗	43.8 %	43.3 %	0.5 %	✓		0x24000	✓	9.19 %	3.11 %	6.08 %	✗
	0x42300	✗	43.4 %	43.3 %	0.1 %	✓		0x42300	✗	3.08 %	3.11 %	0.03 %	✓
	0x81100	✗	43.5 %	43.3 %	0.2 %	✓		0x82100	✓	9.36 %	3.11 %	6.25 %	✗
	0x108000	✗	43.3 %	43.3 %	0.0 %	✓		0x108000	✗	3.08 %	3.11 %	0.03 %	✓

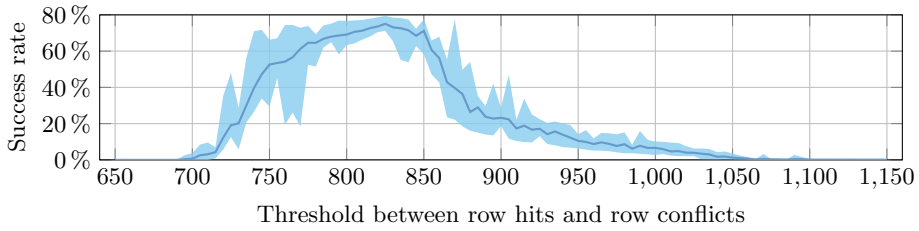
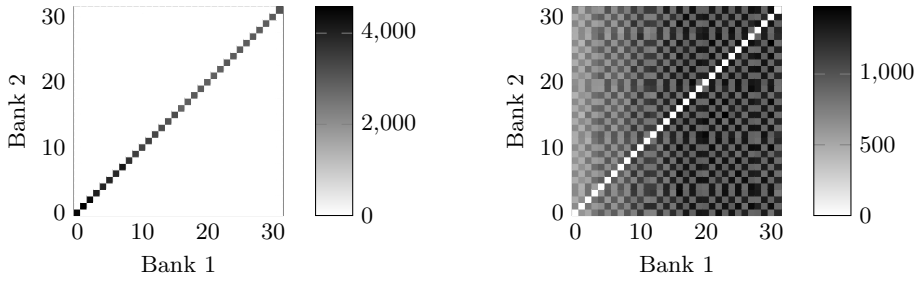


Fig. 3: Success rate of our verification approach depending on the threshold. The graph shows average, minimum, and maximum values over 10 measurements.

decreases to 11.51 % for removing the third function. However, as DRAM bank addressing functions are not ordered, we remove every function from the initial set of all functions individually, comparing the new success rate to the initial one. If a correct function is removed, the success rate is expected to halve.

We evaluate our verification on DDR3, DDR4, and DDR5, each with the set of addressing functions reverse-engineered in Section 4.2 that yields the highest percentage on the respective system. Then, we manually modify some of the DRAM addressing functions. Afterward, we repeat the experiment with the modified function to verify that our approach can detect the modified, now wrong, addressing functions. The results of this evaluation are shown in Table 4.

For *S301*, *S401*, and *S501*, the submitted addressing functions were identified to be correct, with a maximum difference of 0.76 % between the expected and the measured values when no modifications were performed. Without any modified functions, all systems' base success rate ($2 \times \%_{\text{exp}}$) is approximately 90 %. With two modified functions, the success rate drops to 21.34 % on *S301*. Both incorrect functions were identified, with differences of 11.43 % and 11.80 %. The



(a) Number of too fast row conflicts.

(b) Number of too slow row hits.

Fig. 4: Heat maps of both errors: too fast row conflicts and too slow row hits. The x-axis shows the group of the first address, the y-axis the second selected address. Measurements on *S404* (with threshold 495), averaged over 10 measurements.

correct function was identified with a difference of 0.71 %. On *S401*, two modified addressing functions resulted in an overall success rate of 20.88 %. The incorrect functions were identified, with differences of 10.39 % and 12.18 %. Both correct functions were identified, with differences of 0.01 % and 0.08 %. The three modified addressing functions on *S501* resulted in a drop in the overall success rate of 6.22 %. The incorrect functions have differences of 6.32 %, 6.08 %, and 6.25 %. In contrast, the three correct functions have differences of 0.04 %, 0.03 %, and 0.03 %. Our approach identifies correct and incorrect DRAM bank addressing functions in all cases within this experiment.

