

DREAMS: Reliable DRAM Bank Addressing Function Recovery in Asymmetric Channel Setups

Carina Fiedler[1], Fabian Rauscher[1], Markus Pfeifenberger[1]
Martin Heckel[2], Li-Chung Chiang[1], Florian Adamsky[2], Daniel Gruss[1]
[1] Graz University of Technology, [2] Hof University of Applied Sciences
{firstname.lastname}@tugraz.at, markus.pfeifenberger@student.tugraz.at, {martin.heckel.2, florian.adamsky}@hof-university.de

***Abstract*—Rowhammer attacks and DRAM side channel attacks require addresses that map to the same DRAM bank. Such addresses can be found using undocumented, reverse-engineered DRAM addressing functions. A recent large-scale Rowhammer study showed that existing reverse-engineering tools fail in about 50 % of real-world setups. These tools assume conventional setups with a power-of-two number of identical power-of-two-sized DIMMs (PTID) but fail on asymmetric non-PTID setups.**

In this paper, we discover that many non-PTID setups follow previously unknown conditional DRAM mappings, where different parts of memory obey different locally valid linear function sets. We present DREAMS, the first approach to recover DRAM bank addressing functions on non-PTID setups with such conditionally valid function sets. The key novelty is to model asymmetric mappings together with the conditions that select each function set, instead of assuming one globally valid linear function set. DREAMS detects bit-inconsistent behavior, *i.e.*, cases where the influence of an address bit is not consistent across the address space, and uses this signal to infer whether the system is PTID or non-PTID. For non-PTID setups, DREAMS recovers the region- and channel-specific function sets and their selection conditions, and verifies the resulting conditional mapping entirely in software. For overlapping functions, DREAMS uses SAT constraints to model interactions between multiple candidate bits simultaneously, allowing the analytical recovery pipeline to handle functions that share bits. We evaluate DREAMS on 19 non-PTID DIMM configurations across 6 Intel and AMD DDR4/DDR5 systems, recovering mappings with up to 3 conditional function sets selected by address cutoffs and channel functions. We show that these reverse-engineered functions provide security-critical DRAM-bank information, e.g., for Rowhammer attacks, DRAM isolation mechanisms, and side-channel attacks.

I. Introduction

Rowhammer is a reliability and security issue in modern DRAM, where bits in memory are corrupted [1]. Attackers can exploit this issue from software by accessing memory locations within a single DRAM bank at a high frequency [2, 3]. Whether memory locations map to the same bank depends on DRAM addressing functions, which select banks based on physical address bits. These functions are undocumented on most systems and can, among other things, depend on the specific CPU used, number of DRAM DIMMs, number of memory channels, and DIMM sizes installed [4]. Consequently, they have to be reverse-engineered for any attack on a new or unknown system configuration.

A recent large-scale study, reproducing Rowhammer bit flips on 12.5 % of all tested systems, showed that DRAM reverse engineering remains a practical bottleneck: existing tools fail in about 50 % of the tested real-world setups [5]. Existing approaches were developed and evaluated primarily for conventional setups with globally consistent address mappings, *i.e.*, symmetric setups with a power-of-two number of identical power-of-two sized DIMMs [4, 6–8]. This assumption becomes increasingly brittle as commodity systems combine different DIMM capacities, partial channel population, different ranks per channel, and non-power-of-two module sizes or number of channels. We learned from the authors of the study [5] that around 7 % of systems had such an asymmetric setup [1]. Cloud providers commonly use asymmetric setups[2], e.g. full DIMM population of 12 channels. Investigating within our own institution, we learned that off-the-shelf laptops are commonly sold with different-sized DIMMs. In fact, many laptops have a soldered DIMM and an additional slot, often equipped with another different-sized DIMM. Additionally, given the current state of RAM availability and pricing, we believe both consumer- as well as corporate- and cloud-compute infrastructure is getting significantly more likely to employ whichever DIMMs are available, not necessarily caring about keeping setups symmetric or optimal. Consequently, a significant class of asymmetric setups remains out of reach for existing reverse-engineering tools, limiting both Rowhammer-style security evaluations [5, 9, 10] and benign DRAM-aware uses such as bank partitioning or memory-placement optimizations [11–13].

Symmetric setups with a power-of-two number of identical power-of-two sized DIMMs (PTID) use a set of linear functions that are valid for the whole physical memory, making it an ideal scenario for reverse engineering. Outside of this ideal scenario, the CPU has to account for a wide range in DIMM numbers, types, and sizes, which could be mixed and matched

[1]However, they had the concern that there may be a selection bias since the study did not attempt to model precise market share distributions.

[2]We learned this from discussions with Cloudflare employees. Most likely other cloud providers similarly use asymmetric setups.

Network and Distributed System Security (NDSS) Symposium 2027
22–26 March 2027, Seoul, Republic of Korea
ISBN 978-1-970672-09-1
https://dx.doi.org/10.14722/ndss.2027.230177
www.ndss-symposium.org

by the system builder. We refer to these asymmetric setups as *non-PTID*. In these scenarios, using only globally valid linear functions is not sufficient to correctly select DIMM, channel, and bank. Consequently, none of the existing tools successfully recovers non-PTID mappings.

In this paper, we present **DREAMS**, DRAM Reverse Engineering for Asymmetric Memory Setups, the first approach to reliably recover DRAM bank addressing functions on non-PTID setups with certain properties. We discover that memory controllers can use previously unknown conditional DRAM mappings, where locally valid functions are selected based on address ranges or the output of global functions. This introduces non-linearity while still relying on linear functions within each region.

The key novelty of DREAMS is to model asymmetric mappings together with the conditions that select each function set, instead of assuming one globally valid linear function set. For example, in a mixed-size setup, the same address-bit difference can indicate a same-bank different-row (SBDR) relation in one address region, but not in another region where a different function set applies. DREAMS treats this bit-inconsistent behavior as a signal instead of noise: it uses it to detect that no single global function set is sufficient, to derive the address ranges or channel conditions under which each function set is valid, and to recover the corresponding function sets. Since a conditional mapping is only useful if both the functions and their selection conditions are correct, DREAMS also extends software-based verification from PTID setups to conditional non-PTID mappings. Finally, for systems with overlapping functions, DREAMS encodes the consistency constraints between measured bit tuples as a SAT problem, allowing the analytical recovery pipeline to recover functions that share address bits.

We evaluate DREAMS with 7 state-of-the-art DRAM bank function recovery tools on 15 systems with power-of-two identical DIMM configurations. DREAMS consistently produces correct functions across 10 runs on all systems with an average runtime of 22.1 s, comparable to the fastest prior runtime, while the best existing tool recovers correct functions on only 3 systems and reaches only 23.2 % average correctness. We successfully recover conditional bank addressing functions on 19 non-PTID DIMM configurations across 6 different systems. We demonstrate DREAMS's security impact with a DRAM row-buffer covert channel on a non-PTID system configuration reaching a peak true capacity of 299 ± 36 kbit/s. We make DREAMS publicly available to the research community.

In summary, we make the following main contributions:

- We present DREAMS, the first approach to reliably recover DRAM bank addressing functions on non-PTID and PTID setups, by detecting bit-inconsistent behavior, recovering region- and channel-specific function sets, and handling overlapping functions via SAT solving.
- We uncover and systematize conditional DRAM addressing on non-PTID setups, where mixed topology, mixed capacity, and non-power-of-two DIMM setups select locally valid linear function sets instead of one global mapping.

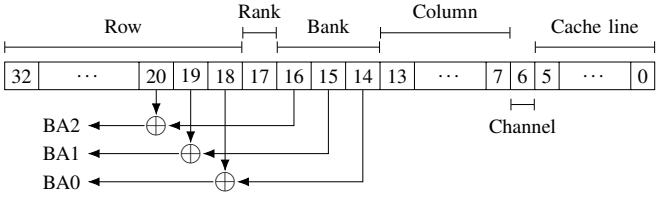


Fig. 1. Example address mapping of a physical address to a 3-bit bank number on Sandy Bridge, reverse-engineered by Seaborn [14], resulting in the following addressing functions: BA2=0x11000, BA1=0x88000, and BA0=0x44000. Usually, the addressing functions are displayed in hexadecimal, indicating which bits need to be XORed.

- We extend software-based DRAM addressing verification to non-PTID setups and evaluate DREAMS on 16 systems, comparing against 7 state-of-the-art recovery tools and reliably recovering 19 non-PTID configurations.
- We quantify the security impact of correct non-PTID mappings for Rowhammer and DRAM isolation, demonstrating a DRAM row-buffer covert channel reaching 299 ± 36 kbit/s true capacity.

II. Background and Related Work

In this section, we discuss background on DRAM and DRAM bank addressing, and related work on reverse-engineering DRAM addressing functions.

A. DRAM Organization

The fundamental unit of memory is the *cell*, consisting of a capacitor that stores a single bit of charge and a transistor that acts as a switch [21, 22]. These cells are organized into *rows* and *columns*, forming a *bank*. A DRAM chip comprises multiple banks and resides on a Dual In-line Memory Module (*DIMM*), which is divided into two *sides*, or *ranks*. To parallelize memory requests, multiple DIMMs can be distributed across different *channels*. Notably, DDR5 introduces *subchannels* directly on the DIMM [23].

During a read operation, an entire row is loaded into a *row buffer*, enabling faster subsequent accesses. However, because DRAM reads are destructive, the memory controller must write the row buffer content back before it can activate a new row. Consequently, frequent switching between rows within the same bank causes *row conflicts*, which incur higher latency than accessing rows in different banks, each having its own row buffer (*row hits*). To mitigate this, the memory controller applies *addressing functions* to physical memory addresses to scatter the seemingly contiguous physical memory across the available DRAM chips to maximize parallelism. Hence, for every DRAM cell, there is a combination of addressing functions that narrow the selection down to the corresponding memory controller, channel, DIMM, rank, bank, row, column and thereby the single capacitor storing a single bit. Mapping schemes vary across specific hardware and hardware configurations and they are usually not publicly documented. While parallelism is beneficial for performance, observable contention in parallelized systems can be exploited by attackers to mount attacks such as side-channel attacks [4].

TABLE I
LIMITATIONS OF STATE-OF-THE-ART DRAM FUNCTION RECOVERY WORKS.
●= SUCCESS. ◕= STRONG PARTIAL SUPPORT. ◐= PARTIAL SUPPORT. ○= FAILURE.

Tool	Non-PTID Setups	PCI Reclaim Offset	Reliable Threshold Detection	No Pure Row Bits	Pure Bank Functions	Function Overlap
DRAMA [4]	○	○	◐	●	●	●
Xiao et al. [6]	○	○	○	○	○	○
DRAMDig [7]	○	○	◐	○	●	●
Helm et al. [8]	○	○	●	●	●	●
Frigo et al. [15]	○	○	◐	○	●	●
AMDRE [16]	○	○	◐	●	●	●
Marazzi et al. [17]	○	○	◐	●	●	●
DARE [9]	○	◐[a]	◐	●	●	●
Knock-Knock[18]	○	○	◐	●	●	●
Chen et al. [19]	○	○	◐	●	●	○
Sudoku [20]	○	○	◐	●	●	●
DREAMS (ours)	◕	●	◕	●	●	●

[a] While their full approach to determine the PCI reclaim offset is still valid on modern machines, Intel Xeons do not follow the assumption about Intel devices and the read from `/proc/iomem` does not yield the correct offset.

B. DRAM Bank Addressing

Addressing functions can be represented as 3 components: a function to map a physical address to a specific row index (row addressing function), a function to map a physical address to a specific column (column addressing function), and a function to map a physical address to a specific bank (bank addressing function). Prior work exploited row conflicts, where two memory accesses hit the same bank but different rows (SBDR), as a side channel [4], and a high frequency of row conflicts as a means to trigger the Rowhammer effect as a fault attack [1, 2]. All of these attacks rely on the ability to determine which addresses map to the same bank, which is determined by the bank's addressing functions. A bank addressing function is a bit mask that is applied to the physical address. The masked bits are XOR-ed to produce one output bit of the bank index. Addressing n distinct banks requires $\log_2(n)$ separate linear addressing functions over $GF(2)$ (one for each bit of the bank index). Two addresses map to the same bank if they agree in the bank index. Figure 1 shows an example of a typical addressing schema on Sandy Bridge. For example, the bank function $0x11000$ details that bit 16 and bit 20 of the physical address are XOR-ed together to produce one bit of the bank index. A single set of linear functions is not sufficient in some non-PTID DIMM configurations, e.g., when the number of banks is not a power of two. We discuss this case in Section III.

