NVM-Flip: Non-Volatile-Memory BitFlips on the System Level

Felix Staudigl
staudigl@ice.rwth-aachen.de
RWTH Aachen
Aachen, NRW, Germany

Jan Philipp Thoma
jan.thoma@rub.de
Ruhr-University Bochum
Bochum, NRW, Germany

Christian Niesler
christian.niesler@uni-due.de
University of Duisburg-Essen
Essen, NRW, Germany

Karl Sturm
sturm@ice.rwth-aachen.de
RWTH Aachen
Aachen, NRW, Germany

Rebecca Pelke
pelke@ice.rwth-aachen.de
RWTH Aachen
Aachen, NRW, Germany

Dominik Germek
dominik.germek@de.bosch.com
Corporate Research Robert Bosch GmbH
Germany

Jan Moritz Joseph
joseph@ice.rwth-aachen.de
RWTH Aachen
Aachen, NRW, Germany

Tim Güneysu
tim.gueneysu@rub.de
Ruhr-University Bochum
Bochum, NRW, Germany

Lucas Davi
lucas.davi@uni-due.de
University of Duisburg-Essen
Essen, NRW, Germany

Rainer Leupers
leupers@ice.rwth-aachen.de
RWTH Aachen
Aachen, NRW, Germany

ABSTRACT

Emerging non-volatile memories (NVMs) are promising candidates to substitute conventional memories due to their low access latency, high integration density, and non-volatility. These superior properties stem from the memristor representing the centerpiece of each memory cell and is branded as the fourth fundamental circuit element. Memristors encode information in the form of its resistance by altering the physical characteristics of their filament. Hence, each memristor can store multiple bits increasing the memory density and positioning it as a potential candidate to replace DRAM and SRAM-based memories, such as caches.

However, new security risks arise with the benefits of these emerging technologies, like the recent NeuroHammer attack, which allows adversaries to deliberately flip bits in ReRAMs. While NeuroHammer has been shown to flip single bits within memristive crossbar arrays, the system-level impact remains unclear. Considering the significance of the Rowhammer attack on conventional DRAMs, NeuroHammer can potentially cause crucial damage to applications taking advantage of emerging memory technologies.

To answer this question, we introduce NVgem5, a versatile system-level simulator based on gem5. NVgem5 is capable of injecting bitflips in eNVMs originating from NeuroHammer. Our experiments evaluate the impact of the NeuroHammer attack on main and cache memories. In particular, we demonstrate a single-bit fault attack on cache memories leaking the secret key used during the computation of RSA signatures. Our findings highlight the need for improved hardware security measures to mitigate the risk of hardware-level attacks in computing systems based on eNVMs.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Security and privacy** → **Hardware attacks and countermeasures**; *Systems security*.

KEYWORDS

ReRAM, NeuroHammer, disturbance errors, thermal corsstalk, eNVM

ACM Reference Format:
Felix Staudigl, Jan Philipp Thoma, Christian Niesler, Karl Sturm, Rebecca Pelke, Dominik Germek, Jan Moritz Joseph, Tim Güneysu, Lucas Davi, and Rainer Leupers. 2024. NVM-Flip: Non-Volatile-Memory BitFlips on the System Level. In *Proceedings of the 2024 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SaT-CPS '24), June 21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3643650.3658606

1 INTRODUCTION

Emerging non-volatile memories (NVMs), such as spin-torque transfer memory (STT-RAM/MRAM)[36], phase-change random access memory (PCRAM)[14], or redox-based random access memory (ReRAM)[59], are disruptive technologies due to their unique characteristics of low access latency, high integration density, and non-volatility [21, 48]. Due to their various benefits NVMs are considered to replace SRAMs [1], eFlash [8] and SSD memories [17], while also enabling novel computing paradigms. CPS (Cyber-Physical Systems) will likely be the first to receive this new memory technology as the memory density in the first years will not reach the requirements of PCs or servers. As CPS or embedded systems are usually less memory intensive, they will likely implement emerging

SaT-CPS '24, June 21, 2024, Porto, Portugal

ACM ISBN 979-8-4007-0555-7/24/06
https://doi.org/10.1145/3643650.3658606

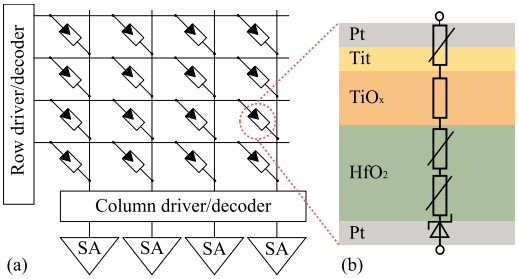

Figure 1: ReRAM structure composed of (a) a crossbar array and (b) the memristive device.

non-volatile memory first. Current research such as [10] already describes the benefits of resistive memories with regard to energy efficiency and performance. Despite the promising traits of NVMs, their characteristics introduce new security threats, such as side-channel and information leakage attacks [19, 24, 63]. These passive attacks aim to leak information by observing system parameters. However, active attacks similar to Rowhammer [34] are also possible. Staudigl et al. [47] proposed an active attack on ReRAMs, called NeuroHammer.

Hardware-enabled attacks, such as Rowhammer, Spectre, and Meltdown, have exposed the potential to disrupt essential computing infrastructure. As a result, research on hardware security vulnerabilities and countermeasures has become indispensable [38]. Hardware security primarily identifies design characteristics that may be leveraged to seize control of the entire system [44]. Due to the inflexible nature of integrated circuits (ICs), these vulnerabilities are challenging to address and remain throughout the IC's lifetime. For instance, the Rowhammer attack capitalizes on disturbance errors in dynamic random access memories (DRAMs) to intentionally enable an adversary to modify the memory's content [23]. Project Zero by Google showed that exploiting Rowhammer can result in privilege escalation attacks [32] or denial of service attacks in cloud systems [5].

ReRAMs utilize crossbar structures that contain memristive elements connecting the word line to the bit lines. Fig. 1 (a) illustrates the typical crossbar array structure of NVMs, which connects each memory cell via bit and word lines. The row/column decoder, drivers, and sense amplifiers (SA) are responsible for reading/writing a certain memory cell by applying a respective voltage pulse to the bit and word line.Fig. 1 (b) depicts a typical material stack of a memristive device and its electrical equivalent circuit.

NeuroHammer demonstrated that an attack comparable to Rowhammer still functions with memristor-based memories [47]. This attack enables malicious actors to flip bits of adjacent memory cells deliberately. The attack takes advantage of disturbance errors based on thermal crosstalk between memory cells. In particular, ReRAM represents a promising alternative for DRAMs and could potentially be used as cache memory in the future [25, 27]. Hence, intentional bit-flips are no longer limited to main memories (Rowhammer) but also potentially threaten cache memories (NeuoHammer).

