ISEC

SCIENCE · PASSION · TECHNOLOGY

TU Graz

David Kröll

Simulating Rowhammer Attacks on Encrypted Memory

BACHELOR'S THESIS

Bachelor's degree programme: Software Engineering and Management

Supervisor

Lukas Lamster

Institute of Information Security
Graz University of Technology

Graz, July 2026

Abstract

Rowhammer attacks remain a serious threat to modern computing systems, especially in cloud environments where victims may share hardware with attackers. While memory encryption protects the confidentiality of data in memory, it is unclear whether encryption without integrity protection can effectively defend against Rowhammer-induced memory corruption.

This work presents a simulation-based analysis of Rowhammer attacks against encrypted but unauthenticated memory. The evaluation focuses on x86 systems and analyzes how Rowhammer-induced bit flips propagate under different encryption modes. Evaluated encryption modes in this work are Cipher Block Chaining (CBC), Counter (CTR), and Xor-Encrypt-Xor (XEX). The results show that Rowhammer-induced bit flips can still produce valid corrupted page table entries (PTEs) with significant probability. CBC mode yields success probabilities between 20% and 65%, CTR mode between 64% and 99%, and XEX mode between 0.1% and 50%. Our proposed simulator generates random PTEs, encrypts them, injects bit flips, and evaluates whether the resulting corrupted PTEs remain valid after decryption.

Overall, the findings demonstrate that pure memory encryption alone is not a sufficient defense against Rowhammer attacks on page tables. Additional integrity protection or dedicated Rowhammer mitigation mechanisms remain necessary to provide reliable security.

Keywords: System security · Memory security · Rowhammer · Cryptography · Unauthenticated encryption · Virtual memory paging

Contents

1 Introduction

Computing systems demand robust security, particularly in cloud environments where multiple tenants share the same hardware but remain mutually distrusted. Moreover, cloud systems often run on hardware owned and operated by third parties, where protection against hardware attacks is necessary.

Modern processors use memory encryption to protect data in DRAM from attackers with physical or low-level system access. These mechanisms aim to preserve confidentiality even when an attacker can observe or manipulate memory contents directly. A prominent example is Intel Total Memory Encryption (TME) [Cor25].

Memory encryption primarily protects confidentiality, but it does not provide integrity or authenticity guarantees by itself. As a result, bit flips caused by hardware faults or attacks can still propagate into decrypted data and modify its meaning. This makes encrypted memory potentially vulnerable to fault attacks such as Rowhammer.

Memory protection mechanisms therefore play a crucial role in defending against such attackers. Various Rowhammer-style attacks [GMM16; LSR+20; GLS18] have already been demonstrated in practice. Additionally, research proposes many Rowhammer-specific defenses [BDG+17; JLK+23; Cor16]. Furthermore, general integrity protection systems have been proposed as well, with Intel's Software Guard Extensions (SGX) as a prominent example [Gue16].

Rowhammer attacks often target page table entries (PTEs), since only small modifications in these structures can lead to privilege escalation or arbitrary memory access, as shown in [SD15]. However, most existing work focuses on unencrypted memory or systems with strong integrity protection. To the best of our knowledge, no prior work has evaluated Rowhammer attacks on encrypted memory.

This work introduces a simulation and analysis framework to study whether pure memory encryption without integrity protection can defend against Rowhammer attacks. We show that even with memory encryption in place, systems remain vulnerable to Rowhammer attacks.

Outline. In Chapter 2, we introduce the relevant background for Rowhammer attacks and defenses. We continue with general cryptographic building blocks also used for memory encryption and virtual memory paging. Chapter 3 introduces the simulator and the capabilities of the attacker. In Chapter 4 the results of the simulator are outlined and discussed. It includes analysis of how memory modifications from Rowhammer attacks influence faults induced in encrypted memory. An overview of future work and related targets is provided in Chapter 5. Finally, in Chapter 6 we summarize with a conclusion.

2 Background and Related Work

In this chapter, a brief explanation of Rowhammer and the underlying physical microarchitectural attack vector is provided. Furthermore, a selection of prominent attacks and defenses is outlined. In section 2.3 cryptographic building blocks required for encryption are outlined. Section 2.4 concludes with an overview of virtual memory paging.

2.1 Rowhammer

Memory isolation is fundamental for reliable and secure computing systems, ensuring that access to one memory address does not unintentionally affect data stored at other addresses. Modern commodity DRAM chips are vulnerable to so-called *disturbance errors*, most notably the Rowhammer phenomenon [KDK+14].