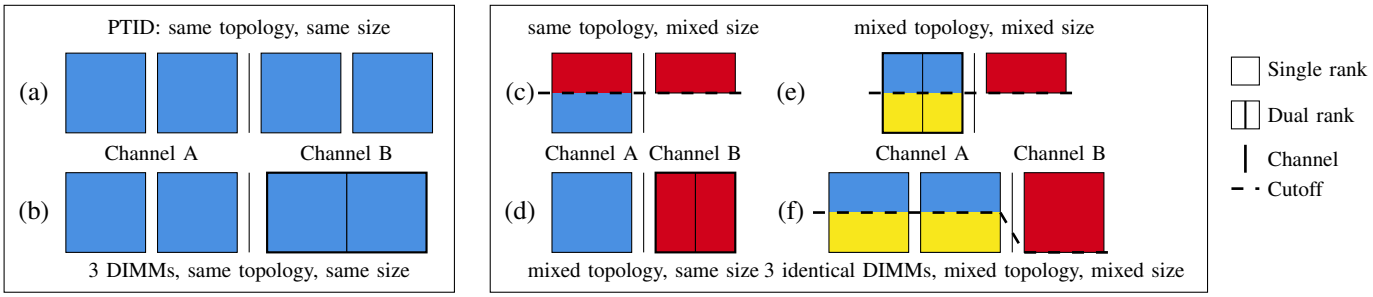
C. Reverse-Engineering DRAM Addressing functions

These DRAM addressing functions can be used for better targeted Rowhammer attacks [4, 9, 10, 24–26] but also for performance optimization [27–29]. Since Intel did not publish these addressing functions at all and AMD only until microarchitecture 16h [30], the research community proposed numerous approaches [4, 6–9, 16–20] to reverse-engineer DRAM bank addressing functions. DRAM addressing functions are often reverse-engineered via side channels, such as timing differences between row conflicts and their absence.

Seaborn and Dullien [24] were the first to use DRAM addressing functions in a security context, showing that an adversary can use these to mount double-sided Rowhammer attacks. The first approach to generically reverse-engineer DRAM addressing functions was presented only afterward by Pessl et al. [4] with DRAMA, a software-only approach validated by physical measurements on the DRAM bus. Pessl et al. [4] use a timing side channel of high-latency row conflicts to identify addresses that map to the same bank. Based on the observed timings, they grouped randomly selected addresses into bank sets. They then brute-forced linear XOR functions, accepting a candidate as valid if it partitioned the addresses into the same bank sets as the timing measurements.

Many future works built on this approach, improving reliability and runtime, and extending it to other platforms. Wang et al. [7] proposed DRAMDig, a knowledge-assisted approach for reverse-engineering DRAM addressing functions, that gathers information about the DRAM layout from software to achieve a more targeted and efficient approach. Frigo et al. [15] presented many-sided hammering, an approach to bypass the TRR mitigation deployed on DDR4 DIMMs. In their tool suite, they also included a slightly improved version of reverse-engineering method by Pessl et al. [4]. Heckel et al. [16] improved the measurement and grouping procedures in their tool AMDRE, enabling partial function recovery restricted to 2 MB huge pages on AMD systems. Jattke et al. [9] fully extended this function recovery approach to AMD systems enabling the first Rowhammer attacks on AMD devices. Their tool DARE takes a fixed reclaim offset into account, which is determined via `proc/iomem`, and shifts addresses accordingly before applying DRAM addressing functions. Marazzi et al. [17] extend the methodology by Heckel et al. [16] to RISC-V CPUs.

In contrast to previous work, Helm et al. [8] used performance counters rather than timing differences. As a result, they achieved significantly lower error rates and greater stability, but at the cost of requiring performance counters, which limits their applicability. Wi et al. [20] proposed an approach for semantic DRAM addressing function reverse-engineering, e.g., detecting which specific DRAM component a function targets. They utilized two previously unknown timing side channels for this: latency induced by refresh operations and timing differences arising from consecutive memory accesses.



(a). Global function set domain

(b). Conditional function sets

Fig. 2. Function set domains in PTID and non-PTID dual-channel setups. Colors indicate distinct function set regions.

In contrast to the bucket sorting approaches, Xiao et al. [6] measured the row-conflict side channel on pairs of addresses that differ only in specific bits. They used a graph-based algorithm to identify and analyze the impact of individual row and bank bits to recover simple addressing functions. Chen et al. [19] extend the approach by Xiao et al. [6] to systems without pure row bits, bits that only impact the row index but not the bank selection. First, they identify all row bits via pairwise testing, then they identify the remaining bank bits and finally the bank functions by toggling multiple bits at once. Plin et al. [18] utilized a linear algebraic model over the finite field GF(2) to identify DRAM addressing functions. They reached a significant speedup compared to previous approaches, reverse-engineering addressing functions on a system with 500 GB of DRAM in a few minutes.

Heckel et al. [5] demonstrated in a large-scale Rowhammer study on more than 800 devices that a failure to reverse-engineer correct addressing functions is one of the largest factors hindering Rowhammer attacks in practice. Given more reliable reverse-engineering, they extrapolated that the number of affected devices in the wild might even double from 12.5 % to 25 %. In fact, more than 10 % of devices in their data set had memory setups that have not been discussed in any prior work on DRAM function reverse-engineering.

Table I shows an overview of the limitations of state-of-the-art DRAM function recovery methods. We categorize their failure reasons into five categories: **Non-PTID Setups.** Prior approaches were designed for setups with a power-of-two number of identical power-of-two sized DIMMs (PTID). Therefore, all prior tools fail on non-PTID setups, e.g., a non-power-of-two DIMMs, that do not result in linear mapping functions that are valid throughout the entire physical address space. **Address Space Offsets.** While Jattke et al. [9] identified simple PCI reclaim offsets as a source of non-linearity, many tools do not consider these constant offsets in practice. This again leads to a failure to reverse-engineer correct functions on many systems, as a simple address offset breaks GF(2)-linearity. **Threshold Determination.** Most approaches rely on a threshold to distinguish row hits from row conflicts, which can be difficult to determine and can lead to incorrect function recovery if the threshold is not set correctly. Most tools try to automatically determine the threshold based on the timing distributions of a large number of random addresses, but this is not always reliable. Other tools, such as Xiao et al. [6], rely on hardcoded thresholds, which are also not suitable for automated end-to-end Rowhammer attacks and can easily be misconfigured. Heckel et al. [5] automated this step for multiple tools, which again may suffer from reliability issues. **Non-existence of Pure Row Bits.** DRAMDig [7] implicitly assumes the existence of pure row bits in the physical address, which are bits that only influence the row index and not the bank index. Their physical address selection algorithm uses the determined pure row bits to reduce the search space. However, their implementation does not simply fall back on random address selection in the entire address space in the absence of pure row bits but rather fails entirely [19]. **Overlapping Functions.** While Chen et al. [19] address the issues of the absence of pure row bits and the presence of pure bank functions in Xiao et al. [6], we found that their approach fails to recover function sets where bits occur in multiple bank functions (see Section VII).

While Gerlach et al. [31] proposed an approach to reverse-engineer non-linear microarchitectural hash functions and their approach should be applicable to DRAM functions as well, they only reverse-engineered linear DRAM addressing functions, not non-linear ones, potentially as such non-linear functions were not encountered in any memory setup.[3] Hence, currently, there is no approach to reverse-engineer DRAM addressing functions that do not follow the idealized PTID model assumed in prior tools, which is a valid assumption for most systems but is not for a still significant number of systems in the wild, as evidenced by the data sets by Heckel et al. [5].

In this work, we also focus on reverse-engineering bank addressing functions, as they are the most relevant for attacks and the most complex to reverse-engineer. In particular, we tackle the remaining open problem of reverse engineering the bank addressing function in non-PTID setups that cannot be addressed via a single set of linear mapping functions.

[3]We also did not encounter any such systems that directly use non-linear functions, leading to our hypothesis that commodity-off-the-shelf systems do not use non-linear functions directly, but rather use linear functions in a non-linear way, e.g., by applying them conditionally.

III. DRAM Address Functions in Non-PTID Setups

In this section, we present our insights on how DRAM functions index non-PTID dual-channel setups, where each channel contains a power-of-two number of DIMMs of identical size and rank. We categorize setups based on whether both channels contain the same total amount of memory (*same size*) or double the amount of memory (*mixed size*) and whether the combined number of ranks and banks in the channels agree (*same topology*) or not (*mixed topology*). Same-size and same-topology setups can be addressed with a single set of linear DRAM addressing functions that holds globally for all addresses (see Figure 2a). In all other setups, different sets of addressing functions hold for distinct sets of physical addresses (see Figure 2b). Systems with mixed-size setups utilize an address cutoff to choose between address function sets. For mixed-topology setups the channel function chooses between different address function sets. We present full details of our findings in Section VI. More complex setups cannot be modelled by these binary selection conditions across channel and/or region cutoff. Hence, size or topology differences beyond factor 2, non-power-of-two number of channels, non-power-of-two sized DIMMs and different DIMMs within channels remain outside of the scope of this paper. We hypothesize that our methodology could be extended to iteratively identify nested selection conditions and linear addressing function sets in some of these more complex setups (see Section IX).

IV. DREAMS

In this section, we present DREAMS, a reliable DRAM recovery approach in both PTID and non-PTID DIMM configurations. First, we explain our generic approach to recover linear addressing function sets including improved threshold selection, an extension to the same-bank different-row (SBDR) test to avoid misclassification and a SAT solver approach to correctly handle function overlap. Then, we outline the steps to extend our approach to non-PTID setups based on bit-inconsistent behavior detection and address range restrictions.

A. Recovering Linear Functions

DREAMS utilizes the known row conflict timing side channel [4] of existing tools, but replaces the exponential function brute-forcing with an efficient bottom-up algorithm that identifies the role of individual bits or bit tuples, following the line of work of Xiao et al. [6] and Chen et al. [19]. The novelty of this linear recovery approach is the combination of structured address-pair selection, automatic pure-row-mask construction, and SAT-based merging of bit groups, which allows DREAMS to recover overlapping linear functions reliably without falling back to exhaustive search. DREAMS is structured as follows:

Threshold Detection. To exploit the row-conflict timing side channel, it is necessary to determine a threshold distinguishing row conflicts from row hits. Prior works determine this threshold by measuring the access times of a large number of randomly selected address pairs, *i.e.*, probabilistically both fast row hits and slow row conflicts. Since modern systems can have hundreds of banks, ensuring a high probability of observing at least one row conflict $1-(1-\frac{1}{\#banks})^n$ may require measuring thousands of addresses. Even then, it is difficult to distinguish a distribution with higher execution time from random noise-induced delays.