CPS systems range from small embedded devices with limited capabilities to devices capable of running full operating systems. Among the more powerful systems for CPS, RTLinux is very popular, which is an extension of the Linux kernel that enables real-time behavior [43]. Rowhammer-style attacks can be performed from isolated processes or RTOS threads. Thus, CPS systems with NVM memory will be vulnerable to NeuroHammer attacks from the system level. However, there has been no evaluation yet as CPS systems with NVM have not hit the industrial market yet. In the past, security has been an afterthought; that is why there have been so many attacks and (hardware) countermeasures in upgrades of DRAM modules. DDR4, for example, has in-build hardware Rowhammer protection, which was circumvented by Frigo *et al.* [13]. We want to investigate Neurohammer attacks from the system level to help develop secure memory modules for future CPS systems.

Contribution: we investigate the impact of deliberate bit-flips on ReRAMs on a *system level*. In particular, we focus on the implications for the ReRAM-based main and cache memory of the system. We propose an extension to the gem5 simulator, called **NVgem5**, capable of simulating ReRAM-based memories. The simulator emulates the NeuroHammer attack by injecting bit-flips based on the access patterns of the adjacent cells. The main contributions of this paper are:

- The implementation of NVgem5, modeling ReRAM-based main and cache memory based on the gem5 simulator.
- The execution of a typical Rowhammer attack scenario on the system's main memory to verify the simulation platform.
- The execution of the first-ever attack on ReRAM-based caches by intentionally triggering bit-flips to leak the secret key of RSA during signature generation.
- A thorough discussion on countermeasures to mitigate NeuroHammer attacks on ReRAM-based memories.

2 BACKGROUND

This section summarizes the related work in the field of hardware security attacks on emerging NVMs. In addition, it describes the background of ReRAMs and the NeuroHammer attack.

2.1 Related Work

Emerging NVMs suffer from hardware security vulnerabilities due to their unique physical characteristics [12]. In particular, side-channel attacks use these characteristics to leak critical information. Khan et al. [19] propose a side-channel attack, which takes advantage of the asymmetric read/write currents of ReRAM devices. The write operation of the victim injects supply noise propagating to the adversary's memory region. Due to the much lower read latency, the adversary can sense the Hamming weight of the victim's data by analyzing the read failures within the adversary's memory space. However, this attack requires exhaustive testing of all possible patterns, locations, and has to consider bit-to-bit variations. In contrast, Krautter et al. [24] use the self-terminating writing scheme to expose information to an adversary. Self-terminating writing schemes aim to increase the write latency by completing the write sequence as soon as the intended state is reached. Consequently, there is a strong correlation between the duration of the write process and the information being written. The adversary can recover the secret purely from software.

NVMs are also susceptible to differential power analysis (DPA) attacks, as shown in [12]. DPA attacks are considered severe threats to system security because of their noninvasive nature. To improve the system's resilience, Dodo et al. [12] propose a novel STT-MRAM

bit-cell design, 500× more resilient than the standard design. In addition, Yang et al. [63] suggest a mitigation technique to prevent replication attacks on memristor-based computing systems. NVMs are particularly vulnerable to replication attacks due to their non-volatility property. Yang et al. utilize the obsolescence effect to reduce the accuracy of the outputs for any unauthorized user. Furthermore, the countermeasure is still effective even if the attacker knows the defense mechanism.

Khan et al. [20] investigate the impact of ground bounce effects in STT-RAMs. Due to high write currents and long write latencies, STT-RAMs suffer from ground bounce, which eventually propagates to adjacent cells. Through simulation, the authors exhibit that repeatedly writing to a single cell may cause a bit-flip in nearby cells. To the best of our knowledge, this work investigates for the first time the impact of deliberate bit-flips in ReRAMs on the security of a full-fledged system. In particular, our investigation demonstrates the significant consequences of intentional bit-flips in ReRAM cache memories, exhibiting a pristine attack surface for novel computing paradigms.

Considering the impact of emerging memory technologies, various simulation platforms have been proposed to study reliability, computational efficiency, and architectures. gem5-ndp [56] describes a near-data processing (NDP) simulator based on gem5 [2] capable of modeling architectures and estimating performance values. On the one hand, the simulator allows easy integration of novel architectures by offering a well-defined programming interface. On the other hand, gem5-ndp supports only the ARM ISA, limiting its feasibility for hardware security analysis. Likewise, PIMSim [61] proposes a simulation framework offering three different modes: fast, instrumentation-driven, and full-system. The platform integrates established memory simulators, such as DRAMSim2 [40], HMCSim [26], and NVMain [37], to support hybrid memory simulations. Mao et al. [31] investigate programming strategies to optimize the energy efficiency and reliability of ReRAMs. The authors use gem5 [2] and CACTI [30, 53] to estimate read/write/leakage energy consumption and the latency of the ReRAMs. Conclusively, various simulation frameworks have been proposed to investigate the merits and shortcomings of emerging memory technologies on an architectural and system level. However, none of these platforms is equipped to investigate hardware security vulnerabilities, in general, and NeuroHammer-based bit-flips, in particular. Hence, we present NVgem5, a simulation platform capable of injecting NeuroHammer-based bit-flips in ReRAMs.

2.2 Rowhammer

Emerging memories have the potential to replace existing memory technologies like DRAM or SRAM due to their advantageous features, including non-volatile memory storage. Memristive memory, for instance, eliminates the need for refresh cycles, thereby enhancing energy efficiency. However, DRAM chips are susceptible to a severe hardware phenomenon called Rowhammer, where a specific memory location is repeatedly accessed, leading to a bit-flip in nearby memory cells due to electrical interaction, causing the capacitor in the DRAM cell to leak its charge [22].

Despite having no capacitors to leak charge, memristive memory is vulnerable to high temperatures, which can be attained through repeated memory access. As a result, Rowhammer attack patterns can be applied to memristive and classic DRAM chips [47]. The intentional bit-flip can be exploited to undermine secure cryptographic keys [39] or gain access to specific data [55]. Researchers have successfully demonstrated that these attacks can be carried out remotely, particularly affecting cloud providers that rely on large clusters of vulnerable servers [6, 28, 50]. Even DRAM chips with error correction are susceptible to Rowhammer attacks [7].

The correct access patterns are a critical factor in triggering these vulnerabilities. DRAM and memristive memory will likely be arranged in rows and columns, and depending on the number and location of rows accessed near a victim row, the attack pattern is referred to as single, double-sided, or n-sided [13]. Direct memory access poses several obstacles, particularly cache memories, temporarily storing frequently used data to improve access times. The *cflush* instruction represents one way to overcome this obstacle by flushing the data out of the cache [16]. However, this instruction may not be available on other Instruction Set Architectures (ISAs), making other methods like eviction sets a viable alternative [57]. An eviction set is a group of addresses that map to the same row of cached entries and access them one after the other, filling the cache row, replacing the older data, and clearing the victim's data from the cache. We provide detailed information about the working principles of eviction sets in Section 2.5.