5 Rank Timing Analysis

To analyze the lower-than-expected success rate (see Section 4.2), we perform multiple measurements and finally show that a second layer of addressing functions is used to determine the rank of a physical address.

5.1 Rank Addressing Functions

We use the same timing notation introduced in Section 4.1 and find four timing cases, **C1**, with $t_c > t_T$ and $t_h < t_T$, **E1**, with $t_c < t_T$ and $t_h < t_T$, **E2**, with $t_c < t_T$ and $t_h > t_T$, and **E3**, with $t_c > t_T$ and $t_h > t_T$. When the addresses are correctly grouped, and both addresses from the same group are not in the same row, t_c cannot be smaller than t_h since a row hit (t_h) is faster than a row conflict (t_c). Depending on the addresses chosen, the case labeled E2 might occur even when correct addressing functions are used. The probability of this happening is discussed below. By default, 1 % of the available addresses is grouped, which is 10 MiB of 1 GiB. If all selected addresses are contiguous, 10 MiB of memory are distributed over n_{banks} banks. In the case of 32 DRAM banks, this results in $\frac{10\,\text{MiB}}{32} = 320\,\text{KiB}$ per DRAM bank. With the assumption that the addresses

completely populate contiguous rows with a row size of 8 KiB, the selected addresses populate $\frac{320\,\text{KiB}}{8\,\text{KiB}} = 40$ rows. So, the probability of one randomly selected address being in one chosen row is $\frac{1}{40} = 2.5\,\%$. Following this, the probability of two randomly selected addresses being in the same DRAM bank is 2.5 %. Therefore, case E2 is doubtful when correct DRAM bank addressing functions are used, since two randomly selected addresses from the same DRAM bank would have to be in the same row, which happens at a probability of 2.5 % in the worst case. Under the assumption that the tested DRAM bank addressing functions are correct, the cases E1, C1, and E3 are statistically relevant.

In contrast to the approach described in Section 4.1, the threshold is manually specified and not measured for this experiment. The threshold impacts the success rate, as shown in Figure 3. The statistically relevant cases E1, C1, and E3 depend on the selected threshold as shown below:

E1: Since both timings (t_c and t_h) are lower than the threshold, row conflicts are misclassified as row hits. The threshold is selected too high (for $t_T \geq 875$ in the graph shown in Figure 3).

C1: Since row conflicts (t_c) are slower and row hits (t_h) are faster than the threshold, both cases are classified correctly. The threshold is set correctly (for $875 \geq t_T \geq 795$ in the graph shown in Figure 3).

E3: Since both timings (t_c and t_h) are higher than the threshold, hits are misclassified as conflicts. The threshold is too low (for $t_T \leq 795$ in Figure 3).

Note that the three cases overlap and merge, so the submitted threshold values describe a range and not a specific value. In these measurements, two types of single errors can occur: **(e1)** a row hit is too slow and misclassified as conflict; and **(e2)** a row conflict is too fast and misclassified as hit.

Figure 4 shows heat maps for both errors. Row conflicts are expected when comparing addresses from the same DRAM bank, so there are n_{banks} different cases (one for each DRAM bank), as shown in Figure 4a. Row hits are expected between addresses from one DRAM bank and addresses from any other DRAM bank, as shown in Figure 4b. The heat map shows no values on the diagonal since addresses from the same bank are not expected to be row hits.

As shown in Figure 4b, there is a pattern in the error number depending on the addresses of which DRAM banks are compared to each other. When comparing an address from bank 0 to another address, the number of errors is lower for the following banks: (2, 5, 7, 9, 11, 12, 14, 16, 18, 21, 23, 25, 27, 28, 30). Similarly, comparing an address from bank 1 with another address has fewer errors for the following banks: (3, 4, 6, 8, 10, 13, 15, 17, 19, 20, 22, 24, 26, 29, 31). So, there are two groups of DRAM banks for which the error rate is lower when addresses are selected from banks within the same group. At the same time, the error rate is higher when selecting two addresses from banks in different groups.