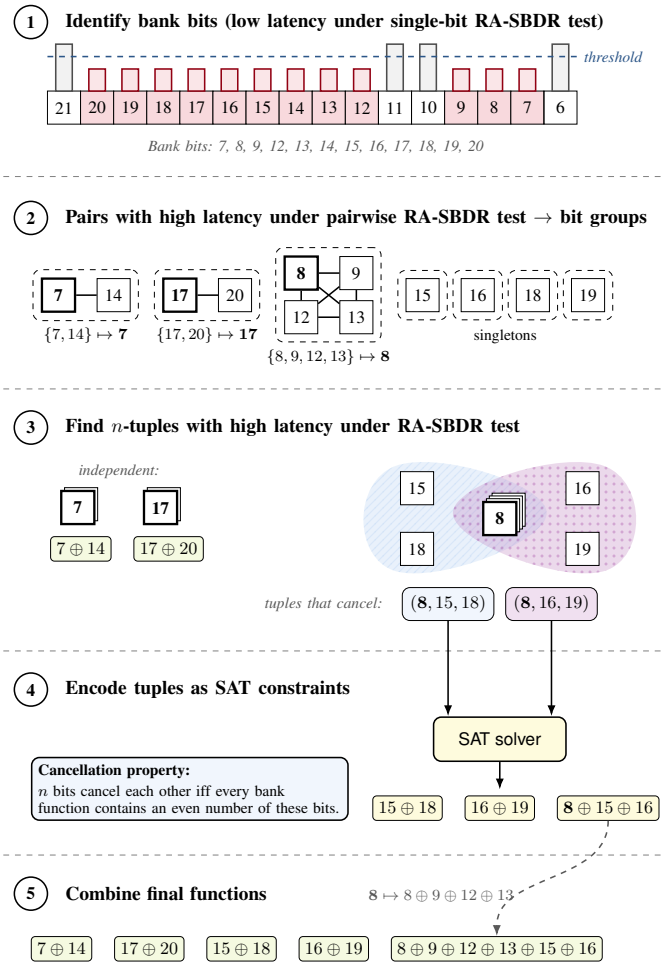
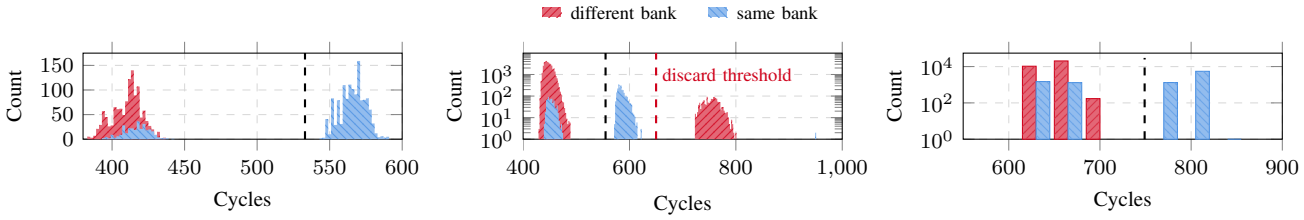


Fig. 3. Overview of DREAMS. We analyze bit influence via timing measurements on address pairs that differ in $n = 1, 2, 3, 4$ bits. Based on this we construct bank bit groups and resolve dependencies between bit groups via a SAT solver.

To minimize the number of addresses we need to test while still guaranteeing the observation of row conflicts, we select addresses strategically. Some architectures use (typically higher) address bits as *pure row bits*, meaning that they impact the row selection but not bank selection. On these systems, measuring an address pair that differs in a single bit position n for all $n \in [6, \log_2(\text{TOM})]$ where TOM denotes the total amount of memory, guarantees the observation of some row conflicts. To be more precise, we observe row conflicts for all bit positions that are pure row bits. On some newer Intel and AMD architectures, all bits that affect the row index are also used in bank-addressing functions, *i.e.*, there are no pure row bits. Hence, the access time of address pairs differing by a single bit always indicates a row hit. To ensure row conflicts on these systems, we need to test address pairs that differ in two bits. We can observe row conflicts for an address pair



(a) S02 (only single-bit) (b) S05 (c) S11

Fig. 4. Median of address pairs with single- and double-bit differences. Color indicates classification as same bank or different bank according to the recovered bank functions. Note that the same bank category contains both row hits and row conflicts. The dashed line marks a suitable threshold.

that differs in exactly two bits iff every bank function contains either none or both bits, and at least one of the bits impacts the row index. We refer to these pairs as *pure row pairs*, based on the concept of "borrowing an inital SBDR state" by Chen et al. [19]. While each bit individually impacts the bank selection, if both are toggled, the bank selection remains unaltered.

DREAMS automatically determines if the system uses pure row bits by testing address pairs that differ in each single bit position $n \in [6, \log_2(\text{TOM})]$ based on the approach by Xiao et al. [6]. We measure several address pairs for each bit position, taking the median over repeated measurements to improve noise resistance. If we observe two distinct timing levels, we identified the threshold. Otherwise, we repeat the measurements for bit position pairs $(n, m) \in [6, \log_2(\text{TOM})]^2$. Besides guaranteeing row conflicts, grouping address pairs by bit differences allows us to take the median across several address pairs, since either all or no address pairs with the same bit difference must cause row conflicts on PTID setups, thereby significantly improving stability. However, our structured address selection removes the probabilistic row-conflict vs row-hit ratio of random addresses, which prior work uses to detect the correctly sized row-conflict distribution. While our method makes it significantly easier to detect distribution valleys, choosing the correct valley and orientation of the threshold towards the edges of the valley remains non-trivial.

Figure 4 shows the resulting median timings for bit differences on three test systems. Note that the blue same bank category includes row hits on the same bank as well as row conflicts. Since Intel Core i9-9900K (S02) has pure row bits, we observe a large fraction of high-latency row conflicts. A large margin clearly separates the row-hit and row-conflict distributions. On the other two systems without pure row bits (log-scaled), we observe lower peaks for high-latency row conflicts compared to the peaks of different bank row hits. However, the blue distributions for row hits on the same bank and row conflicts show similar peaks, indicating that our address selection approximates an equal distribution across these categories. On Intel Core Ultra 7 155H (S05), we observe a significant number of high-latency row hits on different banks. We cannot determine the root cause based on these timing traces alone, but one plausible explanation is occasional interference by platform- or DRAM-internal reliability mechanisms. For example, Rowhammer mitigations such as TRR and DDR5 PRAC/RFM can introduce additional tracking or refresh activity, and prior work has shown that defense-induced latency is large enough to form covert and side channels [32]. Related DRAM timing work also shows that refresh-induced latency is observable from software [20]. Similarly, ECCploit exploits the timing difference introduced by ECC correction [33], and correction, retry, or other memory-controller mechanisms could manifest as high-latency outliers even for address pairs that do not map to the same bank. This motivates the introduction of an additional noise threshold above the main row-conflict distribution to discard these measurements.

Construct Pure Row Mask. The recorded timings from the threshold detection allows us to identify pure row bits or pure row pairs based on timings above the threshold. We collect these bits in a *pure row mask* that allows us to create an SBDR state for all remaining measurements in the algorithm. In the presence of pure row bits, the pure row mask simply contains all identified pure row bits. With pure row pairs, we need to be more selective. We define a *(bank) bit group* as a set of bits that occur in exactly the same bank addressing functions. Since pure row pairs appear in precisely the same bank addressing functions, we group the corresponding bits into the same bit group. Pairs with shared bits are added to the same bit group. From each resulting bit group we only add the largest possible **even** number of bits to the pure row mask, ensuring that all elements of the pure row mask cancel each other out collectively. Hence, applying the pure row mask to an arbitrary address produces an SBDR address.

Identify Bank Bits. Figure 3 visualizes all remaining steps. Timings below the threshold indicate that the corresponding bit is either a column bit, *i.e.*, not involved in neither row nor bank selection (row hit on the same bank) or a pure bank bit (row hit on a different bank). To distinguish these two cases we repeat the measurements on single bit position toggles, but toggle all bits in the pure row mask in addition. We call this a single-bit *row-assisted SBDR test* (RA-SBDR). By forcing a different row, a low timing uniquely identifies a bank bit, while column bits produce a high timing. Combining all bits in pure bit pairs and all bank bits identified in this step, we have identified all bits that impact the bank selection.

Bank Bit Groups. We assign each bit to a bank bit group based on a pair-wise RA-SBDR test of the test bit and a representative bit from the bank bit group. The representative

bit can be any bit of bank group. This means that we toggle the test bits and any bits in the pure row mask with one exception: If the representative bit appears in the pure row mask, we would toggle it twice. Hence, if this is the case we temporarily remove the representative bit and a second element from its bit group from the pure row mask. This ensures that the pure row mask still does not impact the bank selection. If we observe a high timing, the bits appear in precisely the same bank functions, and we add the bit to the bank bit group. Due to the transitivity of the XOR operation, it is sufficient to test against a single element from each bank bit group. If a bit shows low timings with all existing bit groups, it forms a new bank bit group. Identifying (pure) row bits, identifying all bank bits and finally forming bank bit groups roughly correspond to the three steps in the algorithm by Chen et al. [19].

n-Tuples. If all bank bits only occur in a single function, the bank bit groups constitute the final bank addressing functions. However, if bits appear in several functions there are dependencies between the bank bit groups that are not apparent from the pairwise testing in the previous step. Our evaluation Section VII shows that most setups are addressed with overlapping functions. Handling these cases correctly is the major distinction of our work from the methodology by Chen et al. [19]. For example, if function A contains bit 15 and 18 and function B contains bit 8 and 15, pairwise testing between either of these bits will cause fast timings. However, if we toggle all three bits (8, 15, 18) in an 3-tuple RA-SBDR test, the outputs of both functions remain unaltered resulting in a slow row conflict. We detect the dependencies between bit groups via n-tuple RA-SBDR tests on their representative bits. If we observe high timings, the tuple indicates a dependency between the bit groups, *i.e.*, single bit toggles from each bit group cancel each other in all bank functions. If we observe low timings, the corresponding bits do not cancel each other. We call these low-latency tuples *inverse tuples*. Bank bit groups that do not have any dependencies on other bit groups, *i.e.*, whose representative bit does not appear in any tuples, constitute final bank addressing functions. We refer to these bank bit groups as *independent* bank bit groups.

Merging Bank Functions. Next, we determine the minimum set of functions that satisfies the information provided by the tuples. Toggling all bits of a tuple caused a high timing, indicating that two addresses that differ in this tuple map to the same bank. Taking the example of tuple (8, 15, 18), this means that these bits need to cancel each other in all functions. If bit 8 appears in one function, either bit 15 or bit 18 also need to appear in that function to flip the output bit back to the same bank. Generalizing, all functions need to contain an even number of bits from the tuple so that they cancel each other in the XOR-operation. Conversely, addresses that differ in inverse tuples do not map to the same bank. Hence, when toggling all bits of an inverse tuple (8, 15, 16), there needs to be at least one function that changes its output bit. For example, a function that contains just bit 15, but neither bit 8 or bit 16. This is the case when a function contains an odd number of bits from the tuple.

To recover bank functions, we model our constraints in Z3 [34]. We initially restrict the number of resulting functions to 2 and increase it until we can satisfy all constraints:

- Each function is required to include an even number (possibly 0) of bits from every tuple.
- For each inverse tuple there has to be at least one function which includes an odd number of bits from this tuple.
- Each function must contain at least 1 bit.
- Each bit within a tuple must be used by at least 1 function.
- Each bit can occur in maximum 2 functions. [4]

On most systems, we can fully model the correct functions with tuples of size 3. During testing, we only found a single system that requires tuples of size 4. The functions on this system demonstrate a large amount of overlap with all functions containing bits that occur in several functions. We did not observe any systems requiring tuples larger than 4 to be able to fully define the correct functions. Despite this, we are still able to handle such systems, by collecting larger tuple sizes in case the functions are underdefined. Furthermore, testing higher-order tuples is cheap, since the number of combinations of the tuples decreases exponentially with the size.

Since, conflict-based bank function recovery does not assign an order to the banks, there are equivalent bank function sets that produce identical conflict behavior. We choose the SAT solver function assignment that uses the lowest combined bit sum [5]. Finally, we expand each occurrence of a representative bit in the SAT solver function results to its full bit group. Combined with all independent bit groups, we obtain the final bank addressing functions.

