LAD-IXoC: Loop-based Attack Detection with Integrated Xor Filter and CMS

Roberto Martínez*, Alessandro Palumbo†, Pedro Reviriego‡, Rubén Salvador†, David Larrabeiti*
*Department of Telematic Engineering, Universidad Carlos III de Madrid, Spain
Email: {robermar, dlarra}@it.uc3m.es
†CentraleSupélec, Inria, CNRS, IRISA, Rennes, France
Email: {alessandro.palumbo, ruben.salvador}@inria.fr
‡ETSI de Telecomunicación, Universidad Politécnica de Madrid, Spain
Email: pedro.reviriego@upm.es

***Abstract*—With the miniaturization of DRAM technology, memory devices have become increasingly susceptible to hardware-based attacks that exploit physical proximity between rows to alter or extract data. Notable among these are loop-based attacks such as Rowhammer, which induces bit flips in adjacent memory rows. Other known attacks, like Spectre, leverage shared memory to infer confidential data by cache access time estimations. To address these threats, which share similar loop-based code patterns, we propose an integrated detection system based on two data structures: xor filters (XF) and Count-Min Sketches (CMS). The system requires only three memory accesses per activation of a memory row to detect loop attacks, filtering most malicious loops in a first stage (i.e., the XF) and monitoring the remaining ones in a second stage (i.e., the CMS). An alert is triggered when a loop exceeds a defined activation threshold. Experimental results demonstrate that our approach effectively detects not only loop attacks but also other anomalous instructions that generate suspicious loops, achieving low undetected attack rates, on average below 0.4%, while consuming minimal memory and computational resources. These results position our system as a lightweight and reliable defense against diverse loop-based memory attacks, unlike other approaches that account for loops of a single attack type.**

***Index Terms*—Rowhammer, Spectre, Loop-based hardware attacks, CMS, XOR Filter**

I. Introduction

Over the years, the size of DRAM chips has been continuously reduced to improve access speed and performance. However, this miniaturization has also decreased the distance between internal components, making memory cells more susceptible to hardware-based security vulnerabilities. As a result, new generations of DRAM are increasingly vulnerable to physical-level attacks that aim to manipulate or extract sensitive information by altering hardware behavior.

A prominent example is the *Rowhammer* attack [1], which exploits charge leakage between adjacent rows by rapidly activating a row in tight loops. This repeated access can induce *bit flips* in nearby rows, allowing attackers to corrupt data or even gain unauthorized access. Another important example is *Spectre* [2], which abuses shared memory pages and the behavior of cache memory to infer whether specific data has been accessed by another process. It works by mistraining the branch predictor to speculatively execute instructions that access sensitive data, which are then leaked via microarchitectural side effects such as cache timing. Though transient in nature, these effects persist long enough to be measured, revealing secrets without violating architectural isolation.

979-8-3315-1489-1/25/$31.00 ©2025 IEEE

To mitigate such threats, various strategies have been developed [3]–[7]. One popular approach involves counting the number of memory accesses per row or instructions using hardware or software-based counters. When the number of activations approaches the *Rowhammer threshold* (the minimum required to cause a bit flip), the system can trigger a preventive action or alert. However, as DRAM densities increase, this threshold decreases, demanding more accurate and efficient tracking mechanisms. Other techniques, such as increasing refresh rates or employing queue-based filtering, can reduce vulnerability but often at the cost of performance overhead.

Recent research has proposed more compact and configurable countermeasure designs using data structures such as CMSs [8] to efficiently track memory activations. For instance, *CoMeT* [5] and CMS-based detection for microarchitectural attacks have demonstrated high detection accuracy using minimal resources. Yet, many of these solutions are tailored to detect only Rowhammer and struggle with broader loop-based threats, particularly those involving legitimate instructions behaving abnormally.

Moreover, existing counters can suffer from saturation under high instruction throughput, especially when multiple loops are active simultaneously. This saturation can lead to false positives, missed detections, or system instability. Addressing counter overflow often requires additional processing and memory access, which increases latency and power consumption [9]–[11].

In this work, we propose a two-stage loop attack detection system that combines two data structures, a XF and a CMS. The first stage pre-filters memory accesses using the XF, discarding most benign operations and passing only potentially malicious instructions to the next stage. The second stage counts repeated instructions using a CMS within a configurable time window. If the count exceeds a dynamic threshold, an alert is triggered; otherwise, the counters are reset. This design enables accurate detection of different loop-

based attacks with only three memory accesses per instruction and a low memory footprint.