This grouping can be described by a *grouping function* taking bank number (0–31) as input and returning an output of one bit equivalent to the number of the groups. The function can be represented by a bitmask that selects bits of the input and applies a bitwise XOR to them. When bank numbers are represented

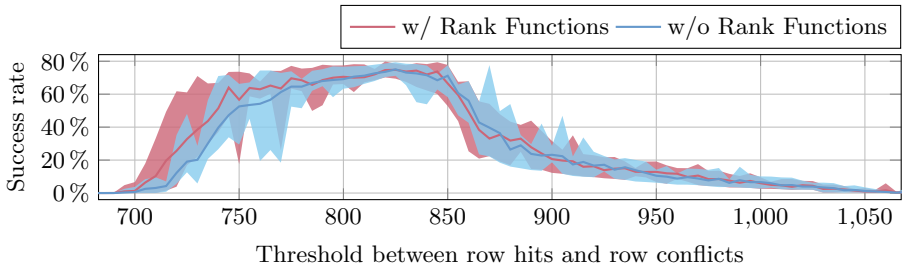


Fig. 5: Average, minimum and maximum success rate (10 measurements) of our verification approach with and without using rank addressing functions. The graph without rank addressing functions is the same one shown in Figure 3.

in binary, the addressing function `0xd` (`0b01101`) can be applied similarly to the DRAM bank addressing functions to get a resulting bit determining the group.

The error rate is lower when two addresses from different banks within the same bank group are selected, so we restrict the measurements only to those cases. We compare the success rates with or without applying the additional DRAM addressing function. The result of this experiment is shown in Figure 5.

When additional rank addressing functions are used, the success rate is higher in the $700 \leq t_T \leq 780$ range. For $t_T \leq 700$, the success rate is nearly 0 for both graphs. When $t_T \geq 780$, the success rates for both graphs are similar.

The range $700 \leq t_T \leq 780$ has a similar upper border as $t_T \leq 795$, as discussed in case *E3*, in which both timings (t_c and t_h) are higher than the threshold. So, the number of row hits misclassified as row conflict is lower when rank addressing functions are used.

Some row hits are faster than others, as shown in Figure 4b. However, in contrast to the patterns in the heatmaps that occurred in the range $920 \leq t_T \leq 980$ as well, the differences in the success rate graph are only relevant in the range of row hits being misclassified as row conflicts.

The memory controller issues *Rank Select* commands every time the rank changes, so it is assumed that accessing two addresses from the same rank that are on different banks (e. g., row hit) is faster than accessing two addresses from different ranks (e. g., row hit). This effect occurs only on systems with multiple ranks (we tested systems with 1 and 2), as further evaluated in Section 5.2. Therefore, we conclude that the effect is related to the DRAM rank.

5.2 Experimental Evaluation

We perform the steps described in Section 5.1 for experimental evaluation. If a DRAM rank addressing function is derived, it is submitted to the verification tool. The maximum distance between the success rates with and without using the rank addressing function is measured. The results are shown in Table 5.

On both DDR3 systems (*S301* and *S302*), it was not possible to see any patterns in the number of row hits that were too slow, similar to the one shown

Table 5: Success rate at the threshold (t_T) with a maximum difference with and without DRAM rank addressing functions (RF) yielding the best results in Section 4.2 (10 measurement average). No rank functions found on other systems.

System	RF	t_T	% w RF	% w/o RF	Δ_{RF}
S402	`2`	540	66.16 %	38.77 %	27.40 %
S404	`13`	730	38.54 %	20.18 %	18.36 %
S501	`6,8,10`	255	77.32 %	42.86 %	34.46 %
S502	`6,8,10`	230	55.63 %	19.52 %	36.11 %

in Figure 4b. Therefore, we could not apply a rank addressing function and skipped the evaluation of the rank addressing function.

For the DDR4 systems, patterns occurred only on systems with two ranks (*S402* and *S404*), there were no recognizable patterns for the systems with one rank (*S401* and *S403*). Therefore, we evaluate rank addressing functions only on *S402* and *S404*. On *S402*, we identify the rank addressing function `2`. We observe a maximum difference of 27.40 %. On *S404*, the rank addressing function is `13`. The same data as in Section 5.1 was used for the evaluation. The maximum distance of 18.36 % was reached at a threshold of 730.

As the DRAM addressing functions on *S503* and *S504* reached very low success rates in our experiments (see Section 4.2), these systems were excluded from the evaluation of rank addressing functions. In contrast to the DDR4 systems discussed before, *S501* and *S502* had correct DRAM addressing functions. Recognizing a pattern in the number of row hits that were too slow was also possible. From this pattern, we derived the rank addressing functions *6,8,10* for both systems. On *S501*, the maximum difference between using and not using the rank addressing functions is 34.46 %. For *S502* it is 36.11 %.