Precise Address Selection. Some existing tools, such as DARE [9], rely on 1 GiB huge pages, which are not suitable to map the entire physical address range, since they are reserved at boot time. To enable access to arbitrary physical memory addresses, we map 2 MiB pages and modify page frame numbers with PTEditor [35] to map our target addresses. 2 MiB huge pages are supported on a wider range of systems [36] and reduce the noise due to less TLB contention with standard 4 KiB pages. By modifying only the page frame numbers, we also ensure that the mapping is identical across different tested physical addresses, preventing potential timing differences due to different mappings. We incorporate the findings from [9], adding a fixed reclaim offset to all addresses above the start of the reclaim on AMD systems and, additionally, to all addresses below 4 GiB on Intel Core systems. However, we discover that Intel Xeon architectures also follow the same approach as AMD, shifting all addresses above the start of the reclaim. The simplified approach by Jattke et al. [9] to determine the reclaim offset based on the difference between the start of the primary PCI range as documented in `/proc/iomem` and the 4 GB boundary does not result in consistent functions on these machines. Instead, we find that the difference between the end of the last System RAM range and the 4 GB boundary,

[4] We did not encounter systems where bits appeared in more than two functions. If the result was UNSAT, this number could be increased.

[5] This is an arbitrary choice that preferences lower and fewer bits.

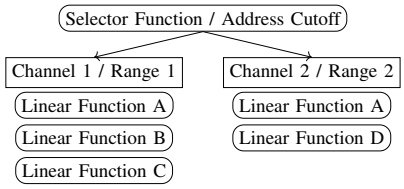


Fig. 5. For same-sized memory ranges with different topologies, a linear XOR channel function is used to choose between distinct sets of linear functions. For mixed-sized memory ranges, an address cutoff is the selection condition.

rounded down to the nearest 1 GB, works as a reclaim offset on all four Intel Xeon test systems.

B. Recovering Non-Linear Functions

The above approach can reliably recover globally valid linear addressing functions over $GF(2)$. Non-PTID setups with different amount of memory or combined number of ranks or banks per channel cannot be addressed with a single linear function set. To recover the functions in non-PTID setups, we leverage the following insight: Mixed-topology or mixed-size configurations exhibit conditions that choose between different sets of linear bank addressing functions, as shown in Figure 5. We automatically identify these conditions and recover each set of linear functions separately.

Mixed Size. We analyze the DIMM configuration with `dmidecode`. If the system contains different-sized DIMMs in each channel, a cutoff splits the DRAM address space, and the two resulting contiguous address spaces use different sets of addressing functions. We identify the size of the lower channel-interleaved address space as twice the size of the smaller DIMM. The remaining upper memory region is the second single-channel address space. In contrast to the non-linearity from the PCI reclaim shift [9], the function sets for each region cannot be converted into each other with a constant offset addition. Hence, we recover the function set of each space separately, restricting the selection of **all addresses** to a single region. Since we assume each region to be individually linearly addresseable, this limits us to combinations where the larger DIMM is double the size of the smaller DIMM.

Mixed Topology. In a mixed-topology setup, each channel uses different bank addressing functions. Since equally sized memory regions are fully channel-interleaved, the two memory regions are not contiguous. As the channel function, *i.e.*, the condition that chooses between the two function sets, is not initially known, we cannot simply split the physical address space by channel before reverse engineering the channel function itself. Instead, we initially follow the linear recovery approach, but we dynamically detect a mixed-topology setup at runtime. At first, our addresses are sampled from both channels, so each address pair can either lie entirely on channel A, entirely on channel B, or be split between the channels. With globally valid bank addressing functions, SBDR tests on address pairs that all differ in the same bits produce either consistently high or low timings.

With conditionally valid functions, we observe *bit inconsistent behavior* where the SBDR test produces high timings only for a fraction of the address pairs with the same bit differences. This can already occur for single-bit differences when a bit's role differs between channels in a way that affects bank or row indexing. It can also occur for bank bit pairs that appear in the same functions on channel A, but in different functions on channel B. The addresses of a pair that demonstrate a row conflict timing are necessarily on the same bank, and therefore on the same channel. The row conflict pairs and the row hit pairs adhere to distinct addressing functions. Hence, we partition the addresses into two sets based on whether they induce row conflicts, with each set representing one channel. In the remainder of the algorithm we recover the addressing functions for each channel separately. However, this partition only gives us seed addresses for each channel, not a complete channel predicate. Unlike the mixed-size case, we therefore cannot construct arbitrary address pairs within a channel without expensive testing, since we have not recovered the channel function yet. Instead, we relax the restriction: only **one address** in each test pair must come from the target channel set. The second address is generated through a bit toggle and may or may not be in the same channel. If the bit toggle moves the second address to the other channel, we get row hit timings, as with any other bank-bit toggle. We then recover the partial functions per channel, including the channel function, which must agree between the two function sets.

We recover the channel function before invoking the SAT solver. We first test if any individual bit group already constitutes the complete channel function by checking if it produces a constant output over a channel set. If the channel function shares bits with the partial functions, we test linear combinations of bit groups as potential channel functions. Once identified, we integrate the channel function into the SAT solver's constraints. If the complete channel function consists of a single independent bank bit group, no additional solver constraint is needed. Otherwise, we constrain the solver to produce a function with precisely these bits. If the channel function additionally includes independent bit groups, we add their representative bits as singleton constraints, forcing the solver to produce them as separate functions. We then let Z3 recover the remaining functions.

V. Verifying Conditional Function Sets

In this section, we present our approach to verify that conditional function sets recovered with DREAMS correctly identify same-bank address pairs, crucial to evaluate the quality of the reverse-engineering result. The expected outcome is a software-only test that detects both incorrect bank functions and incorrect function set selection conditions. To perform this verification entirely in software for non-PTID setups, we therefore have to verify both the functions within each set and the conditions that select which set applies to an address.

Heckel et al. [37] use SBDR tests to verify the accuracy of the bank addressing functions on PTID setups. First, they test what percentage of address pairs classified as same bank

TABLE II
TEST SYSTEMS

	ID	CPU	Architecture	DRAM	DIMMs
Intel	S01	Core i5-7200U	Kaby Lake	DDR4	8 GB 1R + 16 GB 2R
	S02	Core i9-9900K	Coffee Lake-R	DDR4	2x 8 GB 1R
	S03	Core i5-8265U	Whiskey Lake	DDR4	2x 8 GB 1R
	S04	Core i9-13900KF	Raptor Lake-S	DDR5	2x 32 GB 2R
	S05	Core Ultra 7 155H	Meteor Lake	DDR5	2x 16 GB 1R
	S06	Core Ultra 7 265K	Arrow Lake	DDR5	2x 16 GB 1R
	S07	Xeon E5-1630 v4	Broadwell-EP	DDR4	2x 8 GB 2R
	S08	Xeon 4410T	Sapphire Rapids	DDR5	4x 16 GB 1R
	S09	Xeon 4514Y	Emerald Rapids	DDR5	8x 16 GB 1R
	S10	Xeon 6507P	Granite Rapids	DDR5	8x 16 GB 1R
AMD	S11	Threadripper 1920X	Zen	DDR4	2x 8 GB 1R
	S12	Ryzen 7 7700X	Zen 4	DDR5	2x 16 GB 1R
	S13	Ryzen 9 9950X	Zen 5	DDR5	1x 16 GB 1R
	S14	Epyc 7313P	Zen 3	DDR4	1x 16 GB 2R
	S15	Epyc 8024P	Zen 4c	DDR5	2x 16 GB 1R
	S16	Epyc 9015	Zen 5	DDR5	1x 16 GB 1R

according to the test functions actually cause row conflicts. This constitutes the initial benchmark of how many row conflicts the test functions produce as a whole. They then remove individual addressing functions one at a time and recompute the accuracy. Removing a correct function halves the initial value, since each bank class doubles in size while only the original half consists of true same-bank pairs. A function is therefore only deemed correct if its removal halves the accuracy. Since this verification still relies on the row-conflict timing side channel, it provides a meaningful consistency check instead of claiming physical ground truth. Still, their method is limited to PTID setups.

Addressing functions on non-PTID setups typically have several conditional function sets, with an address cutoff or a channel function determining which function set applies to an address. This prevents directly applying the PTID verification method for three reasons: First, each function set is only valid in the address space selected by the cutoff or channel function. Second, in the mixed-topology case, removing the channel function would not only double each bank class due to a missing function, but also invalidate the partial functions by changing their applicable address region. Hence, we could not expect a clean $50\,\%$ accuracy reduction, which is incompatible with prior work. Third, the address cutoff in the mixed-size setup cannot be verified with the same single-bit precision.

We solve these challenges as follows. First, we verify the functions within each set directly. In the mixed-size case, once the cutoff condition is known, we can restrict test addresses to their valid ranges, reducing the verification problem to the linear case when restricting addresses to each range separately. In the mixed-topology case, we also consider each function set separately, restricting addresses to the corresponding channel via the recovered channel function. Following the methodology from Heckel et al. [37], we measure the execution time of random address pairs and check that pairs that agree in all bank addressing functions cause a row conflict. We then remove one function from the set and test whether the respective address pairs cause row conflicts in $50\,\%$ of cases.

For the selection condition in the mixed-topology case, we verify the channel function implicitly. If all conditional function sets verify individually when restricted to appropriate addresses via the selection condition, we infer that the selection criterion is correct. If the channel function were incorrect in a same-size mixed-rank setup, the address selection would still be uniformly distributed across both channels. The maximum number of row conflicts would be achieved if the two function sets only differed in a single function. In that case, when testing correct functions in channel A, two addresses in channel B cause a row conflict with a probability of $50\,\%$. This leads to a combined row conflict probability of only $P(C) = 0.25P(C|\text{both A}) + 0.25P(C|\text{both B}) + 0.5P(C|\text{different channel}) \leq 0.25 * 1 + 0.25 * 0.5 + 0.5 * 0 = 37.5\%$ Using this as an upper bound for incorrect channel functions, with several thousand access pairs we can distinguish this from the near 1 probability of correct addressing functions with high confidence.

TABLE III
EVALUATED NON-PTID SETUP CAPABILITIES OF DREAMS.
●= FUNCTIONS RECOVERED. ⊘= SETUP NOT AVAILABLE.

DRAM	CPU	Same Size Same Topol.	**Mixed Size** Same Topol.	Same Size **Mixed Topol.**	**Mixed Size Mixed Topol.**
DDR4	Intel	●	●	●	●
	AMD	●	●	●	●
DDR5	Intel	⊘[a]	⊘[a]	●	●
	AMD	⊘[a]	⊘[a]	●	●

[a] We did not have suitable hardware available to evaluate these setups individually. Since our findings for the combined cases on DDR5 were consistent with the evaluation on DDR4 systems, we assume DREAMS to be equally applicable to these configurations.

For the selection condition in the mixed-size case, we verify the address cutoff up to the precision relevant for random address selection. An address cutoff that is significantly different such as $8\,\text{GB}$ instead of $16\,\text{GB}$ in a $24\,\text{GB}$ setup is easily detectable. The bottom function set below $8\,\text{GB}$ would still verify correctly with this incorrect cutoff. The top region would be split into two equally sized regions: one with the correct upper region functions and the mid section where these functions are incorrect. Assuming again that the two function sets only differ in a single function, the function set above the cutoff has a row conflict probability of at most $P(C) = 0.5P(C|\text{different region}) + 0.25P(C|\text{both} < \text{true cutoff}) + 0.25P(C|\text{both} > \text{true cutoff}) \leq 0.5 * 0.5 + 0.25 * 1 + 0.25 * 0.5 = 62.5\%$. While this is clearly distinguishable with repeated measurement, a cutoff that is only slightly incorrect, would require significantly more samples to detect. However, a slight offset to the correct cutoff would only misclassify a few addresses and therefore, not play a significant role for randomly selected addresses across the entire address range. Further, in practice we can utilize the heuristic that each address range is required to be a power-of-two. Overall, this allows us to detect both incorrect functions within the conditional function sets as well as incorrect selection conditions.