As bit-flips in DRAMs are primarily a hardware issue, preventing them entirely at the software level is challenging. Although some software solutions have been proposed, they usually rely on hardware features to provide protection. The latest software-based defense, known as *Copy-on Flip* [11], utilizes an interrupt triggered by ECC RAM to implement countermeasures. In addition to these software solutions, manufacturers have implemented defense mechanisms directly into the hardware. One such mechanism is called *Target Row Refresh (TRR)* [18], which aims to identify the memory rows being targeted by Rowhammer attacks and refresh those memory cells before a bit-flip can occur. Nevertheless, the implementation of TRR is highly dependent on the vendor [18], and research has indicated that even TRR can be bypassed [13].

2.3 Resistive RAM (ReRAM)

ReRAM describes an emerging memory technology taking advantage of memristive devices. In general, a memristive device stores the information internally in the form of resistance. The internal state is programmed via suitable voltage pulses applied to the two terminals of the device. Based on the voltage pulse, the so-called memristor switches between a high resistive state (HRS) and a low resistive state (LRS) for a binary switching device.

Fig. 1 illustrates a crossbar structure that connects the memristors via vertical bit lines (BL) and horizontal word lines (WL). The rows and columns of the crossbar structure are connected to decoder/driver circuitry, which drives the necessary pulses and multiplexes them to the respective WL and BL. For example, to read a single device, V_{read} is applied to the WL, while the corresponding BL is set to GND. Due to the structure of the crossbar, all non-selected cells must be protected from unintentional sneak-path currents influencing the read operation. Hence, the so-called $V/2$ scheme is used to set all remaining lines to $V_{read}/2$. Consequently,

the voltage drop over non-selected devices results in $|V_{read}/2|$ compared to $V_{read}/2$ over selected devices. Selector devices have been proposed to isolate the memristor from the crossbar array to omit the influence of sneak-path currents. The most common configuration of a so-called 1T1R crossbar uses a transistor connected in a row with the memristor.

2.4 NeuroHammer

In 2022, Staudigl et al. [47] introduced a new security attack named NeuroHammer, targeting memristive crossbar array memories, enabling attackers to flip bits like the Rowhammer attack. This attack leverages disturbance errors resulting from thermal crosstalk. Prior research conducted by Von Witzleben et al. [58] showed that the switching kinetics of redox-based memristive cells change significantly based on the temperature of the filament. The study concluded that the switching time reduces by over five orders of magnitude with increased filament temperature. Fig.2 illustrates the internal working principles of the attack, with the red cell symbolizing the attacked cell and the blue cell representing the target cell. The attack works as follows:

(a) **Hammering:** Initially, the red cell should be switched to LRS, which maximizes the current through the cell. In order to intentionally elevate the filament temperature of the target cell, the attacker needs to apply pressure to the compromised cell continuously. As a result, the blue cell is subjected to repetitive stress caused by the $V/2$ scheme.
(b) **Temperature increase:** Due to the attacker's repeated access, the red cell undergoes temporary heating with each applied voltage pulse. In addition, the filaments in the blue cell experience an increase in temperature due to thermal crosstalk between the two cells. Meanwhile, the blue cell is subjected to a regular $V_{SET}/2$ voltage pulse, further contributing to its rising temperature.
(c) **Switching kinetics:** According to experiments conducted by von Witzleben et al. [58], the change in temperature affects the switching kinetics of the ReRAM device. Consequently, the device becomes more prone to gradually switching its internal state.
(d) **Bit-flip:** Over time, the blue cell gradually changes its internal state with each applied voltage pulse from the attacker, eventually resulting in a complete switch. By utilizing the thermal crosstalk and the $V/2$ scheme, the attacker can flip a bit without ever directly accessing the ReRAM cell.

To validate the existence of NeuroHammer, Staudigl et al. proposed a simulation methodology combining crossbar and circuit-level simulations. First, a crossbar-level simulation was conducted to examine the thermal crosstalk within crossbar structures and generate a set of thermal coupling coefficients, referred to as *alpha values*. Second, a circuit-level simulation used these alpha values to simulate the behavior of a ReRAM tile, taking into account the coupling effects. The results demonstrated that thermal coupling significantly impacts the characteristics of the memristive crossbar, enabling adversaries to flip bits intentionally.

The authors verified the attack based on crossbar- and circuit-level simulations. Therefore, the system-level impact of NeuroHammer still needs to be clarified. Furthermore, ReRAM has a much broader application space than DRAM, taking into account its potential to replace memories on different hierarchies [27].

2.5 Caches

Caches are small chunks of memory situated physically close to CPU cores. These days, volatile SRAMs are commonly used for building caches. Caches are utilized for storing regularly accessed data to conceal the access latency of the main memory. Modern CPUs generally have three cache levels: the first-level cache (L1) is the smallest and fastest, the second-level cache (L2) is larger and slower, and the last-level cache (LLC) is even larger but also significantly slower. Additionally, L1 and L2 caches are specific to each CPU core, while LLC is shared among all cores.

Inclusive caches contain copies of all data stored in the L1 and L2 caches, enabling faster lookup times as the private caches of other cores do not need to be queried during access. Inclusive caches are extensively used in both consumer and server devices. However, non-inclusive and exclusive cache solutions exist that do not duplicate entries of private caches in the shared LLC. These solutions necessitate a directory structure or a snoop-based setup. However, the latter generates significant traffic on the memory bus, making it less desirable [45].

Caches are primarily designed as a set-associative structure because access latency is critical to the system's performance. In this structure, any memory address only maps to a few possible cache entries (a set). During access, the cache controller only needs to search these entries for the data to determine whether the data was cached (cache hit) or not (cache miss). This structure significantly reduces the lookup complexity and latency compared to fully-associative caches, where an address may be mapped to any cache entry [62]. The set-associative structure can be visualized as a table with 2^s sets (rows) and w ways (columns). To map a given address to this table, s bits of the physical address are used, which determine the cache set. The remainder of the address, except for some offset bits that determine a shift within the stored data, is stored along with the data as a tag. During access, the tag of the requested address is compared to all tags stored in the given cache set, and if there is a match, the data is returned. If there is no matching tag, i.e., a cache miss occurred, the data is requested from the memory side of the cache. Due to the temporal locality of most workloads, i.e., recently accessed addresses are likely to be reaccessed, the requested data will be loaded into the cache after a cache miss and replace some other data. The entry to be replaced is determined by the replacement policy, such as least-recently-used [49].