The row-conflict side-channel can be used to detect rank functions equivalent to bank functions, which is the case when only a single bit is set in the rank function mask (e. g., `2`, `8`). However, most rank function masks (e. g., `13`, `6`, `10`) have multiple bits set, e. g., combine more than one bank addressing function. Thereby, the row-conflict side-channel itself is not sufficient to detect them.

6 Row-Conflict Covert Channel on DDR5

This section presents the first row-conflict covert channel on DDR5, including cross-VM, with transmission speeds up to 2.23 Mbit/s. The sender and receiver have *no shared memory* and encode data into same-bank row conflicts, similar to previous covert channel designs: A low timing (absence of a row conflict) corresponds to a '1' and a high timing (row conflict) corresponds to a '0'.

We use a time-sliced protocol, synchronized via `rdtsc` and a 75 %-majority vote to decide whether the accesses within a time slice were mainly row conflicts or not, i. e., a '0'-bit or not. For cross-VM, the Time Stamp Counter (TSC) can have different values but run at the same speed in each VM. Hence, we synchronize by transmitting a predefined sequence at a predefined rate. The

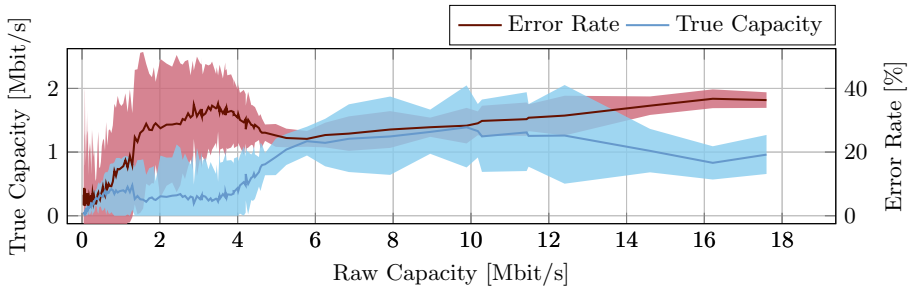


Fig. 6: Raw and true capacity of our covert channel on *S502* with 99 % confidence intervals.

Table 6: True capacity of the covert channel on multiple systems where it worked.

System	Error Rate	Raw Capacity	True Capacity	System	Error Rate	Raw Capacity	True Capacity
S301	39.22 %	0.27 Mbit/s	0.01 Mbit/s	*S402*	24.01 %	3.21 Mbit/s	0.66 Mbit/s
S302	21.74 %	9.14 Mbit/s	2.23 Mbit/s	*S501*	25.88 %	7.39 Mbit/s	1.29 Mbit/s
S401	43.66 %	3.72 Mbit/s	0.16 Mbit/s	*S502*	28.30 %	9.89 Mbit/s	1.39 Mbit/s

receiver reads the sequence and can adjust its offset to the timestamp counter. After 8 bits were received, the receiver increases the offset by $\frac{1}{20}$ of the specified transmission window.

Suppose the receiver encounters a byte that is either `10101010`, i. e., `0xaa`, or `01010101`, i. e., `0x55` (one bit shifted); the current offset is stored for the first valid sequence. Likewise, the offset of the first following sequence that is neither `0xaa` nor `0x55` is stored for the following invalid sequence. Afterward, the average of both offset values is used as offset during the rest of the transmission.

Next, it is required to synchronize the border of bytes. This is done by transmitting two bytes of `0x00`. Because the sequence `10101010` ends with a `0`, the receiver should receive $1 + 8 \times 2 = 17$ zeroes in a row. After receiving 17 zeroes in a row, the receiver starts to receive a new byte. Finally, the byte `0xaa` is sent again to verify that the synchronization was successful.

Experimental Evaluation. We transmit 6 000 randomly generated bytes. The raw capacity, i. e., the number of bits sent divided by the transmission time, and the error rate, i. e., bit-edit distance divided by the number of bits, is in Figure 6.[5] We tested 25 gradually decreasing window sizes per system. The true capacity varies slightly with the error rate.

We perform experiments on multiple test systems and calculate the true capacity. Table 6 shows the raw capacity, the error rate, and the true capacity we reach. The covert channel did not work on the systems with AMD CPUs (*S403*, *S404*, *S503*, *S504*). The reason might be that Jattke et al. [12] found that AMD requires offsets for specific physical addresses, which was not considered.