VI. EVALUATION OF NON-PTID SETUPS

In this section, we present and systemize the function sets of non-PTID that we reverse-engineered with DREAMS. In our analysis, we vary the number of DIMMs, as well as the size and number of ranks and banks per DIMM and channel,

TABLE IV
ADDRESSING FUNCTIONS FOR SAME-SIZE, SAME-TOPOLOGY NON-PTID SETUPS WITH 1 FUNCTION SET ON INTEL CORE I9-9900K (S02) AND AMD THREADRIPPER 1920X (S11)

	Setup	Set
S02	2x1R(8 GB) + 1x2R(16 GB)	0x4080, 0x4b300, 0x88000, 0x110000, 0x220000, 0x440000
S11	2x1R(8 GB) + 1x2R(16 GB)	0x100, 0x7fec0, 0x111120000, 0x222207ec0, 0x88890000, 0x444408000

TABLE V
ADDRESSING FUNCTIONS FOR SAME-SIZE MIXED-TOPOLOGY NON-PTID SETUPS WITH 2 CONDITIONAL FUNCTION SETS ON INTEL CORE I9-9900K (S02), AMD THREADRIPPER 1920X (S11), INTEL XEON E5-1630 V4 (S07), INTEL CORE ULTRA 7 265K (S06), AND AMD RYZEN 9 9950X (S13)

	Setup	Channel	Set 1 [Channel = 0]	Set2 [Channel = 1]
S02	1x1R(8 GB) + 1x2R(8 GB)	0xc3300	0x4080, 0x48000, 0x90000, 0x120000	0x4080, 0x88000, 0x110000, 0x220000, 0x440000
S11	1x1R(8 GB) + 1x2R(8 GB)	0x100	0x44450000, 0x111203C0, 0x888A0000, 0x222228000	0x7FFC0, 0x44448000, 0x88890000, 0x111120000, 0x222203C0
S07	1x1R(8 GB) + 1x2R(8 GB)	0x5555080	0x200040, 0x440000, 0x880000, 0x1100000	0x4000, 0x400040, 0x880000, 0x2200000, 0x1100000
S06	1x1R(16 GB, 32 banks) + 2x1R(8 GB, 16 banks)	0x49900	0x4400, 0x8000, 0x10000, 0x22600, 0xa0000, 0x140000	0x4400, 0x8000, 0x50000, 0x82600, 0x120000, 0x2080000
S13	1x1R(16 GB, 32 banks) + 2x1R(8 GB, 16 banks)	0x100	0x3e40, 0x42101000, 0x84200200, 0x108400400, 0x210802000, 0x421080800	0x1e40, 0x40000, 0x88880800, 0x111101000, 0x222200200, 0x444400400

while focusing on configurations with identical DIMMs within each effective channel. Each channel contains either the same or double the amount of memory and either the same or double the number of banks and ranks. Our approach relies on linear function selectors and address cutoff conditions. While our evaluation is restricted to dual-channel setups, our approach should extend to power-of-two channel counts (e.g., 4,8) without changing the underlying model. In contrast, an effective 3- or 6-way channel interleaving would require distinguishing a non-power-of-two channel selector, which our current tooling does not infer.

We recover function sets and selection conditions for each available setup on a subset of our test systems (see Table II): Intel Core i9-9900K (S02), AMD Threadripper 1920X (S11), Intel Xeon E5-1630 v4 (S07), Intel Core i5-7200U (S01), and AMD Ryzen 9 9950X (S13). Table III lists the different categories of non-PTID configurations tested. The number of possible non-PTID evaluation setups is limited by what each system supports, by the available DIMMs with the correct number of ranks and banks, size, and ECC support, and by the mapping classes that our current tooling can model. Consequently, it is not possible to test every possible configuration type on every setup. Despite this, we have a large enough range of systems available to test multiple systems per configuration type. All recovered function sets pass the consistency check outlined in Section V: they demonstrate the near-100 % conflict rate for the full conditional function set and half the conflict rate for each removed function.

Linear Non-PTID Setups. Interleaving different DIMMs and ranks is more performant than sequential addressing. In a same-size non-PTID setup, it is still possible to fully interleave the different types of DIMMs. We found that under specific circumstances, these same-size non-PTID setups can still exhibit globally valid bank addressing functions. Table IV shows the recovered addressing functions on Intel Core i9-9900K and AMD Threadripper 1920X with two single-rank DIMMs of size 8 GB in channel A and a dual-rank DIMM of size 16 GB in channel B. The same function that is used for rank selection in channel B is utilized to select which DIMM is addressed in channel A. From an addressing function perspective, the two single-rank DIMMs are equivalent to one double-sized dual-rank DIMM.

More precisely, dual-channel non-PTID setups can still exhibit globally valid bank addressing functions if 1.) each channel contains a power-of-two number of DIMMs of identical size and rank, 2.) the memory size is channel-balanced (same size), and 3.) the numbers of ranks and banks are channel-balanced (same topology). Next, we will cover bank addressing of setups that do not satisfy all three requirements.

Mixed Topology. First, we consider the case where requirements 1) and 2) are still satisfied, but the combined number of ranks and banks differs between the two channels. We evaluate two scenarios: a single-rank (channel A) and dual-rank DIMM (channel B) of the same size, and 16 GB single-rank DIMM with 32 banks in channel A with two 8 GB DIMMs with 16 banks each in channel B. [6] Since we have a same-size setup, it is still possible to fully interleave the two channels. However, the indexing of ranks, bank groups, banks, and rows necessarily differs between the two channels.

In the first scenario, channel B requires an additional rank-addressing function, while channel A needs to increment the row index at a higher rate. Hence, both channels require their individual addressing function sets. Functions that are shared between the sets are globally valid. This must include a common channel selection function that determines which set of addressing functions is applied to map a physical address to a DRAM bank. Table V shows the reverse-engineered functions for a mixed-topology setup on five test systems. In all three instances of the first scenario, channel A contains 5 functions and channel B contains an additional addressing function to index the additional rank. The channel function is part of each addressing function set. Most other functions differ between the two sets because of shifted bits due to the presence or absence of the additional rank selection function. For example, on the Intel Core i9-9900K the higher bits $18 \oplus 22$ are used in an additional function in channel B. In the remaining channel B functions, all function bits higher than or equal to 18 are shifted by 1 bit, e.g., from 0x48000 in channel

[6] Unless explicitely stated otherwise DIMMs are assumed to have the same number of banks per rank.

TABLE VI
ADDRESSING FUNCTIONS FOR MIXED-SIZE SAME-TOPOLOGY NON-PTID SETUPS WITH 2 CONDITIONAL FUNCTION SETS ON INTEL CORE I9-9900K (S02), AMD THREADRIPPER 1920X (S11), AND INTEL XEON E5-1630 V4 (S07)

	Setup	Cutoff	Set
S02	1x2R(8 GB) + 1x2R(16 GB)	<16 GB	0x4080, 0x4b300, 0x88000, 0x110000, 0x220000, 0x440000
		>16 GB	0x2040, 0x44000, 0x88000, 0x110000, 0x220000
S11	1x2R(8 GB) + 1x2R(16 GB)	<16 GB	0x100, 0x7FEC0, 0x44448000, 0x88890000, 0x111120000, 0x222207EC0
		>16 GB	0x3FFC0, 0x22204000, 0x44448000, 0x88890000, 0x1111003FC0
S07	1x2R(8 GB) + 1x2R(16 GB)	<16 GB	0x4000, 0x400040, 0x880000, 0x1100000, 0x2200000, 0x40510C0
		>16 GB	0x2000, 0x200040, 0x440000, 0x880000, 0x1100000

TABLE VII
ADDRESSING FUNCTIONS FOR MIXED-SIZE MIXED-TOPOLOGY NON-PTID SETUPS WITH 3 CONDITIONAL FUNCTION SETS ON INTEL CORE I9-9900K (S02), AMD THREADRIPPER 1920X (S11), INTEL XEON E5-1630 V4 (S07), INTEL CORE I5-7200U (S01), INTEL CORE ULTRA 7 265K (S06) AND AMD RYZEN 9 9950X (S13)

	Setup	Cutoff	Channel	Set 1 [Channel = 0]	Set 2 [Channel = 1]
S02	1x1R(8 GB) + 1x2R(16 GB)	<16 GB	0xc3300	0x4080, 0x48000, 0x90000, 0x120000	0x4080, 0x88000, 0x110000, 0x220000, 0x440000
		>16 GB		0x2040, 0x44000, 0x88000, 0x110000, 0x220000	
	3x1R(8 GB)	<16 GB	0xc3300	0x4080, 0x48000, 0x90000, 0x120000	0x4080, 0x88000, 0x110000, 0x220000, 0x440000
		>16 GB		0x2040, 0x44000, 0x88000, 0x110000, 0x220000	
S11	1x1R(8 GB) + 1x2R(16 GB)	<16 GB	0x100	0x44450000, 0x888A0000, 0x111107EC0, 0x222228000	0x7FEC0, 0x44448000, 0x88890000, 0x111120000, 0x222207EC0
		>16 GB		0x3FFC0, 0x22204000, 0x44448000, 0x88890000, 0x1111003FC0	
	3x1R(8 GB)	<16 GB	0x100	0x44450000, 0x888A0000, 0x111107EC0, 0x222228000	0x7FEC0, 0x44448000, 0x88890000, 0x111120000, 0x222207EC0
		>16 GB		0x3FFC0, 0x22204000, 0x44448000, 0x88890000, 0x1111003FC0	
S07	1x1R(8 GB) + 1x2R(16 GB)	<16 GB	0x5555080	0x200040, 0x440000, 0x880000, 0x1100000	0x4000, 0x400040, 0x880000, 0x2200000, 0x1100000
		>16 GB		0x2000, 0x200040, 0x440000, 0x880000, 0x1100000	
S01	1x1R(8 GB) + 1x2R(16 GB)	<16 GB	0xc3300	0x4080, 0x48000, 0x90000, 0x120000	0x4080, 0x88000, 0x110000, 0x220000, 0x440000
		>16 GB		0x2040, 0x44000, 0x88000, 0x110000, 0x220000	
S06	1x1R(8 GB) + 1x1R(16 GB)	<16 GB	0x49900	0x4400, 0x8000, 0x22600, 0x50000, 0xa0000	0x4400, 0x8000, 0x10000, 0x22600, 0xa0000, 0x140000
	(16 banks + 32 banks)	>16 GB		0x2100, 0x4000, 0x8000, 0x20700, 0x50000, 0xa0000	
S13	1x1R(8 GB) + 1x1R(16 GB)	<16 GB	0x100	0x1E40, 0x44440800, 0x88881000, 0x111100200, 0x222200400	0x3E40, 0x21080800, 0x42101000, 0x84200200, 0x108400400, 0x210802000
	(16 banks + 32 banks)	>16 GB		0x1F40, 0x10840400, 0x21080800, 0x42100100, 0x84200200, 0x108401000	
	1x1R(16 GB) + 1x2R(32 GB)	<32 GB	0x100	0x3E40, 0x42101000, 0x84200200, 0x108400400, 0x210802000, 0x421080800	0x3E40, 0x80000, 0x42100800, 0x84201000, 0x108400200, 0x210800400, 0x421002000
		>32 GB		0x1F40, 0x40000, 0x21080400, 0x42100800, 0x84200100, 0x108400200, 0x210801000	

A to 0x88000 in channel B. While the functions share many structural similarities, the only shared global function besides the channel function involves only bits below bit 18. Based on the observation of equivalent addressing of two single-rank DIMMs and a double-sized dual-rank DIMM, we hypothesize that a 16 GB single-rank DIMM in channel A and two 8 GB single-rank DIMMs in channel B can be addressed in the same fashion. The channel B function set should contain an additional function to switch between the fully interleaved DIMMs. However, we did not have any larger single-rank DIMMs at the time of writing to confirm this hypothesis.

In the second scenario, each channel contains the same combined number of banks and therefore requires the same total number of functions. Still, we find in our two test setups (see Table V) that each channel is addressed with a separate function set. In the previous section, we observed that two single-rank DIMMs were addressed with the same addressing function as a double-sized dual-rank DIMM. This equivalence does not hold between single-rank DIMMs and double-sized single-rank DIMMs with double the number of banks. Even when the total number of banks is the same across both channels, we observe distinct bank addresses when the combined number of ranks differs between the channels.