The main contributions of this paper are:

1) A novel two-stage loop detection system using a XF and CMS that requires only three memory accesses to check each instruction.
2) Effective filtering of up to 98% of memory activations from loop-based attacks (including Rowhammer and Spectre) before they reach the counting stage.
3) Detection of anomalous looping patterns from legitimate instructions, by tracking instruction-level repetitions using compact CMS counters.
4) Configurable alert mechanism, which raises alarms when instruction repetition exceeds a threshold within a specified time window.
5) Compatibility with other mitigation techniques, making our system integrable with existing defense mechanisms.

We validate our system through Java-based simulations under three scenarios:

- Safe instruction loops across various thresholds and time windows (to assess false alarms);
- Synthetic Rowhammer and Spectre attacks (to test detection rates); and
- Execution of a legitimate program under active attack (to evaluate robustness and false negative rates).

The remaining part of the document is organized as follows: Section II covers fundamental concepts related to loop attacks and relevant probabilistic data structures. Section III presents our proposal, which is based on an integrated data structure. Section IV reports the experimental results, and Section V provides the analysis and conclusions.

II. Background

A. Microarchitectural Loop-based Attacks

Microarchitectural attacks exploit hardware-level features and behaviors to bypass security boundaries. Two prominent classes of such attacks, particularly relevant in the context of loop-based memory access patterns, are Rowhammer and Spectre. These attacks manipulate memory access operations to induce faults or retrieve confidential data.

1) Rowhammer: Rowhammer is a hardware-based attack that exploits the charge leakage vulnerability in DRAM cells. DRAM memory is composed of rows of capacitors that store bits of data. Over time, cells naturally leak charge, but if certain rows are accessed (”hammered”) frequently, they can induce bit flips in adjacent rows due to electromagnetic interference or charge disturbance. This phenomenon is known as ”row disturbance error” and was first demonstrated in [12]. Attackers repeatedly access two memory rows adjacent to a target row, rapidly opening and closing them to induce a bit flip in the victim row. The loops used to hammer the rows involve legitimate memory instructions, but their pattern creates an abnormal activation rate of DRAM rows. The critical parameter here is the Rowhammer Threshold (RH-TH), which refers to the minimum number of row activations required to induce a bit flip, typically between 139K and 500K activations within a refresh interval.

2) Spectre: Spectre is a class of speculative execution attacks that exploit modern processors' performance optimization features to access and leak sensitive information across security boundaries [2]. At its core, Spectre manipulates branch prediction and speculative execution to transiently execute instructions that would not occur during correct program execution, thereby exposing data that should remain inaccessible.

The attack works by mistraining the processor's branch predictor to speculatively execute code along a mispredicted path. During this speculative window, instructions can access memory outside of their permitted context. Although the results of speculative execution are eventually discarded, their side effects (particularly on the cache) are not. By measuring cache access times, an attacker can infer which memory locations were accessed, effectively exfiltrating sensitive data.

Spectre attacks are typically carried out through tight instruction sequences crafted to manipulate speculative behavior and measure resulting microarchitectural states. They do not rely on software vulnerabilities in isolation, but rather on fundamental architectural features such as out-of-order and speculative execution pipelines. This makes them broadly applicable across different architectures and difficult to mitigate completely without significant performance costs.

Several variants of Spectre have been developed, targeting indirect branches, return addresses, and store-to-load forwarding, among others. These variants underline the pervasiveness of speculative execution side channels and the need for both hardware and software-level defenses [2], [13]. We specifically focused on Spectre-v1, also known as Bounds Check Bypass [2], using the reference implementation provided in [14].

B. Xor Filter

XFs are a family of compact, probabilistic data structures used for membership testing, similar to Bloom filters [15] but offering better memory efficiency and faster lookups [16]. XFs use three hash functions to map each element to three positions in an array of fingerprints. During the build phase, the xor of fingerprints from all three positions equals the actual fingerprint of the element. At query time, an element's fingerprint is xored with the values at its hash-determined positions. If the result matches the stored fingerprint, the element is likely in the set.