[5] Shannon's noisy-channel coding theorem yields the true capacity T as $T = r \cdot (1 + ((1-p) \cdot \log_2(1-p) + p \cdot \log_2(p)))$.

On the Intel systems with DDR3, the true capacity differs significantly. This can be explained by the error rate on *S301*, which is significantly higher than on *S302*, even though the raw capacity is significantly lower. The error rate on *S301* was even higher at higher capacities. Therefore, the true capacity of 0.01 Mbit/s was reached at a raw capacity of 0.27 Mibit/s. In contrast, the true capacity on *S302* is 2.23 Mbit/s at a raw capacity of 9.14 Mbit/s.

On *S401*, the error rate of 43.66 % is significantly higher than 24.01 % on *S402*. Even though a raw capacity of 3.72 Mbit/s and 3.21 Mbit/s is close, the big difference in the error rates leads to a significantly different true capacity. Therefore, the true capacity is 0.16 Mbit/s on *S401* and 0.66 Mbit/s on *S402*.

In contrast to the previous experiments, the true capacity of both DDR5 systems is similar. On *S501*, the true capacity is 1.29 Mbit/s at a raw capacity of 7.39 Mbit/s with an error rate of 25.88 %. On *S502*, the true capacity is 1.39 Mbit/s at a raw capacity of 9.89 Mbit/s with an error rate of 28.30 %.

7 Website Fingerprinting Attack

We utilize the DRAMA side channel to mount a website fingerprinting attack. We measure memory access patterns while accessing websites using Firefox. We hypothesize that the browser's memory access patterns depend on the website rendered at that moment. For evaluation, we train a machine learning (ML) model to classify the websites based on measured memory accesses. We verify our fingerprinting approach by classifying 100 websites with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

Fingerprinting Procedure. First, we spawn a process that measures the access times to addresses on different system memory banks. At the same time, we access a website with Firefox and wait 8 s for the website to be rendered. We then stop Firefox and the measuring process, aggregate the data to reduce data size, and prepare them for ML model training or evaluation. Afterward, we use the aggregated data to train our ML model. Finally, we reaccessed the websites, measured and aggregated memory access data, and used our ML model to predict which website we had accessed.

Access Time Measurements. We take DRAM addressing functions, allocate two 2 MiB hugepages on the system. Then, we resolve the mapping of the virtual addresses to physical addresses using `/proc/self/pagemap`. Pagemap is **unnecessary** for an actual attack because Heckel et al. [8] showed that we can dynamically group addresses based on access times. However, for the purpose of demonstration, we used it to reduce the initialization time and increase the stability for the experimental evaluation. We then start n threads for measuring. Each thread measures the memory access times of a specific DRAM bank. If the measured access time was bigger than the threshold between row hit and row conflict, the access time and timestamp are stored in a buffer. We measure the loading of each website for approximately 8 s. The number of threads n is set to $n_{\text{proc}} - 2$, where n_{proc} is the number of logical CPU cores in the system. Hence, there are still two CPU cores left for Firefox and system.

Aggregation of Data. Next, we take the files created in the previous step. We specify a window size and aggregate the number of row conflicts in that window. We then store the data in a three-dimensional array. We use a window size of 100 µs. The first dimension contains the number of row conflicts within the specified window. The second dimension contains the banks, e. g., one first-dimension list for each bank measured. The third dimension contains multiple measurements; in our case, 100 accesses the same website.

Description of the ML Model. Our ML model consists of 9 convolutional layers in groups of three with max pooling and dropout layers in between. The output of the convolutional layers is then flattened, and the final prediction is made after three dense layers. The input of a single website to the model is a 3 dimensional spectrogram with the dimensions time, frequency, and DRAM bank.

Experimental Evaluation. For experimental evaluation, we access 100 websites 100 times each. Afterward, we use 80 % of the measurements to train our ML model and 20 % to test our model. On a test system with DDR5 (*S502*), we reach an overall F_1 score of 74 % and plot the predictions of our model in Figure 7b. On a test system with DDR4 (*S401*), we reach an overall F_1 score of 84 % and plot the predictions of our model in Figure 7a. *S502* has 24 logical cores (22 threads for measurement) and *S401* has 20 logical cores (18 threads for measurement). Because each thread measures a single DRAM bank, we can measure 16 of 16 banks on *S401*, so we measure accesses of Firefox to all DRAM banks. On *S502*, we can only measure 22 of 64 banks, so 34.38 % of the DRAM accesses performed by Firefox (assuming equal distribution of accesses over all banks). We hypothesize that this is the reason for the lower accuracy on *S502*.