Mixed Size. Next, we consider the case where requirements 1) and 3) are still satisfied, but channel A contains a larger amount of memory than channel B. While full channel interleaving is not possible, contiguous memory $2\times$ the size of the memory in channel B is still operated in fully interleaved channel mode [38]. In this setup, the first part of channel A is fully interleaved with channel B, behaving identically to a PTID setup with a set of linear addressing functions, including a channel function. The remaining upper portion of channel A is operated in single-channel mode. If this remaining region is power-of-two-sized, it can also be addressed via a set of linear addressing functions without a channel function. Instead of the channel fuction in the mixed-topology case, in this scenario, the cutoff between the two contiguous memory regions determines which addressing function set applies to a physical address.

We restrict our analysis to cases where the upper region is also a power-of-two, *i.e.*, to setups where channel A contains twice as much memory as channel B. Table VI details the recovered function sets for mixed-sized setups with an 8 GB dual-rank DIMM in Channel A and a 16 GB dual-rank DIMM in channel B on three test system. In each case, the lower 16 GB are fully interleaved via an additional channel function,

TABLE VIII
BANK ADDDRESSING FUNCTIONS FOR EACH EVALUATED PTID SETUP (SEE TABLE II)

System	Functions
S02	0x4080, 0x1b300, 0x48000, 0x90000, 0x120000
S03	0x4080, 0x1b300, 0x48000, 0x90000, 0x120000
S04	0x82600, 0x102100, 0x104200, 0x820000, 0x111040000, 0x222080000, 0x444208000, 0x888410000
S05	0x49900, 0x82600, 0x42140000, 0x84204400, 0x108408000, 0x210810000, 0x4210a0000
S06	0x49900, 0x82600, 0x42140000, 0x84204400, 0x108408000, 0x210810000, 0x4210a0000
S07	0x4000, 0x400040, 0x880000, 0x1100000, 0x2200000, 0x40510c0
S08	0x40, 0x404100, 0x2020900, 0x20000080, 0x40000200, 0x80001000, 0x100002000, 0x200000400
S09	0x40, 0x404100, 0x808200, 0x2020900, 0x40000080, 0x80000400, 0x100002000, 0x200004000, 0x400001000
S10	0x80, 0x404100, 0x808200, 0x1010400, 0x2020800, 0x40000040, 0x80001000, 0x100004000, 0x400002000
S11	0x100, 0x44450000, 0x888a0000, 0x111107ec0, 0x222208000
S12	0x100, 0x7e000, 0x42140000, 0x84204000, 0x108408000, 0x210810000, 0x4210a0000
S13	0x1f40, 0x21080800, 0x42100100, 0x84200200, 0x108401000, 0x210840400
S14	0x3ffc0, 0x44448000, 0x88890000, 0x111103fc0, 0x222204000
S15	0x3e40, 0x40214100, 0x42101000, 0x84200200, 0x108400400, 0x210802000, 0x421080800
S16	0x40, 0x21080800, 0x42100100, 0x84200200, 0x108401000, 0x210840400

resulting in function sets with 6 elements. The remaining upper region is indexed single-channel via 5 addressing functions.

Mixed Topology and Mixed Size. Finally, we consider the combined case that only satisfies 1), but mixes different numbers of ranks and banks and sizes. We evaluate three different scenarios: 3 identical DIMMs, a power-of-two number of DIMMs of different size and rank, and a power-of-two number of DIMMs of different size and same rank, but differing number of banks. These setups combine the characteristics of the two cases discussed above. Again, we restrict the size difference to a power of two. We observe a cutoff between two contiguous memory regions, with the lower region operating in dual-channel mode and the upper region in single-channel mode. Below the cutoff, we observe different bank addressing function sets for each channel. Hence, we obtain three addressing function sets: above the cutoff, below the cutoff in channel B, and below the cutoff in channel A. Assuming channel A contains the 2 DIMMs, the larger dual-rank DIMM, or the DIMM with more banks, in each scenario, respectively, each of these function sets contains 1 more function than the previous one: an additional channel function below the cutoff to address the channel and an additional rank or DIMM or bank selection function in channel A. Table VII shows the recovered function sets for all three configurations, with either 3 identical DIMMs or 2 DIMMs of different sizes and topologies, across 6 test systems.

Overall, we observe that the function sets for different setups on the same machine have similarities, but we still do not have a consistent and predictable form of transforming functions from one setup to another. For example, for Intel Core i9-9900K (S02), we observe precise function set matches for channel A containing a single-rank 8 GB DIMM, indpendently from whether channel B containes an 8 GB or 16 GB dual-rank DIMM. Still, we observe different bits appearing in the channel function for various setups on this machine: While all setups have 8,9,12,13 as channel bits, some have bits 15 and 18 in addtion, while others have bits 18 and 19 instead. We also found that the absence of pure row bits in PTID setups does not imply the absence of pure row bits on the same machine in a non-PTID setup. In PTID setups, Intel Core Ultra 7 265K (S06) utilizes all addressing bits in bank functions, while the mixed-size and mixed-topology setup exhibits pure row bits again. On the other test systems, the presence of pure row bits in the tested non-PTID setups matches the PTID findings in prior work: the AMD machines (S11, S13) demonstrates no pure row bits, while the older Intel Core (S01, S02) and Xeon (S07) machines do.

VII. EVALUATION OF PTID SETUPS

We compare our tool against 7 state-of-the-art tools on 15 systems (S02–S16) across a wide range of DIMM setups, as shown in Table II. To demonstrate the effectiveness of DREAMS on PTID setups, we compare it with existing tools [4, 7, 9, 15, 16, 18, 19]. As we focus on generic reverse engineering, we exclude tools that are platform-specific (RISC-V [17]), require manually determined parameters (Sudoku [20]) or rely on system features not widely available. We decided to exclude the tool by Helm et al. [8] since the required performance counters are only available on a single one of our 16 test systems. Further, Xiao et al. [6] do not provide a public artifact and their approach has already been demonstrated to be non-generic by Wang et al. [7]. We evaluate the performance, correctness, and reliability of recovering linear functions on setups with power-of-two number of DIMMs of identical size and number of ranks and banks.

Methodology. To determine not only correctness, but also reliability and consistency, we execute each function recovery tool 10 times on each test system and record the runtime and

TABLE IX
AVERAGED RUNTIME, CORRECTNESS, AND STABILITY OF TESTED ADDRESS REVERSING TOOLS OVER 10 RUNS ON 15 TEST SYSTEMS.

Tool	Runtime[s] ↓	Correctness[%] ↑	Stability[bits] ↓
DRAMA	1895.7	11.2	23.3
DRAMDig	169.7	3.3	0.1
AMDRE	1295.9	8.0	1.5
Frigo et al. [15]	2215.3	14.8	0.4
DARE	418.4	23.2	5.4
Knock-Knock	17.5	1.8	0.0
Chen et al. [19]	26.8	4.2	8.1
DREAMS (ours)	22.1	100	0.0

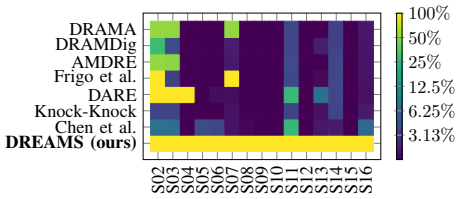


Fig. 6. Highest achieved function correctness percentage across 10 runs for each DRAM function recovery tool on our 15 test systems. Each missing or incorrect function halves the percentage.

recovered functions of each run. We calculate the arithmetic mean of the runtime on each system and compute the geometric mean across the 15 test systems. To evaluate correctness, we compare each function set with reference functions verified using the methodology of Heckel et al. [37] for each system, listed in Table VIII. First, we generate 2 500 random physical address pairs. For each pair, both addresses map to the same bank according to the reference functions. Next, we determine the fraction of these address pairs that map to each bank. The sample size guarantees a standard error $\leq 0.01\%$. Finally, we take the maximum percentage of correctly grouped addresses across the 10 runs per system and compute the arithmetic mean across all test systems. Note that the term correctness here does not constitute a comparison against physical ground truth. We consider a set of functions to be correct, if they can fully explain all observed row-conflict behavior. To evaluate the stability of the function recovery, we measure the pairwise Hamming distance between the recovered function sets for each run; e.g., a distance of 0 bits indicates that all sets are equal. We add the distances for a single run, then compute the arithmetic mean across all runs on the same system, and finally compute the arithmetic mean across these per-system results. The results over all systems are shown in Table IX. Figure 6 provides a visual overview of which tools yielded correct results on which systems. For each combination of tool and system, we consider the maximum correctness achieved across 10 runs of the tool on the system.

Results. For Knock-Knock [18], reverse-engineering failed on all of our tested systems; *i.e.*, the tool did not return any functions. We indicate the function as `0x00` when the tested tool returned no function. In our experiments, Knock-Knock reaches a stability of 0.0, a runtime of 17.5 s, and a correctness of 1.8 %. Since Knock-Knock did not return any functions, we exclude Knock-Knock in the following discussion.

The tool by Frigo et al. [15] runs on average the longest, followed by DRAMA [4] and AMDRE [16]. With 22.1 s on average, DREAMS is the fastest, closely followed by Chen et al. [19] with 26.8 s on average. Among the existing tools, DARE [9] achieves the highest average best-run correctness of 23.2 % recovering fully correct functions in 3 tested setups, followed by the tool by Frigo et al. [15] with 14.8 %. In contrast, DREAMS produces stable and correct results on **all** systems. DRAMA produces the least stable output with 23.3 bits variability, followed by the tool by Chen et al. [19] with 8.1 bits. The most stable existing tool is DRAMDig, with a stability of 0.1 bits, which is comparable to our tool's stability of 0.0 bits.

Possible Failure Reasons. Some of our results are to be expected given the assumptions and inner workings of the tested tools. All tested tools besides Knock-Knock [18] and Chen et al. [19] use brute-force methods to identify linear addressing functions. Depending on the complexity, e.g., the number of bits used in these functions, this process can result in a long runtime. In contrast, DREAMS performs targeted measurements and bit flips to recover the addressing functions, thereby achieving significantly greater efficiency.

Several tools face restrictions in what type of functions they can principally recover. Frigo et al. [15] state that their tools make significant assumptions regarding the bits used for different purposes, limiting its scope mostly to the Intel consumer platforms they tested [39]. AMDRE recovers functions inside a 2 MB allocated huge page. Hence, it cannot recover bank bits above and including bit 21 used by newer Intel Core and AMD systems. Jattke et al. [9] were the first to shift physical addresses by a PCI reclaim offset before applying linear XOR functions. Since no tools besides DARE and DREAMS account for these offsets, they failed on many of our AMD-based test systems. Only DREAMS handles PCI reclaims correctly on Intel server CPUs. DRAMDig [7] implicitely assumes the presence of pure row bits, which is not the case on many newer Intel consumer generations and AMD devices resulting in recovery failure. The tool introduced by Chen et al. [19] failed to recover complete functions on some systems because it cannot handle bits used by multiple functions. We found that 13 out of the 15 tested systems PTID setups had at least one bit that was shared across multiple functions. DREAMS correctly handles function overlap via a SAT-solver-based approach on all of our test systems. Many of these restrictions of prior work either lead to complete function recovery failure or result in partial functions. For example, on S02 (the system where most tools succeeded) the tool by Chen et al. [19] only achieves partial function recovery, precisely not detecting overlapping function (parts) while recovering all functions that did not overlap at all. When picking colliding addresses based on a set of bank addressing functions, e.g., for a Rowhammer attack, each correct function

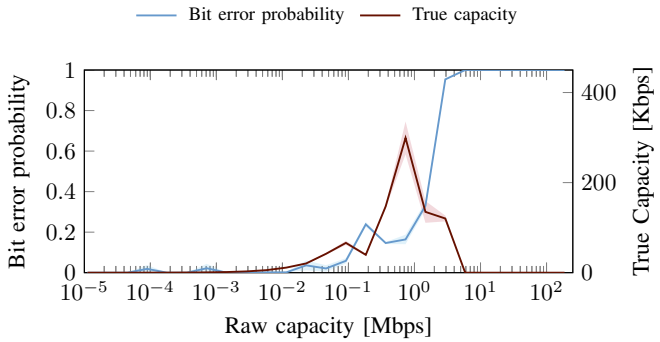


Fig. 7. Covert-channel performance over 20 runs with one same-bank sender address and one receiver address. The x-axis shows the mean raw capacity of each bit-slot group. Shaded areas show the standard error.

doubles the likelihood of bank collision. In contrast, a partial addressing function, even if it just misses a single bit, does not double the likelihood of a bank collision.