Unlike Bloom filters, XFs use less space and have lower false positive rates. Additionally, they support constant-time queries and insertions with only three memory accesses, making them highly efficient for hardware-level filtering where latency and memory access are critical.

C. Count-Min Sketch

The CMS is a probabilistic, space-efficient data structure designed for frequency estimation in data streams. It provides fast and approximate counts of events, which is particularly useful in systems with limited memory and high throughput requirements [8].

A CMS consists of a two-dimensional array of counters and a set of independent hash functions. For each incoming element, the hash functions determine one counter in each row to increment. To estimate the count of an element, the algorithm takes the minimum value among the counters indexed by the hash functions. This approach bounds the overestimation error and supports constant-time updates and queries.

Unlike Bloom filters, CMS supports frequency tracking rather than membership testing, making it complementary to XFs in multi-stage processing architectures.

III. LAD-IXoC

This paper proposes LAD-IXoC, a hardware-level system for detecting loop attacks in DRAM memory, based on an integrated data structure composed of an XF and a CMS. The key advantage of this system lies in its two-stage detection capability. In the first stage, it filters out the majority of address-instruction tuples that do not correspond to any installed program. If no attack is detected at this point, the second stage checks whether the counters associated with the tuple exceed a predefined threshold. If they do, an alarm is triggered, indicating a potential loop attack.

A. Construction

The system leverages the integrated data structure introduced in [17], which combines the capabilities of XFs and CMS to perform both lookup and frequency estimation operations using a single memory access. The construction of the filter begins by calculating the memory required to store N address-instruction tuples and their associated counters. For an XF with an Undetected Attack probability of 2^{-r}, approximately 1.23×N rows of r bits are needed. Then, using three independently distributed hash functions (h_1, h_2, and h_3), all legitimate address-instruction tuples are mapped to three positions within the system, as illustrated in Fig. 1. The three CMS counters corresponding to each tuple are initialized to zero.

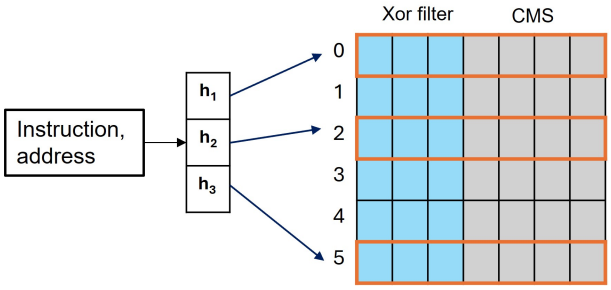


Fig. 1: Access to the integrated Xor Filter with CMS

B. Detection

Once all address-instruction tuples have been registered in the system, the detection process can be activated. The first stage aims to determine whether a given tuple belongs to any legitimate program currently in execution. To achieve this, the tuple is mapped, using the same three hash functions selected during the construction phase, to three positions in the filter.

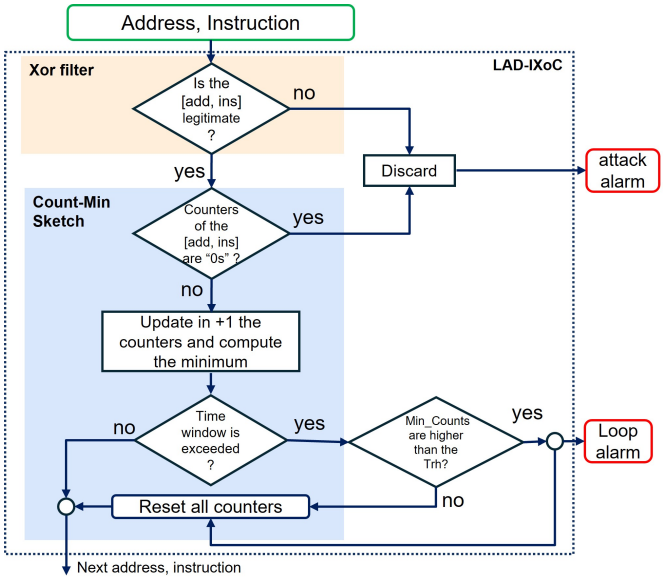


Fig. 2: Flowchart of the proposed attack detection process

The information stored at these positions is then retrieved: the first f bits are used by the XF, while the subsequent c bits are reserved for the CMS counters, which will only be used if the second stage is reached.