Caches, shared by multiple components and have distinguishable response times for cache hits and misses, have been exploited in various cache attacks. Flush-based attacks [15, 64] are executed when the attacker and the victim share some read-only data section. The attacker flushes the data from the cache and monitors the access pattern of the victim to reveal sensitive information. On the other hand, contention-based attacks [35, 54] do not require shared data but instead use the set-associative structure of caches. To take advantage of the set-associative structure, these attacks utilize eviction sets [29], a group of addresses that occupy all cache entries in a given set. By accessing the addresses within an eviction set, the attacker can replace all existing entries with the data from the

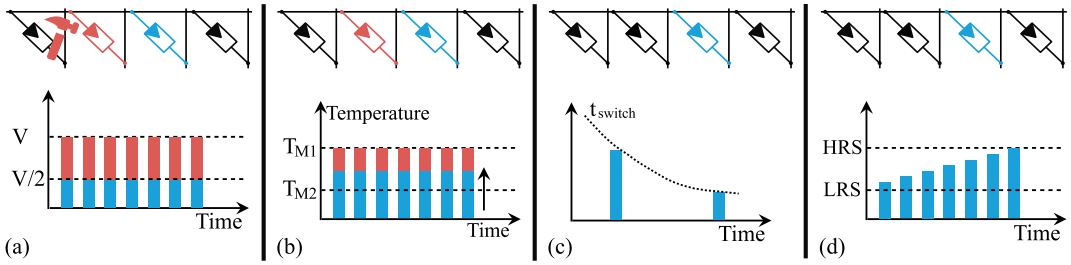

Figure 2: NeuroHammer attack: (a) hammering the attack cell (red), (b) temperature increase of the target cell (blue), (c) change of the target cell's switching kinetics, and (d) gradual increase of the target cell's resistance [47].

eviction set, thus monitoring the victim's subsequent cache accesses, similar to flush-based attacks. Efficient construction methods for eviction sets have been proposed in [46, 51].

3 METHODOLOGY

This section describes the simulation methodology consisting of NVgem5 and the threat model. Fig. 3 (a) shows our simulation platform. NVgem5 builds upon the architecture simulator gem5 [2], which models a processor including caches and main memory. The processor model simulates an out-of-order x86 processor clocked at 1GHz with L1 caches (instruction and data cache with 16 kB each), a unified 256 kB L2 cache, and 2 GB of main memory.

3.1 NVgem5

NVgem5 extends the gem5 simulator by a ReRAM-based memory implementation which can be configured as cache or main memory. Fig. 3 (b) depicts the general internal structure of the NVM module.

Main memory mode: The NVgem5 mimics the fundamental structure of DRAM memory. The lowest level represents the crossbar structure consisting of binary bit-cells. Each cell holds certain meta information to track the number of read/write accesses. If the configurable bit-flip threshold is exceeded, the adjacent cells are flipped based on a given pattern. The patterns are customizable and represent a matrix of probabilities. We use the experimental results from [47] to define a simple pattern that changes the adjacent cells above and below the target cell. All crossbars are combined to banks and chips, following the typical DRAM hierarchy. The memory controller translates the given address to determine the correct crossbar and rows, respectively.

Cache mode: In the cache mode, the memory controller acts as a standard SRAM cache controller. The same crossbar modules are used, which features the meta information property of each memory cell. To increase the performance of the memory model, we reduce the bit-cell complexity by stripping the pattern of its probability values. Hence, the pattern only consists of binary entries, which determines the adjacent cells affected by bit-flips. As expected, the memory controller intercepts read and write requests from the CPU to the main memory. The given address of the request is divided into the tag, the set index and the data offset. These three fields determine the resulting memory location to store/fetch the data to/from the crossbar.

3.2 Threat Model

The attacker is assumed to have the following capabilities:

(1) The attacker can run arbitrary code within his isolated user process.
(2) The attacker cannot manipulate data outside the virtual memory space of his process.
(3) The attacker has knowledge of the processor architecture and the memory hierarchy.

Similar to existing Rowhammer attacks on DRAM, the adversary intentionally triggers bit-flips within the victim's memory region by taking advantage of the NeuroHammer attack. To illustrate the impact of this attack on a system level, we perform two case studies.

(1) *Main memory:* Fig. 4a depicts our first attack scenario aiming to attack the main memory of the system. The adversary uses eviction sets to circumvent the L1/2 caches. This scenario mirrors the Rowhammer attack for DRAMs, which we use to verify our simulation platform.
(2) *Cache:* The second case study targets the L1 data cache. To the best of our knowledge, NeuroHammer is the only non-intrusive attack capable of actively faulting bits in ReRAM caches, yielding a completely new attack scenario. We showcase the danger of this attack by compromising the secret key during an RSA signature generation with a single attack trace.

4 RESULTS

In this section, we evaluate system-level attacks on ReRAM-based memory in our NVgem5 environment. First, we explore the traditional Rowhammer setting where vulnerable ReRAM is implemented in main memory. After that, we consider attacks on cache memory which has not previously been explored. We demonstrate a fault attack on RSA-CRT and leak the private key during signature generation. Moreover, we discuss further attack vectors on vulnerable cache memory.

4.1 Main Memory

In the following, we demonstrate the feasibility of the fault-attacks on ReRAM when deployed as main memory. Therefore, we use the proposed NVgem5 to simulate a system with 2 GB of ReRAM. The system is equipped with a single-issue in-order CPU and two levels of SRAM cache memory. In our experiment, we show that

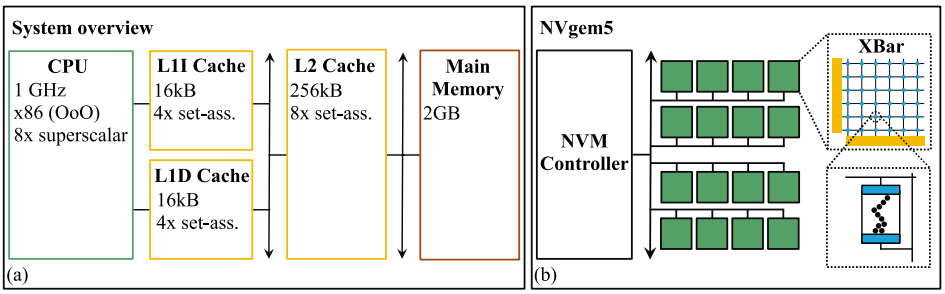

Figure 3: Simulation overview: (a) system components and (b) internal structure of NVGem5.

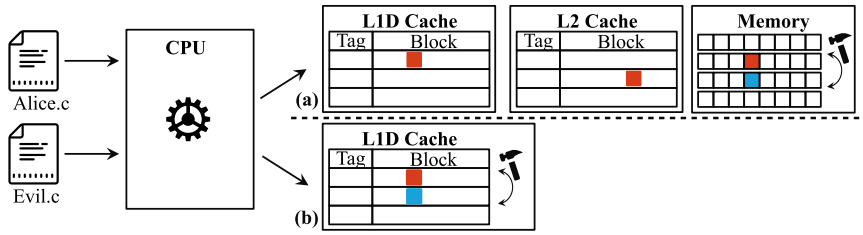

Figure 4: Attack scenario: (a) deliberately flipping bits in non-volatile main memory and (b) provoking cache bit-flip.

traditional Rowhammer attack patterns are transferable to ReRAM enabled systems.