We only provide a manually determined threshold to tools that do not implement this functionality themselves or produced very unstable thresholds across several runs, such as Knock-Knock and TRRespass. For tools that automatically determine thresholds, a failure to determine a correct threshold can also lead to incorrectly recovered functions. Finally, some tools, such as DRAMA, might fail due to system noise as their recovery approaches may not be sufficiently noise-resilient.

VIII. Security Evaluation

Reverse-engineered DRAM addressing functions have direct security implications beyond recovering an undocumented hardware configuration. They make DRAM-internal contention predictable from software and allow an attacker to deliberately select addresses that share, for instance, a bank but not a row, or a bank and a row. We evaluate this impact with two components. Both components are evaluated on S13, an AMD Ryzen 9 9950X (Zen 5) system with one 8 GB (16 banks) and one 16 GB (32 banks) single-rank DDR5 DIMM. First, we use a row-buffer covert channel, as this is the most direct primitive enabled by correct bank addressing functions and a useful indicator for other attacks relying on the same row-hit and row-conflict behavior [4]. Second, we model how wrong or incomplete functions reduce the Rowhammer budget within one refresh interval, because compensating for uncertain bank-conflict predictions requires hammering more candidate addresses.

A. Covert Channel

The covert channel follows the DRAMA row-buffer covert channel [4]. Sender and receiver each use one private address; the channel does not require shared memory, but it requires suitable same-bank addresses. With incorrect or only globally valid addressing functions, same-bank selection becomes unreliable on non-PTID setups, while DREAMS can choose such address pairs for any region. To transmit a one bit, the sender repeatedly accesses its address during the bit slot. To transmit a zero bit, the sender stays idle. The receiver repeatedly accesses and times its own address: an active sender frequently opens the sender row, causing receiver row conflicts, while an idle sender lets the receiver row remain open more often, causing faster row-buffer hits. The receiver decodes the bit by voting over the timing samples collected during the bit slot.

TABLE X
Comparison with DRAM and memory-system covert channels.

Work	Mechanism	Reported Capacity
Wu et al. [40]	Memory-bus lock contention	739 bit/s
Pessl et al. [4]	DRAM row-buffer conflicts	2.1 Mbit/s
Lv et al. [41]	DRAM row-buffer conflicts	12 kbit/s
Paiva et al. [42]	DRAM power via RAPL telemetry	936 bit/s
Dagli et al. [43]	Shared-DRAM contention	5.9 kbit/s
Bostanci et al. [32]	Rowhammer-defense latency (PRAC/RFM)	39.0 kbit/s/48.7 kbit/s
DREAMS	DRAM row-buffer conflicts	299 ± 36 kbit/s

Figure 7 shows the aggregated result over 20 runs per bit-slot length, with each point placed at the mean raw capacity of that bit-slot group. The channel reaches its highest true capacity[7] of 299 ± 36 kbit/s at a raw bitrate of 732 kbit/s, with a bit error probability of 16.4 %. For a more conservative operating point, 45.8 kbit/s raw bitrate still yields 41.9 ± 2.0 kbit/s true capacity with a mean bit error probability of 2.08 ± 1.60 %.

Our covert channel is in the order of magnitude of prior DRAM-based covert channels [44], as shown in Table X. We conclude that non-PTID setups do not prevent DRAM covert channels: once the addressing functions are recovered correctly, the same row-buffer primitive becomes usable on configurations where previous reverse-engineering tools would misclassify address pairs or fail entirely. Consequently, DRAM-bank partitioning, memory coloring, VM placement, and similar isolation mechanisms need to and, with our work, can account for non-PTID mapping behavior rather than assuming one global bank-function set.

B. Rowhammer Hammering Budget

Correct DRAM functions are equally relevant for Rowhammer attacks, where the attacker must spend a limited refresh interval on useful aggressor accesses [3, 9, 10, 15, 26, 45, 46]. For example, on our S13 system, one flush-based hammering round with the selected address pattern takes 774 cycles, so a 64 ms refresh interval at 3 GHz allows at most 248 062 rounds. This budget only fully contributes to Rowhammer if the selected address group is a real bank conflict. If only a fraction p of predicted conflicts is correct, the useful work drops proportionally: 50 % correct predictions leave about 124 000 useful rounds, 25 % leave about 62 000, and 10 % leave about 25 000. Compensating with more candidates further dilutes the refresh window, as splitting it over n groups leaves only $248\,062/n$ rounds per group and wrong groups consume time without producing useful activations. This is

[7] True capacity from Shannon's theorem: $T = C \cdot (1 + ((1-p) \cdot \log_2(1-p) + p \cdot \log_2(p)))$, with p the bit error probability and C the raw capacity.

still optimistic, since memory-controller scheduling and request merging, bank-level parallelism, cache and TLB effects, TRR sampling behavior, temporal variation in disturbance thresholds, and self-interference inside the hammering loop are not accounted for [15, 45, 47, 48]. The completeness of reverse engineering can therefore decide whether enough useful activations fit into one refresh window and Rowhammer is feasible or not, *i.e.*, in line with Heckel et al. [5].

IX. Discussion and Limitations

Reverse-engineered DRAM functions have both offensive and defensive applications. The most well-known application is in Rowhammer attacks [3, 4, 9, 10, 49], where the knowledge of the DRAM addressing functions allows an attacker to efficiently target specific banks and even rows in memory. While hammering in principle is also possible without knowledge of the addressing functions, using single-sided, one-location, or RowPress-style hammering [2, 50–52], the more effective double-sided and many-sided hammering techniques require precise knowledge [2, 15, 26, 45, 46, 53]. Heckel et al. [5] demonstrated in a large-scale Rowhammer study that the lack of correctly reverse-engineered DRAM addressing functions is a central limitation for the effectiveness of Rowhammer attacks in practice. Prior work also utilized DRAM row-buffer behavior for row-hit and row-conflict side channels [4, 54–56]. Some memory coloring and partitioning schemes take DRAM banks and other linear indexing functions into account and, therefore, require knowledge of the DRAM addressing functions [11–13, 57]. Software Rowhammer mitigations based on physical row or bank isolation can likewise benefit from accurate DRAM address mappings [58–60].

Our approach builds upon observations and mechanisms from prior DRAM function recovery work, but targets a different failure mode. In contrast to brute-force based approaches such as DRAMA [4], we use an analytical approach following the line of work of Xiao et al. [6] and Chen et al. [19]. In particular, identifying pure row bits, pure row pairs, and bank bit groups is conceptually similar to Chen et al. [19], but DREAMS uses these observations differently: we use structured bit-difference measurements for threshold selection, construct a combined pure row mask, and then test single candidate bits together with this mask in the RA-SBDR test. However, analytical approaches that reason about bits or bit pairs locally can miss functions whose bits overlap with other functions. The approach of Xiao et al. has already been shown to fail in such cases [7], and our evaluation shows that for the approach of Chen et al. this limitation still applies. For example, on S02, where we observed the highest success rate among prior tools, the approach of Chen et al. recovered only the non-overlapping parts of the channel function and missed the two functions that share bits with this channel function. DREAMS addresses this limitation with a SAT-based recovery step that models the interaction between multiple candidate bits simultaneously, preserving the performance advantages of an analytical approach while recovering overlapping functions reliably. The closest prior work in terms of runtime is Knock-Knock [18], which uses null-space analysis to reduce the search space and supports overlapping linear functions in the PTID model. In our evaluation, however, Knock-Knock did not recover a correct function set on any tested system. This may be improved with engineering changes, such as better threshold selection, offset handling, or address selection, *i.e.*, steps that help with automation of the reverse-engineering. While Knock-Knock then may succeed on additional PTID systems, these adaptations would not address the central non-PTID challenge: mixed-size and mixed-topology setups require conditional function sets that are valid only for selected address ranges or channels, whereas prior tools still assume one globally valid set of linear functions.

Since hardware vendors do not publish bank addressing functions, verification against ground truth is not possible. The verification procedure outlined in Section V ultimately relies on the same row-conflict timing side channel as our function recovery. Therefore, it suffers from the same restriction as prior work et al. [37]: it provides a meaningful consistency check, but does not constitute independent proof of correctness. Prior work [4], however, demonstrated that row-conflict timing measurements were consistent with physical probes. Furthermore, since the row-conflict timing side channel does not leak the ordering of banks, there are equivalent bank addressing functions that explain the same row-conflict observations. Therefore, our verification does not prove that these exact functions are indeed implemented in hardware, rather it demonstrates that the model is internally consistent, following the approach of software-only verification of prior work [37].

While our approach goes beyond the linearity assumption, completeness is infeasible in the vast design space available to CPU manufacturers. For example, if a memory controller scattered DRAM addresses with a cryptographic function, reverse engineering would require inferring the cryptographic key, which good cryptographic functions are designed to prevent. Even without cryptographic functions, some designs may be outside the scope of our approach and prior approaches. Our observations point to non-linear combinations of linear functions over memory ranges, but some hardware may directly use non-linear functions, similar to non-power-of-two last-level cache slices. Systems that directly interleave a non-power-of-two number of effective channels, e.g., three or six, may therefore require extensions that handle non-linear functions, such as the cache-slice approach by Gerlach et al. [31]. Similarly, we do not claim to cover every possible asymmetric setup. Our evaluation covers a substantial set of non-PTID configurations not addressed by prior work, while focusing on configurations supported by our test systems: power-of-two effective channel counts, identical DIMMs within each channel, and equal-sized or $2\times$-sized channel regions. Other asymmetric cases remain open, including non-power-of-two numbers of effectively interleaved channels, non-power-of-two size ratios between channels, and channels populated with different DIMMs internally. For non-power-of-two size ratios, the controller may follow the same pattern as the power-of-two cases, but additionally splitting the upper single-channel space

into multiple contiguous regions. For internally asymmetric channels, the mapping may likewise remain simple, e.g., placing DIMMs linearly within a channel, but this requires additional conditions that our current model does not infer. Non-power-of-two interleaved channel counts are potentially more demanding, as they may require a genuinely non-linear selector similar to non-power-of-two last-level cache slices and could benefit from techniques such as Gerlach et al. [31]. Future work should extend the tooling to these cases and determine whether they continue the observed pattern, *i.e.*, conditional selection among locally linear function sets, or require a more general non-linear model. Still, our work is the first to address non-PTID setups with a novel approach that accounts for conditional function sets, and we are able to successfully reverse-engineer functions on a wide range of systems with different non-PTID setups, demonstrating the practical applicability of our approach.

X. Conclusion

Rowhammer-style security evaluations, DRAM side channels, and DRAM isolation mechanisms rely on correct knowledge of which physical addresses share a DRAM bank. After a decade of research reverse-engineering these mappings, we discovered that non-PTID setups can follow previously unknown conditional DRAM mappings, where different parts of memory obey different locally valid linear function sets. DREAMS is the first approach to reliably recover DRAM bank addressing functions for these setups by detecting bit-inconsistent behavior, recovering the applicable function sets and their selection conditions, and verifying the resulting conditional mapping in software.

DREAMS also handles overlapping functions by using SAT constraints to model interactions between candidate bits, allowing one recovery pipeline to cover both PTID and non-PTID setups. We evaluate DREAMS on 19 non-PTID DIMM configurations across 6 Intel and AMD DDR4/DDR5 systems, recovering mappings with up to 3 conditional function sets selected by address cutoffs and channel functions.