Next, a bitwise xor operation is performed on the three f-bit blocks: $f_1 \oplus f_2 \oplus f_3$. If the result matches the fingerprint associated with the tuple, defined during system construction, then the tuple is considered legitimate. Otherwise, the instruction is identified as illegitimate and an alert is triggered, signalling a potential attack.

Because the use of an XF in the first stage and the detection of legitimate loops entails a small undetected attack probability (for example < 0.004 for $r = 8$), a second-stage verification is required. This is carried out using the CMS counters accessed in the first stage: c_1, c_2, and c_3. The counters are incremented by 1, and the minimum of them is selected. If, after a predefined time window, this minimum counter exceeds a given threshold, an alarm is triggered to indicate the detection of an anomalous loop. Fig. 2 illustrates the system's detection flow diagram.

It is important to note that the system does not attempt to classify the type of attack–whether it is caused by Rowhammer, Spectre, or merely an irregular loop of legitimate instructions. The precise identification and mitigation of such attacks are beyond the scope of this work.

TABLE I: Benchmarks used

Benchmark	# Static instructions	# Executed instructions
Coremark	1290	73295
Median	1011	95147
Matrixmul	181	1.23E+05
Rsort	4460	71474
SHA	490	90139

IV. Experimental Evaluation and Results

We evaluated our proposal through Java-based experiments. Table I summarizes the programs used in our test environment,

TABLE II: Range of Thresholds and time Windows considered

Run 100 times for each pair (TH, TW)						
Threshold (TH)	25	50	75	100	125	150
Time Window (TW)	50	100	150	200	250	300

including the number of instructions in each and the subset of instructions executed. Table II presents the combinations of threshold values and time windows considered to evaluate the performance of our system. These parameters are used to measure the probability of undetected attacks as well as the number of false alarms generated by the system. Rowhammer and Spectre attacks are used as reference cases.

For the experiments, we relied on the BOOM RISC-V toolchain [14]. Three scenarios were considered to test the integrated filter. In the first, only legitimate programs are executed, representing an attack-free environment. The second scenario includes only Rowhammer and Spectre attacks. The third scenario reflects a realistic setting, where both legitimate instructions and attacks (Rowhammer and Spectre) are present.

1) Scenario 1: In the first scenario, since no attacks are present, only repeated instructions from legitimate programs are counted. Whenever these repetitions exceed the defined thresholds, an alarm is triggered. As there are no actual attacks, these alarms are considered false alarms. We then compute the average number of undetected attacks of the 5 programs presented in Table II.

Figure 3 a) shows that threshold values below 50 result in a higher number of false alarms, even across different time window configurations. The time window defines the interval at which alarms are reported to the system and the counters are reset. In the worst-case scenario, false alarms remain below 1,000. This occurs because lower threshold values increase the likelihood of legitimate loops exceeding the threshold.

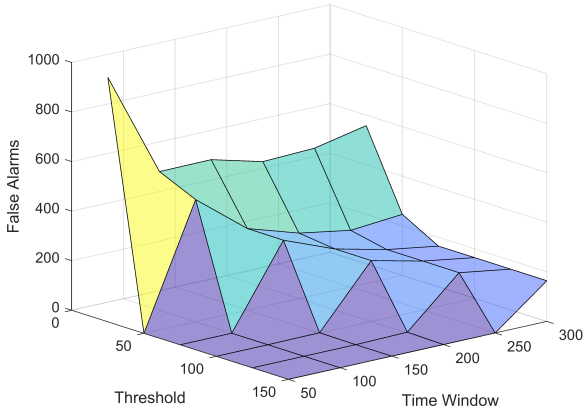
Figure 3 b) displays the estimation error introduced by the CMS. Notably, this error increases only in cases where the threshold is particularly low.

2) Scenario 2: Only attacks running: Figure 4 shows the probability of undetected attacks in the first stage (corresponding to the XF) and the attacks detected in the second stage (corresponding to the CMS): Rowhammer results are shown in (a) and (b), and Spectre results in (c) and (d). The results for both attacks demonstrate that the majority of attacks are filtered in the first stage. For those that reach the second stage, the CMS counters successfully identify them as attacks when the threshold is below 100. For higher threshold values, the system no longer considers these repetitions as attacks.