Algorithm 1: ReRAM fault attack on main memory

```
Data: Target Address t, Hammer Count c
Addr h ← getAdjacentCrossbarAddr(t);
for i from 0 to (c − 1)/2 do
    h.data ← 0b0000...1...000;
    clflush(h);
    h.data ← 0b0000...0...000;
    clflush(h);
end
if v.data == 0 then
    return false;
else
    return true;
end
```

Algorithm 1 shows the pseudo-code of our attack which we used to determine the hammering threshold required to introduce observable bit-flips to the target memory crossbar. Therefore, the attacker first selects a target memory address at which they attempt to inject bit-flips. Then the attacker needs to obtain an adjacent memory address and repeatedly issue write accesses to it to trigger faults in the target crossbar. Since theoretically it is only feasible to inject 0⟶1 bit-flips, but not the other way around, we select a target memory address and initialize it with zeros. This allows us to observe any errors injected by the hammering process. Since our simulated system includes a two-level cache hierarchy with a write-back LLC, the attacker must bypass the cache to ensure that the data-writes actually reach the main memory and are not buffered in the LLC. For this, several well-documented solutions exist. Since our system operates on an x86 architecture, we can simply use the `clflush` instruction which forces the eviction of a given memory address. After each write to the hammering address, we use `clflush` to evict it from the cache. Since the evicted entry is modified (dirty) after the write-access, the data is written back to memory during the eviction. If the `clflush` instruction is not available, the attacker could alternatively use an eviction set or non-temporal write instructions to make sure that the changes buffered in the cache actually reach the main memory.

For the attack demonstration, we allocate memory in a crossbar adjacent to our target address. The feasibility of finding such adjacent memory lines in real-world environments has been demonstrated among others by Seaborn et al. [42]. The results of our experiment show a one-bit flip in the data located at the target address when executing the algorithm with more than 40 000 writes to the adjacent crossbar. Hence, the frequency that is achievable from system level is sufficient to trigger bit-flips in the simulated crossbar. We therefore conclude that the current struggles to overcome Rowhammer attacks on DRAM technology might continue even further into the future and even a technology switch to ReRAM will not necessarily mitigate this attack vector. For this reason, it is crucial to investigate potential attacks and countermeasures at this early stage.

4.2 Cache

We demonstrate the effects of an intentional bit-flip in the cache memory by leaking an RSA secret key from a co-located process during signature generation. Such attack are usually only feasible trough physical fault attacks, e.g., on embedded devices and smartcards. The attack described in the following is purely executed from software level and can therefore be exploited remotely. Moreover, due to the short-lived nature of cache memory, the attack does not leave any long-term traces in the CPU and is therefore especially hard to detect.

To attack the RSA secret key, we use the well-known fault attack on RSA-CRT [3]. We therefore implemented the RSA signature scheme using the Chinese Remainder Theorem (CRT). In a nutshell,

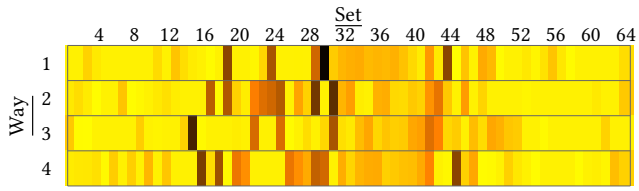

Figure 5: Write access pattern for the L1-Data cache during RSA signature generation without attacker.

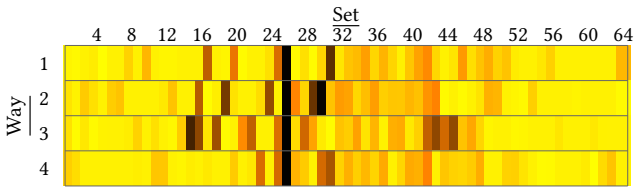

Figure 6: Write access pattern for the L1-Data cache during RSA signature generation with active attacker hammering the 26th cache set.

the CRT divides the computation of the RSA signature in modular residue classes of p and q. By faulting the computation in one of these module residues, only one part of the solution has a common prime factor with N, which allows factoring N efficiently by calculating the gcd of N and the faulty signature. To exploit this property using NeuroHammer, the attacker first needs to profile relevant cache sets where parts of the RSA computations are stored. Fig. 5 shows a heatmap of the L1-cache write-activity during the RSA signature generation. Noticeably, some cache sets are used much more often than others. Since the attackers aim is to introduce a single-bit fault, they are ideally looking for a cache set where one entry is heavily used while the others are rarely used. An example from Fig. 5 would be cache set 15. Since the attacker cannot simply observe the entire cache activity as we did in Fig. 5, they must use a different strategy to obtain this information. Therefore, the adversary can allocate a large chunk of data and access it. After the access, the data will be stored in the L1 cache and occupy many entries there. Then, the attacker triggers the signature generation and observes which entries are evicted by the computation. This process is repeated several times and the attacker learns which addresses are frequently evicted. Those entries have a high probability of colliding with data that is used during the computation of the RSA signature. After the attacker has learned potential target cache sets, they subsequently need an address in the adjacent cache set. If the attacker has the ability to use huge pages, they can arbitrarily select the cache set by changing the lower bits of the address within the huge page. However, even without huge pages, the attacker can easily obtain adjacent addresses. Therefore, they only need to invert the least significant bit (LSB) of the virtual address that is used for the set-indexing. The six LSBs of the virtual address are usually used to determine the offset within the 64 Byte cache entry. The following bits of the virtual address are then used to determine the cache set. Hence, flipping the seventh bit of the virtual address inverts the LSB of the set index and the resulting address will be adjacent to the original address in the cache. Alternatively, the attacker can choose an address with 64-Byte offset to the original address which will also result in an adjacent cache line.

Algorithm 2 shows the pseudo code used to trigger the fault during RSA signature generation in the cache. We assume the adversary and the victim are co-located on a CPU core (hyper-threading) and thus share the L1 cache. We discuss attacks in non-co-located environments later in this paper. The algorithm takes a target address as input. This address must map to a cache set that is used during the computation of the RSA signature. We found that in order to increase the success-rate of the attack, it is beneficial to hammer the entire cache set instead of a single entry. Therefore, the attacker needs to construct an eviction set for the hammering address. This can be achieved efficiently using either a top-down [46], or a bottom-up strategy [51]. Using an eviction set for the hammering has the advantage, that the attacker becomes independent of the caches' replacement policy. If the attacker only uses a single address for the hammering, the attacker must hope that the RSA-CRT data is allocated in the same cache way as the hammering address. The eviction set strategy on the other hand produces faults in all cache ways and therefore has a higher chance of hitting the correct data. However, with the additional faults, the probability of crashing the process increases as well although during our evaluation, we found that this rarely happens. Often the data is not crucial to the systems stability and/or is not used anymore. Crucially, the bit-flips will almost never be propagated outside the cache, since the entries are not marked as $dirty$ when a NeuroHammer attack changes the value.

Algorithm 2: ReRAM fault attack on RSA-CRT in the cache

```
Data: Target Address t
EvSet ev ← getEvSet(t − 64);
(e, N) ← getPubKey(victim);
sendSignReq(victim, msg);
while !sig = recvSignResp(victim) do
    for Addr a in ev do
        a.data ← 0b0000...1...0;
        a.data ← 0b0000...0...0;
    end
end
if sig.valid then
    return 0;
else
    return gcd(sig^e − msg, N);
end
```

Only entries that have been modified prior to the attack are written back to main memory and may contain faults. In this context, the profiling of frequently used cache sets by the attacker serves a double purpose: First, the attacker learns potential target sets, and secondly, the attacker evicts potentially modified cache entries and therefore reduces the risk of writing back faulted data after the attack.