8 Conclusion

In this paper, we introduced a novel approach to reliably verifying DRAM addressing functions and function components from software. A first systematic analysis of 5 DRAM function reverse-engineering tools on 10 different system configurations showed significant variance in the success rate of these tools, from 0 % to 92.9 %. We discovered the previously unknown rank selection side channel and reverse engineer its function on two DDR4 and two DDR5 systems. These results enable novel DDR5 row-conflict side-channel attacks, which we demonstrated in two scenarios: a covert channel with 1.39 Mbit/s, and a website fingerprinting attack with an F_1 score of 84 % on DDR4 and 74 % on DDR5. We conclude that as reverse-engineering of DRAM address functions remains relevant, our new verification methodology provides a cheap and reliable alternative to verification using expensive physical measurements.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft under grant number 503876675 and the Austrian Science Fund under grant number 10.55776/I6054, as well as the European Union under grant number ROF-SG20-3066-3-2-2.

This preprint has not undergone any post-submission improvements or corrections. The version of Record of this contribution will be published in the proceedings *Computer Security – ESORICS 2025*

References

[1] AMD. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors*. 2015. URL.

[2] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. "Software-only reverse engineering of physical DRAM mappings for rowhammer attacks". In: *International Verification and Security Workshop (IVSW)*. 2018.

[3] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. "There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack". In: *ISCA*. 2022.

[4] Pietro Frigo, Emanuele Vannacci, 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: *S&P*. 2020.

[5] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. "Another Flip in the Wall of Rowhammer Defenses". In: *S&P*. 2018.

[6] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA*. 2016.

[7] Youngkwang Han and John Kim. "A Novel Covert Channel Attack Using Memory Encryption Engine Cache". In: *DAC*. 2019.

[8] Martin Heckel and Florian Adamsky. "Reverse-Engineering Bank Addressing Functions on AMD CPUs". In: *DRAMSec Workshop*. 2023.

[9] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. "Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters". In: *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2020.

[10] Suman Jana and Vitaly Shmatikov. "Memento: Learning Secrets from Process Footprints". In: *S&P*. 2012.

[11] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. "BLACKSMITH: Rowhammering in the Frequency Domain". In: *S&P*. 2021.

[12] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms". In: *USENIX Security*. 2024.

[13] Jonas Juffinger, Fabian Rauscher, Giuseppe La Manna, and Daniel Gruss. "Secret Spilling Drive: Leaking User Behavior through SSD Contention". In: *NDSS*. 2025.

[14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors". In: *ISCA*. 2014.

[15] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. "Half-Double: Hammering From the Next Row Over". In: *USENIX Security*. 2022.

[16] Ajaykumar Kushwaha, Ajay Jain, Mahendra Patel, and Biswabandan Panda. "Golmaal: Thanks to the Secure TimeCache for a Faster DRAM Covert Channel". In: *DRAMSec*. 2022.

[17] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. "RAMBleed: Reading Bits in Memory Without Accessing Them". In: *S&P*. 2020.

[18] Butler W Lampson. "A note on the confinement problem". In: *Communications of the ACM* (1973).

[19] Lukas Maar, Jonas Juffinger, Thomas Steinbauer, Daniel Gruss, and Stefan Mangard. "KernelSnitch: Side-Channel Attacks on Kernel Data Structures". In: *NDSS*. 2025.

[20] Michele Marazzi and Kaveh Razavi. "RISC-H: Rowhammer Attacks on RISC-V". In: *DRAMSec Workshop*. 2024.

[21] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. "Reverse Engineering Intel Complex Addressing Using Performance Counters". In: *RAID*. 2015.

[22] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. "C5: Cross-Cores Cache Covert Channel". In: *DIMVA*. 2015.

[23] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud". In: *NDSS*. 2017.

[24] Thomas Moscibroda and Onur Mutlu. *Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems*. Tech. rep. Feb. 2007.

[25] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. "Reducing Memory Interference in Multi-Core Systems via Application-Aware Memory Channel Partitioning". In: *MICRO*. 2011. DOI: 10.1145/2155620.2155664.

[26] Keisuke Okamura and Yoshihiro Oyama. "Load-Based Covert Channels between Xen Virtual Machines". In: *Symposium on Applied Computing (SAC)*. 2010.