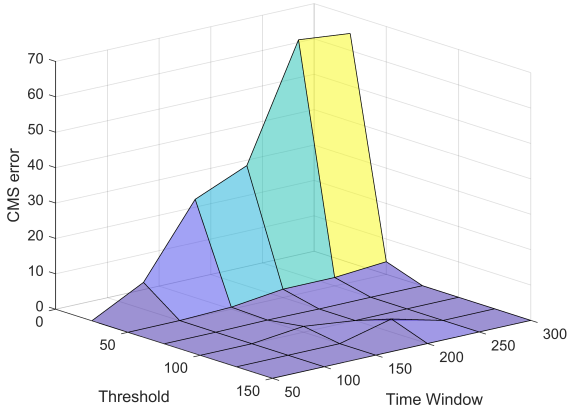
3) Scenario 3: safe instructions and attacks running: Figure 5 presents the results for Scenario 3, including both Rowhammer and Spectre attacks. In both cases, the percentage of undetected attacks remains below 0.4%, consistent with theoretical expectations. For attack detection in the CMS, the behavior observed in Scenario 2 applies: when the threshold is set below 100, all attacks are successfully detected.

A. Memory overhead comparison

In this section, a comparison between the proposed mechanism LAD-IXoC and other similar approaches that mitigate some loop attacks, specifically rowhammer, is presented. Table III shows the memory overhead required by each proposal to minimize the rowhammer threshold for CoMet, QPRAC, START and LAD-IXoC.



(a)



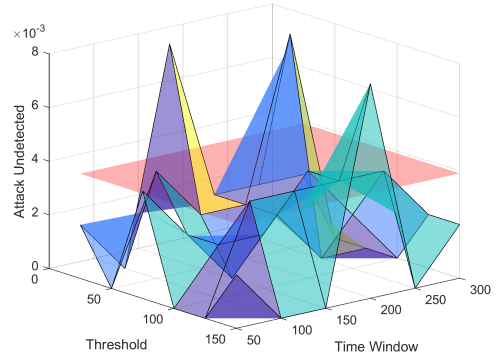
(b)

Fig. 3: Scenario 1: a) Average false alarm number b) Average error estimation of CMS

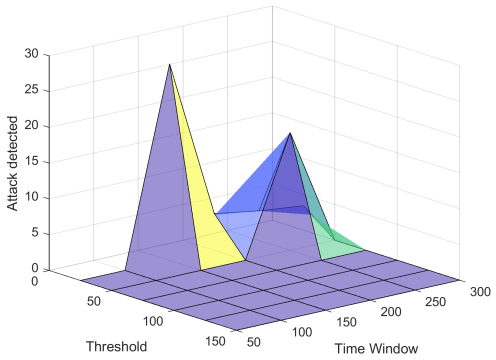
TABLE III: Memory overhead per bank of 128K rows

	Memory	KB
CoMet	SDRAM/CAM	3.2
QPRAC	SDRAM	112
START	SRAM/LLC	8
LAD-IXoC(simulated)	SDRAM	158

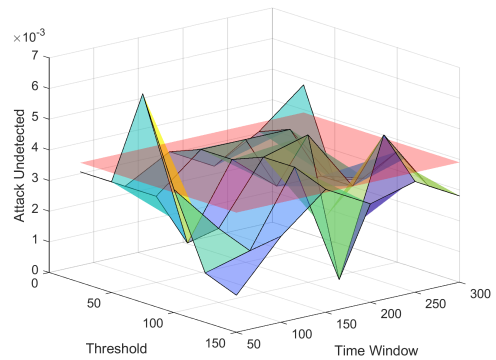
More specifically, when normalized to a bank of 128 K rows, START leverages the Last-Level Cache (LLC) on demand and requires only 8 KB of dedicated SRAM per bank for the most frequent activations. This footprint is modest—about 6% of a 128 KB bank—and benefits from ultra-low access latency, but under sustained high activation rates the limited number of cache-resident counters can be evicted, risking undetected disturbance errors. QPRAC, by contrast, embeds one full byte of counter state per row directly in DRAM, resulting in 112 KB of SDRAM overhead per bank. Although