Fig. 6 shows the write access pattern to the L1-data cache during the attack. In this case, the attacker used an eviction set cache set 26 to flip bits in the 27th set. In our evaluation environment, we found that the attacker can reliably introduce faults to the RSA signature generation, while not crashing the program due to random faults in other cache entries. Thus, we successfully calculated a prime factor of N, and recovered the RSA secret key.

Attacks on the LLC: Above, we conducted the attack on the L1 cache, which requires the attacker and the victim to share a processor core. While this is feasible due to hyper-threading, many security sensitive systems disable hyper-threading to reduce the attack surface to microarchitectural attacks. However, an adversary can execute the same attack on the shared last-level cache (LLC). If the LLC is inclusive, this is trivial, since any write to an L1 cache entry is automatically written to the LLC. Thus, the hammering in the L1 cache automatically causes hammering of the same memory line in the LLC, leading to bit-flips in both caches. Note that the bit-flips will likely not occur on the same data in both caches since. However, as we found in our experiment above, introducing a random fault in the cache does in most cases not cause reliability issues or crashes.

In non-inclusive LLCs, the opportunities for the attacker are smaller. That is, since the coherency policy prohibits injecting bit-flips in private caches of other cores. Therefore, the attacker needs to find a cache set that holds the data required by the victim process, but that is not present in the victim's core private cache. While this is more challenging than in the inclusive environment, in the RSA example above, the attacker has an almost unlimited amount of tries and a single success leaks the secret key.

Additional attack vectors: We demonstrated that ReRAM memory in caches allows adversaries to inject faults in the data section of the cache in order to leak sensitive information. In addition to the data section, each cache entry contains several other bits of information whose integrity is crucial for the system's security. For example, an adversary can manipulate the tag section of an entry by repeatedly accessing eviction sets larger than the caches' associativity for adjacent cache sets. In doing so, the tag sections will be overwritten with different tags constantly, increasing the risk of NeuroHammer-based bit-flips in target entry. There are two scenarios where an attack on the tag section can be harmful: First, the attacker can cause false cache hits for some addresses, i.e., by faulting the tag, different addresses may cause a cache hit at the cache entry with the faulted tag. Secondly, the attacker can delete write operations by faulting the tag of a dirty entry. This can lead to corrupted data states in a program, and, more crucially, cause memory corruption outside the victims address space. That is, since the faulty entry remains valid and dirty, and thus, the data is written back to memory at some point. However, since the tag is corrupted, the restored address for the write-back will be corrupted and could potentially overwrite any address in memory.

Next to the tag, the attacker can also target the flags section of a cache entry. For example, by repeatedly accessing and invalidating the same entry, the valid flag is implicitly hammered. Since the NeuroHammer attack on NVMs only allows bit-flips from zero to one, the attacker can re-validate entries invalidated before (assuming a high-active valid flag). This can also cause data corruption and incoherent cache states.

5 COUNTERMEASURES

In the following, we discuss countermeasures to mitigate the NeuroHammer attack on main and cache memories.

Error correction codes (ECCs) have been successfully used in various applications to defend against data corruption [33]. Each line of memory (e.g., a crossbar) is extended by some additional bits that are used to store an error correcting code. On each access, the ECC is validated and if a corruption is detected, the data is restored. ECCs can only detect and correct a limited number of faults whereby usually, the number of detectable faults is greater than the number of correctable faults. Generally, the number of correctable and detectable faults is a function of the length of the code. Hence, the hardware and storage overhead depends on the targeted capabilities of the error correction.

Notably, for Rowhammer it has been shown that even ECC protected memory is vulnerable to attacks since the error correction creates a timing side channel that attackers can exploit to leak information [7]. However, the severity of this attack is arguably less since no data is corrupted. Hence, the timing attack on error correction only allows data leakage but not corrupting attacks like privilege escalation. In the cache setting, error correction can be omitted in favor of error detection in most cases. That is, since when an error is detected, the cache line can simply be invalidated and the data will be loaded from memory on the next access. Only if the cache entry was flagged as dirty, the data must be corrected or a fault must be raised on access.

With NeuroHammer, it is much easier for attackers to inject targeted single-bit errors than it is with Rowhammer. Therefore, the application of ECC as a countermeasure is challenging. Consider the following example, where the Hamming weight of the data is stored as ECC alongside the memory line:

	Data								Checksum		
Victim	0	0	1	1	0	1	1	0	1	0	0
Attacker 🔨	0	0	0	0	0/1	0	0	0	0	0	0/1

Now, if the attacker writes a single 0 and 1 in alternating order, they not only hammer the respective bit in the victim's data, but also the checksum. In this simple example, the checksum remains valid:

	Data								Checksum		
Victim	0	0	1	1	**1**	1	1	0	1	0	**1**
Attacker 🔨	0	0	0	0	0/1	0	0	0	0	0	0/1

A trivial way to prevent this would be to store the inverted data as a error correction. Since bit-flips occur only from low to high, the attacker cannot construct a payload that modifies the data while keeping the error correction data intact. However, this requires the length of the checksum to be equal to the data, and is therefore very expensive. While more complex functions increase the complexity of constructing valid payloads for the hammering, it is intuitively hard to construct an ECC resistant against checksum manipulation. One possible solution would be to store the ECCs independently of the actual ReRAM, and thus shield them from NeuroHammer attacks. However, this significantly increases the complexity of the memory/cache.

A further downside of using ECC, especially in the cache use-case, it the increased complexity. Data can only be returned when the ECC check (and possibly the correction) is successful and thus, the access latency of the cache is negatively impacted.

Cache randomization [41, 52, 60] is a promising approach to significantly reduce the threat of contention-based cache attacks, like Prime+Probe [60]. By randomizing the mapping of addresses to cache sets, attackers can no longer efficiently construct eviction sets. The randomization also increases the complexity of NeuroHammer attacks in the cache, as the adversary can no longer construct

addresses that map to adjacent cache sets. Therefore, the attacker can only introduce faults at random positions, which reduces the probability of success. Considering attack scenarios in which a single bit-flip is sufficient for the attack, probabilistic protection by cache randomization is not sufficient.

Likewise, randomization techniques may also be applied to main memories. By randomizing the mapping of addresses to crossbars, attackers can no longer conveniently find adjacent memory lines. Especially considering the size of main memory, this approach is promising, since the randomization space is large enough for the attacker to only have a very small success probability. At the same time, the overhead of a randomized mapping is small. Overall, randomizing the mapping can increase the complexity of both cache and main memory NeuroHammer attacks. However, randomization does not solve the underlying problem, and the attacker always has a small chance of succeeding.