XI. Ethical Considerations

All experiments were solely performed on machines owned and controlled by our lab that did not contain any confidential or sensitive data. DREAMS is dual-use, as are most offensive-security artifacts (e.g., DRAMA [4], TRRespass[15], Phoenix[10]): the recovered addressing functions could improve Rowhammer, covert-channels, or isolation bypass attacks, but they are equally necessary for developing and evaluating sound defenses. Prior bank function reverse engineering works [4, 7, 18, 19] have published functions for common architectures, as well as released full artifacts. Hence, for many setups, bank addressing functions are already known or can be reverse-engineered with some manual effort, e.g., to make existing tools more noise-resistant. While our artifact extends the capabilities of these tools to (a smaller number of) non-PTID setups, it does not change the nature of the dual-use risk consideration regarding bank addressing functions. Publishing our methodology, while witholding functions or tooling, would not prevent malicious actors from reimplementing our algorithm. However, it hinders other reseachers to gain a wider understanding of these mapping behaviors, reverse-engineer the functions on their own systems, build on our methodology, e.g., extending it to further asymmetric setups, and ultimately develop informed mitigations. Consistent with established norms in the security literature, we judge the defensive and scientific value of full disclosure to outweigh the risk, and release both the tool and the discovered addressing functions publicly.

XII. Acknowledgment

We thank Mikka Rainer and Michael Schwarz for discussions, early confirmation experiments and hardware access. This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85, FWF projects NeRAM 10.55776/I6054 and RAMSys 10.55776/PIN1109425), the Deutsche Forschungsgemeinschaft (grant no. 503876675), and the European Union (grant no. ROF-SG20-3066-3-2-2). Additional funding was provided by generous gifts from Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties. We utilized the following LLM tools: Claude and ChatGPT. These tools were used for wording improvements, improvements of selected sentences for clarity and presentation, and assistance with figure presentation. All LLM-assisted output was reviewed, edited, and verified by the authors.

References

[1] 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.

[2] M. Seaborn, "Exploiting the DRAM rowhammer bug to gain kernel privileges," 2015. [Online]. Available: http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[3] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.

[4] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.

[5] M. Heckel, N. Sayadi, J. Juffinger, C. Fiedler, D. Gruss, and F. Adamsky, "FlippyRAM: A Large-Scale Study of Rowhammer Prevalence," in *NDSS*, 2026.

[6] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.

[7] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A Knowledge-assisted Tool to Uncover

DRAM Address Mapping," in *Design Automation Conference (DAC)*, 2020.

[8] C. Helm, S. Akiyama, and K. Taura, "Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020.

[9] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms," in *USENIX Security*, 2024.

[10] D. Meyer, P. Jattke, M. Marazzi, S. Qazi, D. Moghimi, and K. Razavi, "Phoenix: Rowhammer Attacks on DDR5 with Self-Correcting Synchronization," in *S&P*, 2026.

[11] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[12] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses," in *International Conference on Computational Science and Engineering*, 2013.

[13] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic Memory Hierarchy and Virtualization for Modern Multi-Core Embedded Systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[14] M. Seaborn, "How physical addresses map to rows and banks in DRAM," 2015. [Online]. Available: http://lackingrhoticity.blogspot.com/2015/05/how-physical-addresses-map-to-rows-and-banks.html

[15] 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.

[16] M. Heckel and F. Adamsky, "Reverse-Engineering Bank Addressing Functions on AMD CPUs," in *DRAMSec Workshop*, 2023.

[17] M. Marazzi and K. Razavi, "RISC-H: Rowhammer Attacks on RISC-V," in *DRAMSec Workshop*, 2024.

[18] A. Plin, L. Casalino, T. Rokicki, and R. Salvador, "Knock-Knock: Black-Box, Platform-Agnostic DRAM Address-Mapping Reverse Engineering," in *uASC*, 2025.

[19] W. Chen, S. Tang, Y. Tang, X. Luo, Y. Zhang, and W. Qiang, "ρHammer: Reviving RowHammer Attacks on New Architectures via Prefetching," in *MICRO*, 2025.

[20] M. Wi, W. Doh, S. Baek, S. Park, M. Erez, and J. H. Ahn, "Sudoku: Refining DRAM Address Mapping through Advanced Timing and Configuration Analysis," *IEEE Access*, 2026.

[21] I. Micron Technology, "DDR4 SDRAM: MT40A1G4, MT40A512M8, MT40A256M16," 2014. [Online]. Available: http://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf

[22] JEDEC Solid State Technology Association, "DDR4 SDRAM Standard," 2021. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[23] ——, "DDR5 SDRAM Standard," 2024. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5c01

[24] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer bug to gain kernel privileges," in *Black Hat USA*, 2015.

[25] F. de Ridder, P. Jattke, and K. Razavi, "Posthammer: Pervasive Browser-based Rowhammer Attacks with Postponed Refresh Commands," in *USENIX Security*, 2025.

[26] 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.

[27] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement," in *ASPLOS*, 2010.

[28] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing Memory Interference in Multi-Core Systems via Application-Aware Memory Channel Partitioning," in *MICRO*, 2011.

[29] L. Kirk, "Tailslayer," 2026. [Online]. Available: https://github.com/LaurieWired/tailslayer

[30] AMD, "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors," 2015. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/archived-tech-docs/programmer-references/48751_16h_bkdg.pdf

[31] L. Gerlach, S. Schwarz, N. Faroß, and M. Schwarz, "Efficient and generic microarchitectural hash-function recovery," in *S&P*, 2024.

[32] F. N. Bostancı, O. Canpolat, A. Olgun, I. E. Yüksel, K. Kanellopoulos, M. Sadrosadati, A. G. Yağlıkçı, and O. Mutlu, "Understanding and Mitigating Covert Channel and Side Channel Vulnerabilities Introduced by RowHammer Defenses," in *MICRO*, 2025.

[33] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.

[34] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[35] M. Schwarz, "misc0110/PTEditor," 2026. [Online]. Available: https://github.com/misc0110/PTEditor

[36] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide," 2024.

[37] M. Heckel, F. Adamsky, J. Juffinger, F. Rauscher, and D. Gruss, "Verifying DRAM Addressing in Software," in *ESORICS*, 2025.

[38] Intel, "12th Generation Intel Core Processors: Datasheet Volume 1," 2021.

[39] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass Github Repository," 2020. [Online]. Available: https://github.com/vusec/trrespass/tree/master/drama
[40] Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud," *ACM Transactions on Networking*, 2014.
[41] Z. Lv, Y. Zhao, C. Zhang, and H. Li, "DRAMD: Detect Advanced DRAM-Based Stealthy Communication Channels with Neural Networks," in *IEEE Conference on Computer Communications*, 2020.
[42] T. B. Paiva, J. Navaridas, and R. Terada, "Robust Covert Channels Based on DRAM Power Consumption," in *International Conference on Information Security*, 2019.
[43] I. Dagli, J. Crea, S. Seckiner, Y. Xu, S. Köse, and M. E. Belviranli, "MC³: Memory Contention-Based Covert Channel Communication on Shared DRAM System-on-Chips," in *DATE*, 2025.
[44] A. Kushwaha, A. Jain, M. Patel, and B. Panda, "Golmaal: Thanks to the Secure TimeCache for a Faster DRAM Covert Channel," in *DRAMSec*, 2022.
[45] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," in *S&P*, 2021.
[46] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering From the Next Row Over," in *USENIX Security*, 2022.
[47] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.
[48] A. Olgun, F. N. Bostanci, I. E. Yuksel, O. Canpolat, H. Luo, G. F. Oliveira, A. G. Yaglikci, M. Patel, and O. Mutlu, "Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance," in *HPCA*, 2025.
[49] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.
[50] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *S&P*, 2018.
[51] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *ISCA*, 2023.
[52] J. Juffinger, S. R. Neela, M. Heckel, L. Schwarz, F. Adamsky, and D. Gruss, "Presshammer: Rowhammer and Rowpress without Physical Address Information," in *DIMVA*, 2024.
[53] A. Kaur, P. Srivastav, and B. Ghoshal, "Flipping Bits Like a Pro: Precise Rowhammering on Embedded Devices," *IEEE Embedded Systems Letters*, vol. 15, no. 4, pp. 218–221, 2023.
[54] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.
[55] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX," in *CCS*, 2017.
[56] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *S&P*, 2018.
[57] J. Hofmann, C. Fournet, B. Köpf, and S. Volos, "Gaussian Elimination of Side-Channels: Linear Algebra for Memory Coloring," in *CCS*, 2024.
[58] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory," in *USENIX Security*, 2017.
[59] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.
[60] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *USENIX OSDI*, 2018.

APPENDIX A
ARTIFACT APPENDIX

DRAM bank function recovery tools assume conventional setups with a power-of-two number of identical power-of-two-sized DIMMs (PTID) but fail on asymmetric non-PTID setups. In this paper, we discover that many non-PTID setups follow previously unknown conditional DRAM mappings, where different parts of memory obey different locally valid linear function sets. This artifact contains DREAMS, the first approach to recover and verify DRAM bank addressing functions on non-PTID setups with such conditionally valid function sets.

A. Description & Requirements

1) How to access: Our original artifact is available at https://doi.org/10.5281/zenodo.22831587. Updated versions can be found at https://github.com/isec-tugraz/DREAMS.

2) Hardware dependencies: Intel or AMD system. Symmetrical DIMM setup with 1, 2, 4, or 8 identical DIMMs equally distributed over all channels or dual-channel asymmetric setup that meets the following requirements:

- each channel only contains identical DIMMs
- each DIMM is power-of-two sized
- channels can vary in the following properties:
  - channel A contains 2x the memory of channel B
  - channel A contains 2x the number of ranks of channel B
  - channel A contains 2x the number of banks of channel B

3) Software dependencies: Our scripts assume a Ubuntu/Debian based operating system. Secure Boot must be disabled to support PTEditor (https://github.com/misc0110/PTEditor).

4) Benchmarks: None.

B. Artifact Installation & Configuration

Download the zipped artifact file and unzip its contents to `/path/`. PTEditor and other dependencies can be installed via the installation script `sudo /path/install.sh`.

C. Major Claims

- (C1): DREAMS achieves comparable performance as state-of-the-art bank function recovery tools on PTID systems. This is outlined in Section-VII and proven by the experiment (E1).
- (C2): DREAMS can recover conditional bank addressing function sets in certain non-PTID setups. This is outlined in Section-VI and proven by the experiment (E2).

D. Evaluation

1) Experiment (E1): [PTID Setup] [15 human-minutes + 5 compute-minutes]: Bank function reverse-engineering and verification. This should run within a few minutes and produce a verified function set.

[Preparation] Disable Secure Boot. Ensure that the PTEditor module is loaded. Equip the system with a power-of-two number (1, 2, 4, 8) of identical DIMMs that are equally distributed amongst the channels.

[Execution] Run `sudo /path/run.sh`.

[Results] The output will contain the recovered functions, a statement whether all functions verified correctly as well as the runtime of the successful run.

2) Experiment (E2): [Non-PTID Setup] [20 human-minutes + 5 compute-minutes]: Bank function reverse-engineering and verification. This should produce verified conditional function sets.

[Preparation] Disable Secure Boot. Ensure that the PTEditor module is loaded. Equip the system with an asymmetric DIMM setup that meets the criteria outline in Section A-A2, e.g.,

- different rank:
  - channel A: a single-rank 8 GB DIMM
  - channel B: a dual-rank 8 GB DIMM, or
- different topology:
  - channel A: a single-rank 8 GB DIMM
  - channel B: a single-rank 16 GB DIMM

[Execution] Run `sudo /path/run.sh`.

[Results] The output will contain the recovered conditional function sets, a statement whether all functions verified correctly as well as the runtime of the successful run.

E. Customization

The automated script tests common parameters. Some systems may require further tuning of parameters in `config.h`. Since the automated test varies these parameters, the tool should then be run manually via `make; sudo ./dreams`. Note that parameter changes require recompilation.