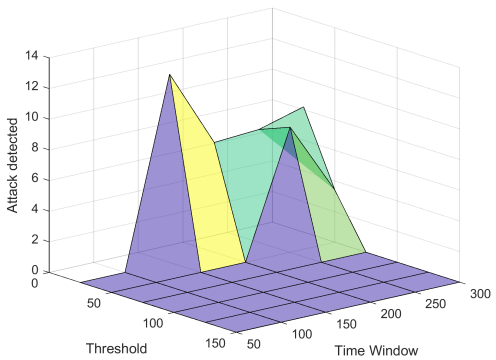
(a)



(b)

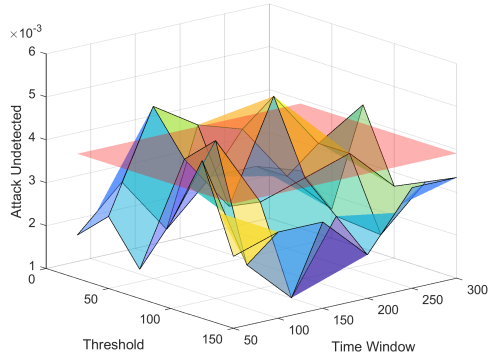


(c)

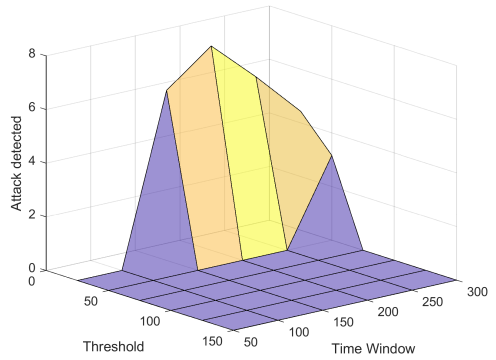


(d)

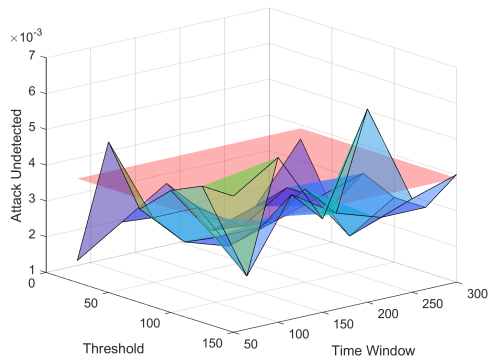
Fig. 4: Scenario 2: Probability of undetected attack and number of attacks detected for Rowhammer in a) and b); and for Spectre in c) and d)



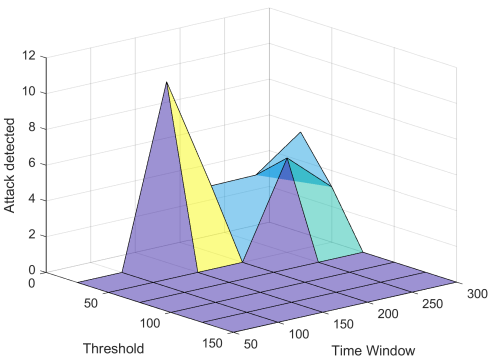
(a)



(b)



(c)



(d)

Fig. 5: Scenario 3: Probability of undetected attack and number of attacks detected for Rowhammer in a) and b); and for Spectre in c) and d)

this design shifts all additional storage off-chip and avoids consuming on-chip SRAM, it consumes nearly the entire bank's capacity and incurs extra DRAM-access latency on every activation.

CoMet introduces a compact hybrid of a Count-Min Sketch (CMS) implemented in SRAM and a small Content Addressable Memory (CAM) for row tags, totaling just 3.2 KB per bank. This low on-chip memory supports a Rowhammer threshold of 125 with only $\approx 2.5\%$ footprint and very fast lookups, at the cost of approximate counting and the silicon complexity of a CAM. Finally, our proposal LAD-IXoC defines an upper bound of 158 KB per bank for the counters (one byte per row) if every row were to pass the xor prefilter. In practice, the XF rejects the vast majority of benign rows, so the average memory usage is far lower, while the combined XF + CMS structure still enforces a threshold of 80 without resorting to fixed priority evictions. This dynamic filtering minimizes the likelihood that a malicious row simply slips through an eviction tail, making LAD-IXoC robust even under heavy loads.