Cell Refreshment: Since DRAM is a volatile memory technology, the cells need to be refreshed frequently. Traditional Rowhammer attacks can be prevented by increasing the cell refresh rate, preventing sufficient charge leakage to flip the bit [23]. However, ReRAMs are non-volatile and store the information in the form of resistance, which is modulated by the redistribution of ionic defects [59]. Therefore, usually the cells do not need to be refreshed with ReRAM, and implementing a generic refresh mechanism would introduce significant overhead in energy consumption. More suited would be a mechanism that monitors cell accesses and identifies hammered cells during runtime, as discussed in [23]. Considering the endurance problem of ReRAMs [65], an intelligent cell refreshment would be applicable because it detects the likelihood of a NeuroHammer attack on a particular cell with a certain probability. The memory controller could then temporarily block access to this memory region or map it to another memory location.

Detection: Another possibility to protect against ReRAM fault attacks is using a detection mechanism to identify attack access patterns before bit-flips can occur. The detection of attacks to main memory is similar to detecting cache attacks. That is, since the hammered data must be evicted from the cache repeatedly and therefore, many cache misses occur. A variety of proposals [4, 9, 66] have been made how to detect cache attacks using hardware performance counters (HPCs). These counters are meant for performance optimization and can be configured to count CPU internal events like cache misses and flushes. When a attack pattern is detected by such a mechanism, the memory controller could prevent either remap the memory to a different crossbar, or prevent the attacker from hammering the address by prohibiting cache flushes, e.g., by locking the cache entry to the cache.

The cache use case is more difficult to detect using protection mechanisms. That is, since it does not inherently require unusual access patterns that raise events like cache misses in the CPU. Hence, HPC-based detection is not feasible. Each cache entry could be extended by a write counter that would identify memory lines that are hammered. If such a threshold detection would trigger, it would need to stop the attacking program which can be a threat to safety assurances the target device might need to fulfill. While it would also be possible to switch to a write-through policy on a detected attack, this would then expose the main memory to attacks.

6 CONCLUSION

In this paper, we investigated the system-level impact of NeuroHammer attacks on NVMs. We implemented NVgem5, a gem5-compatible NVM extension, which allows the injection of deliberate bit-flips on cache and main memory level. Our experiment demonstrated an attack that recovers the secret key during an RSA signature generation by introducing a single bit-flip in the cache. With the presented work, we show that the implications of the NeuroHammer attack go far beyond isolated bit-flips in NVMs, allowing attackers to compromise full-fledged systems. In the future, we want to verify the attack on real hardware to verify our simulation results.

ACKNOWLEDGMENTS

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, under the Priority Program SPP 2253 Nano Security (Project RAINCOAT - Number: 440059533) and by the Federal Ministry of Education and Research (BMBF, Germany) in the project NEUROTEC II (16ME0398K, 16ME0399).

REFERENCES

[1] An Chen. A Review of Emerging Non-Volatile Memory (NVM) Technologies and Applications. *Solid-State Electronics* (2016).
[2] Binkert, N., Beckmann, B., Black, G., et al. The Gem5 Simulator. *SIGARCH* (2011).
[3] Boneh, D., DeMillo, R. A., and Lipton, R. J. On the Importance of Eliminating Errors in Cryptographic Computations. *J. Cryptol.* (2001).
[4] Chiappetta, M., Savas, E., and Yilmaz, C. Real Time Detection of Cache-based Side-Channel Attacks Using Hardware Performance Counters. *Applied Soft Computing 49* (2016), 1162–1174.
[5] Cojocar, L., Kim, J., Patel, M., et al. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *IEEE Symposium on Security and Privacy (SP)* (May 2020), IEEE.
[6] Cojocar, L., Kim, J., Patel, M., et al. Are we susceptible to rowhammer? an end-to-end methodology for cloud providers. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 712–728.
[7] Cojocar, L., Razavi, K., Giuffrida, C., et al. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, IEEE, pp. 55–71.
[8] De, A., Khan, M. N. I., Park, J., et al. Replacing eFlash with STTRAM in IoTs: Security Challenges and Solutions. *Journal of Hardware and Systems Security* (2017).
[9] Demme, J., Maycock, M., Schmitz, J., et al. On The Feasibility of Online Malware Detection With Performance Counters. In *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013* (2013), A. Mendelson, Ed., ACM, pp. 559–570.
[10] DeSalvo, B., Vianello, E., Thomas, O., et al. Emerging resistive memories for low power embedded applications and neuromorphic systems. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (2015), IEEE, pp. 3088–3091.
[11] Di Dio, A., Koning, K., Bos, H., et al. Copy-on-flip: Hardening ecc memory against rowhammer attacks. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium* (2023).
[12] Dodo, S. B., Bishnoi, R., and Tahoori, M. B. Secure STT-MRAM Bit-Cell Design Resilient to Differential Power Analysis Attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2020).
[13] Frigo, P., Vannacc, E., Hassan, H., et al. Trrespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 747–762.
[14] Gallo, M. L., and Sebastian, A. An Overview of Phase-Change Memory Device Physics. *J. Phys. D Appl. Phys. 53*, 21 (2020), 213002.
[15] Gruss, D., Maurice, C., Wagner, K., et al. Flush+Flush: A Fast and Stealthy Cache Attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings* (2016), J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds., vol. 9721 of *Lecture Notes in Computer Science*, Springer, pp. 279–299.
[16] Guide, P. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part 2*, 11 (2011).

[17] Intel. Intel® Optane™ Memory Series, 2022.

[18] Jiang, Y., Zhu, H., Shan, H., et al. Trrscope: Understanding target row refresh mechanism for modern ddr protection. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2021), IEEE, pp. 239–247.

[19] Khan, M. N. I., Bhasin, S., Yuan, A., et al. Side-Channel Attack on STTRAM Based Cache for Cryptographic Application. In *2017 IEEE International Conference on Computer Design (ICCD)* (2017), IEEE.

[20] Khan, M. N. I., and Ghosh, S. Analysis of Row Hammer Attack on STTRAM. In *IEEE 36th International Conference on Computer Design (ICCD)* (Oct. 2018), IEEE.

[21] Khan, M. N. I., and Ghosh, S. Information Leakage Attacks on Emerging Non-Volatile Memory and Countermeasures. In *Proceedings of the International Symposium on Low Power Electronics and Design* (2018), ACM.

[22] Kim, Y., Daly, R., Kim, J., et al. 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)* (June 2014), IEEE.

[23] Kim, Y., Daly, R., Kim, J. S., et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014* (2014), IEEE Computer Society, pp. 361–372.

[24] Krautter, J., Mayahinia, M., Gnad, D. R., et al. Data Leakage through Self-Terminated Write Schemes in Memristive Caches. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)* (2022), IEEE.

[25] Lee, M.-J., Lee, C. B., Lee, D., et al. A Fast, High-Endurance and Scalable Non-Volatile Memory Device Made from Asymmetric Ta2O5-x/TaO2-x Bilayer Structures. *Nature Materials* (2011).

[26] Leidel, J. D., and Chen, Y. HMC-Sim-2.0: A Simulation Platform for Exploring Custom Memory Cube Operations. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (May 2016), IEEE.