[27] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security*. 2016.

[28] Antoon Purnal and Ingrid Verbauwhede. "Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache". In: *arXiv:1908.03383* (2019).

[29] Yi Qin and Chuan Yue. "Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7". In: *TrustCom/BigDataSE*. 2018.

[30] Fabian Rauscher, Andreas Kogler, Jonas Juffinger, and Daniel Gruss. "IdleLeak: Exploiting Idle State Side Effects for Information Leakage". In: *NDSS*. 2024.

[31] Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. "Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion". In: *ASPLOS*. 2021.

[32] Mark Seaborn. *Exploiting the DRAM rowhammer bug to gain kernel privileges*. 2015. URL.

[33] Mark Seaborn. *How physical addresses map to rows and banks in DRAM*. 2015. URL.

[34] Mark Seaborn and Thomas Dullien. "Exploiting the DRAM Rowhammer bug to gain kernel privileges". In: *Black Hat USA*. 2015.

[35] Benjamin Semal, Konstantinos Markantonakis, Raja Naeem Akram, and Jan Kalbantner. "Leaky Controller: Cross-Vm Memory Controller Covert Channel On Multi-Core Systems". In: *ICT Systems Security and Privacy Protection (SEC)*. 2020.

[36] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. "Robust Website Fingerprinting Through The Cache Occupancy Channel". In: *USENIX Security*. 2019.

[37] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. "Exploiting data-usage statistics for website fingerprinting attacks on Android". In: *ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 2016.

[38] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement". In: *ASPLOS*. 2010. DOI: 10.1145/1735970.1736045.

[39] Victor van der Veen and Ben Gras. "DramaQueen: Revisiting Side Channels in DRAM". In: *DRAMSec*. 2023.

[40] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. "Dramdig: A Knowledge-assisted Tool to UncoverDRAM Address Mapping". In: *Design Automation Conference (DAC)*. 2020.

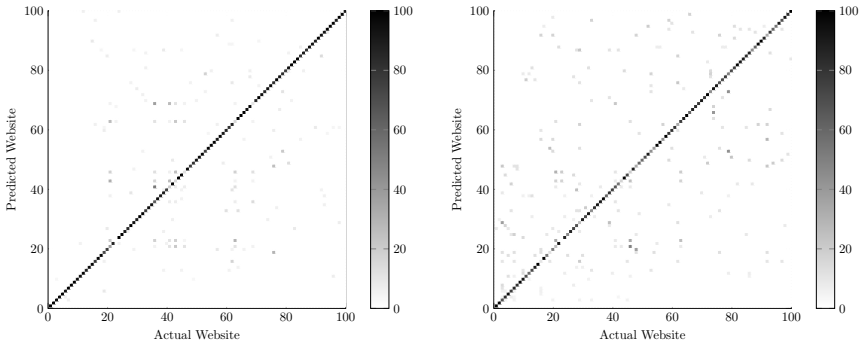
[41] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX". In: *CCS*. 2017.

[42] Zhenghong Wang and Ruby B Lee. "Covert and Side Channels due to Processor Architecture". In: *ACSAC*. 2006.

[43] Zhenyu Wu, Zhang Xu, and Haining Wang. "Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud". In: *ACM Transactions on Networking* (2014).

[44] Zhenyu Wu, Zhang Xu, and Haining Wang. “Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud”. In: *USENIX Security*. 2012.
[45] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. “An exploration of L2 cache covert channels in virtualized environments”. In: *CCSW*. 2011.
[46] Yuval Yarom and Katrina Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security*. 2014.
[47] Ruiyi Zhang, Taehyun Kim, Daniel Weber, and Michael Schwarz. “(M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels”. In: *USENIX Security*. 2023.
[48] Zhi Zhang, Wei He, Yueqiang Cheng, Wenhao Wang, Yansong Gao, Minghua Wang, Kang Li, Surya Nepal, and Yang Xiang. “BitMine: An end-to-end tool for detecting rowhammer vulnerability”. In: *TIFS* 16 (2021), pp. 5167–5181.

A Confusion Matrix Website Fingerprinting



(a) Website confusion matrix on DDR4 (*S401*), with F_1 score 86.7 %.

(b) Website confusion matrix on DDR5 on *S502*, with F_1 score 74.0 %.

Fig. 7: Confusion matrices. The actual website is shown on the x axis, the website predicted by the model is shown on the y axis.