V. Analysis and Conclusions

Our two-stage detection framework with an XF and a CMS enables the use of similar, per-tuple thresholds to competing schemes while still maintaining undetected attack probabilities below 0.4% for all tested configurations.

By combining lightweight filtering and frequency counting, our system not only flags anomalous execution loops but also provides an on-the-fly estimate of loop frequency. This dual functionality is critical to detect benign "hot loops" (e.g., tight kernel routines) and attack-induced repetition, which many single-stage filters cannot make without significant memory overhead or off-chip support.

A major advantage of our approach is that no per-attack metadata (e.g., trace logs or instruction histories) needs to be maintained. All relevant information, fingerprints, and frequency counts are stored implicitly within the XF/CMS structure itself. This yields an extremely compact footprint, easily implementable alongside existing DRAM controllers without requiring additional on-chip SRAM blocks for logs or micro-architectural buffers.

Our detection module can seamlessly integrate with current mitigation techniques (e.g., loop throttling, row hammer refresh protocols, or speculative-execution fences). Once an anomalous loop is signaled, higher-level firmware or OS routines can apply targeted countermeasures, such as dynamic instruction scheduling, cache invalidation, or selective memory refresh, to mitigate the attack before data corruption or privilege escalation occurs.

Overall, the proposed two-stage architecture offers a practical balance between detection accuracy, resource efficiency, and operational simplicity. It outperforms many threshold-based designs using only three memory accesses (and thus faster reaction times) without incurring a prohibitive rate of undetected attacks. Future work will implement our solution in hardware prototypes to validate performance and power overhead.

Acknowledgment

This work was supported by the FUN4DATE (PID2022-136684OB-C22), SMARTY (PCI2024-153434), and ITACA (PDC2022-133888-I00) projects funded by the Spanish Agencia Estatal de Investigacion (AEI), TUCAN6-CM (TEC-2024/COM-460), funded by CM (ORDEN 5696/2024) and by the European Commission through the Chips Act Joint Undertaking project SMARTY (Grant 101140087).

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 *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014.

[2] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[3] J. Woo *et al.*, " QPRAC: Towards Secure and Practical PRAC-based Rowhammer Mitigation using Priority Queues ," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2025.

[4] K. Arıkan *et al.*, "Processor security: Detecting microarchitectural attacks via count-min sketches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 7, pp. 938–951, 2022.

[5] F. N. Bostanci *et al.*, "Comet: Count-min-sketch-based row tracking to mitigate rowhammer at low cost," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 593–612.

[6] A. Jaleel *et al.*, "Pride: Achieving secure rowhammer mitigation with low-cost in-dram trackers," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[7] A. Saxena and M. Qureshi, "Start: Scalable tracking for any rowhammer threshold," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 578–592.

[8] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[9] Y. Kim *et al.*, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020. [Online]. Available: $https://people.inf.ethz.ch/omutlu/pub/rowhammer-revisited_isca20.pdf$

[10] W. Zhang *et al.*, "Counter pools: Counter representation for efficient stream processing," *arXiv preprint arXiv:2504.11235*, 2025. [Online]. Available: https://arxiv.org/abs/2504.11235

[11] M. Huang *et al.*, "Enhancing resiliency of sketch-based security via lsb sharing-based dynamic late merging," *arXiv preprint arXiv:2503.08112*, 2025. [Online]. Available: https://arxiv.org/abs/2503.08112

[12] Y. Kim *et al.*, "Flipping bits in memory without accessing them: an experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. IEEE Press, 2014.

[13] C. Canella *et al.*, "A systematic evaluation of transient execution attacks and defenses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 249–266.

[14] C. Celio, D. A. Patterson, and K. A., "BOOM: A BERKELEY Out-of-Order Machine," in *Proceedings of the First Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2015. [Online]. Available: https://github.com/riscv-boom

[15] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, jul 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[16] T. Mueller Graf, D. Lemire, "Xor filters: Faster and smaller than bloom and cuckoo filters," *CoRR*, vol. abs/1912.08258, 2019. [Online]. Available: http://arxiv.org/abs/1912.08258

[17] R. Martínez *et al.*, "Supporting dynamic insertions in xor and binary fuse filters with the integrated xor/bif-bloom filter," *IEEE Transactions on Network and Service Management*, vol. 21, no. 3, pp. 3068–3079, 2024.