[27] Li, H. H., Chen, Y., Liu, C., et al. Looking ahead for resistive memory technology: A broad perspective on ReRAM technology for future storage and computing. *IEEE Consumer Electronics Magazine 6*, 1 (Jan. 2017), 94–103.

[28] Lipp, M., Schwarz, M., Raab, L., et al. Nethammer: Inducing rowhammer faults through network requests. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2020), IEEE, pp. 710–719.

[29] Liu, F., Yarom, Y., Ge, Q., et al. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy* (May 2015), IEEE.

[30] Liu, F., Zhao, W., Zhao, Y., et al. SME: ReRAM-based Sparse-Multiplication-Engine to Squeeze-Out Bit Sparsity of Neural Network, 2021.

[31] Mao, M., Cao, Y., Yu, S., et al. Programming Strategies to Improve Energy Efficiency and Reliability of ReRAM Memory Systems. In *IEEE Workshop on Signal Processing Systems (SiPS)* (Oct. 2015), IEEE.

[32] Mark Seaborn, T. D. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges, 2015.

[33] Morelos-Zaragoza, R. H. *The Art of Error Correcting Coding*. John Wiley & Sons, 2006.

[34] Mutlu, O., and Kim, J. S. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39*, 8 (2019), 1555–1571.

[35] Osvik, D. A., Shamir, A., and Tromer, E. Cache Attacks and Countermeasures: The Case of AES. In *Topics in Cryptology – CT-RSA 2006*. Springer Berlin Heidelberg, Berlin, Heidelberg, Jan. 2006, pp. 1–20.

[36] Parkin, S., Jiang, X., Kaiser, C., et al. Magnetically Engineered Spintronic Sensors and Memory. *Proc. IEEE 91*, 5 (2003), 661–680.

[37] Poremba, M., and Xie, Y. NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories. In *IEEE Computer Society Annual Symposium on VLSI* (Aug. 2012), IEEE.

[38] Rai, S., Garg, S., Pilato, C., et al. Vertical IP Protection of the Next-Generation Devices: Quo Vadis? In *Design, Automation Test in Europe Conference Exhibition (DATE)* (2021), pp. 1905–1914.

[39] Razavi, K., Gras, B., Bosman, E., et al. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 1–18.

[40] Rosenfeld, P., Cooper-Balis, E., and Jacob, B. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters 10*, 1 (Jan. 2011), 16–19.

[41] Saileshwar, G., and Qureshi, M. K. MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021* (2021), M. Bailey and R. Greenstadt, Eds., USENIX Association, pp. 1379–1396.

[42] Seaborn, M., and Dullien, T. Exploiting the DRAM Rowhammer Bug to gain Kernel Privileges. *Black Hat* (2015).

[43] Serino, A., and Cheng, L. Real-time operating systems for cyber-physical systems: Current status and future research. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)* (2020), IEEE, pp. 419–425.

[44] Sisejkovic, D., and Leupers, R. Trustworthy Hardware Design with Logic Locking. In *IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)* (2021), pp. 1–2.

[45] Smith, A. J. Cache Memories. *ACM Computing Surveys 14*, 3 (Sept. 1982), 473–530.

[46] Song, W., and Liu, P. Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID* (2019).

[47] Staudigl, F., Al Indari, H., Schön, D., et al. NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2022), IEEE.

[48] Staudigl, F., Merchant, F., and Leupers, R. A Survey of Neuromorphic Computing-in-Memory: Architectures, Simulators, and Security. *IEEE Design & Test* (2022).

[49] Tanenbaum, A. S., and Goodman, J. *Computerarchitektur: Strukturen, Konzepte, Grundlagen*. Pearson Studium München et al., 2006.

[50] Tatar, A., Konoth, R. K., Athanasopoulos, E., et al. Throwhammer: Rowhammer attacks over the network and defenses. In *2018 {USENIX} {ATC} 18)* (2018), pp. 213–226.

[51] Thoma, J. P., and Güneysu, T. Write Me and I'll Tell You Secrets - Write-After-Write Effects On Intel CPUs. In *25th International Symposium on Research in Attacks, Intrusions and Defenses, RAID* (2022), ACM.

[52] Thoma, J. P., Niesler, C., Funke, D. A., et al. ClepsydraCache – Preventing Cache Attacks with Time-Based Evictions. In *32nd USENIX Security Symposium (USENIX Security 23)* (Anaheim, CA, Aug. 2023), USENIX Association.

[53] Thoziyoor, S., Muralimanohar, N., Ahn, J. H., et al. CACTI 5.1. Tech. rep., Technical Report HPL-2008-20, HP Labs, 2008.

[54] Tromer, E., Osvik, D. A., and Shamir, A. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology 23*, 1 (2010), 37–71.

[55] Van Schaik, S., Milburn, A., Osterlund, S., et al. Ridl: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP 2019)* (United States, May 2019), Proceedings - IEEE Symposium on Security and Privacy, Institute of Electrical and Electronics Engineers Inc., pp. 88–105. 40th IEEE Symposium on Security and Privacy, SP 2019 ; Conference date: 19-05-2019 Through 23-05-2019.

[56] Vieira, J., Roma, N., Falcao, G., et al. gem5-ndp: Near-Data Processing Architecture Simulation From Low Level Caches to DRAM. In *IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (Nov. 2022), IEEE.

[57] Vila, P., Köpf, B., and Morales, J. F. Theory and practice of finding eviction sets. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 39–54.

[58] von Witzleben, M., Fleck, K., Funck, C., et al. Investigation of the Impact of High Temperatures on the Switching Kinetics of Redox-based Resistive Switching Cells using a Highspeed Nanoheater. *Adv. Electron. Mat.* (2017).

[59] Waser, R., and Aono, M. Nanoionics-based Resistive Switching Memories. *Nature Materials* (2007).

[60] Werner, M., Unterluggauer, T., Giner, L., et al. ScatterCache: Thwarting Cache Attacks via Cache Set Randomization. In *USENIX* (2019), N. Heninger and P. Traynor, Eds.

[61] Xu, S., Chen, X., Wang, Y., et al. PIMSim: A Flexible and Detailed Processing-in-Memory Simulator. *IEEE Computer Architecture Letters 18*, 1 (Jan. 2019), 6–9.

[62] Yan, M., Sprabery, R., Gopireddy, B., et al. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), pp. 888–904.

[63] Yang, C., Liu, B., Li, H., et al. Thwarting Replication Attack Against Memristor-Based Neuromorphic Computing System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[64] Yarom, Y., and Falkner, K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014* (2014), K. Fu and J. Jung, Eds., USENIX Association, pp. 719–732.

[65] Zahoor, F., Zulkifli, T. Z. A., and Khanday, F. A. Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (MLC) Storage, Modeling, and Applications. *Nanoscale Research Letters* (2020).

[66] Zhang, T., Zhang, Y., and Lee, R. B. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. In *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings* (2016), F. Monrose, M. Dacier, G. Blanc, and J. García-Alfaro, Eds., vol. 9854 of *Lecture Notes in Computer Science*, Springer, pp. 118–140.