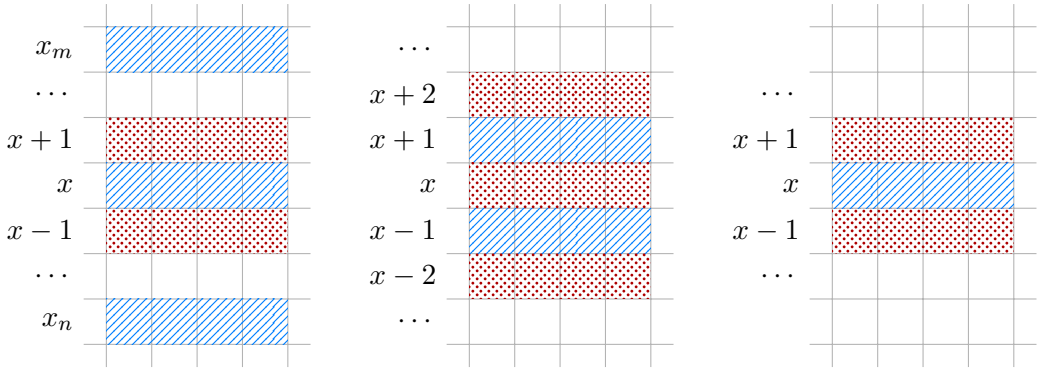
Rowhammer exploits the physical properties of DRAM cells: by repeatedly activating (or hammering) the same memory row, an attacker can induce electrical interference in adjacent rows. This repeated activation accelerates the electrical discharge of nearby cells. However, discharging of DRAM cells is a known issue. Therefore, a so-called *refresh* is implemented, which restores the current value in the cell. When the victim cell discharges faster than it is refreshed, the value is lost, ultimately causing bit flips in memory locations that are never accessed. As Kim et al. already showed, the likelihood of disturbance errors increases when the physical adjacency of aggressor and victim rows increases [KDK+14].

Patent filings show that the industry has known about these disturbance error problems since at least 2012 [BHSG15]. Rowhammer becomes worse as DRAM technology continues to shrink and memory density increases, making newer generations of DRAM more vulnerable to this problem [KDK+14].

Row Activation Threshold. The Rowhammer threshold is the minimum number of aggressor row activations required to induce bit flips in victim rows. Studies show significant differences in the value of the threshold between DDR3 and DDR4 DRAM generations. Early experiments on DDR3 reported relatively high thresholds, typically around 139k [SD15] activations for single-sided hammering, with observed ranges between 100k and 200k depending on the module and testing conditions. In contrast, DDR4 shows lower thresholds, with bit flips often occurring after tens of thousands of activations. Under certain hammering strategies, DDR4 bit flips have been observed at activation counts as low as approximately 4k to 50k. Although mitigation mechanisms such as *Target Row Refresh (TRR)* (see Section 2.2) are commonly used in DDR4, research has demonstrated that it is still possible to induce bit flips. Overall, the thresholds are

decreased with the shift to newer and denser DRAM technology [KDK+14; JVF+22; SD15].

Hammering Strategies. Several strategies have been proposed to reliably induce bit flips through specific memory access patterns, as shown in Figure 2.1. Some of them use so-called *row conflicts*, which are based on a caching technique introduced in DRAM. Accessing memory always makes use of a row buffer, but when the requested row is not yet in the buffer, a row conflict occurs [GLS18].



(a) Single-sided (b) Double-sided (c) One-location

Figure 2.1: Rowhammer strategies: in blue (lines) the hammered location, in red (dots) are the locations most likely to be affected by bit flips. Reproduced from [GLS18].

Single-sided hammering does not necessarily involve accessing only a single memory location. Seaborn and Dullien accessed eight randomly selected memory locations simultaneously on a typical DDR3 system with 32 DRAM banks. Based on the principle of the birthday paradox, there is a high probability that at least two of these addresses map to the same DRAM bank. Repeatedly accessing these locations generates frequent row conflicts, which most likely leads to bit flips close to one of the 8 hammered rows.

In *double-sided hammering*, the attacker targets three adjacent rows and repeatedly accesses the two outer rows. This pattern increases the likelihood that bit flips occur in the middle row. However, this technique requires at least some knowledge of virtual-to-physical memory mapping.

In *one-location hammering*, only a single memory location is accessed at a very high frequency. Rather than directly causing row conflicts, this approach repeatedly reopens the same row. In addition, the memory controller may close rows earlier than necessary. Therefore, subsequent accesses reopen the row, indirectly creating row conflicts that can induce bit flips near the repeatedly accessed row [GLS18].

Attacks using Rowhammer. Various attacks applying the Rowhammer phenomenon have been shown to compromise system security. Most notably, Seaborn and Dullien showed, that by only using unprivileged applications, it is already possible to gain read-write access to the full physical memory. This is possible by introducing bit flips in page table entries (PTEs) effectively resulting in kernel privileges [SD15].

Rowhammer.js is a JavaScript-based Rowhammer attack that induces DRAM bit flips from within a web browser. Cache eviction in combination with regular memory accesses in carefully crafted patterns is exploited here, triggering repeated row activations. This demonstrates that Rowhammer attacks can be mounted through websites and even inside a sandboxed environment like JavaScript [GMM16].

In addition, an attack without any code running on the target machine has been shown. *NetHammer* is a remote Rowhammer attack that induces DRAM bit flips by using just network traffic without local code execution. Sending network packets at a high frequency to the target system causes memory accesses high enough to cause bit flips in memory. This attack demonstrates the vulnerability over the network, expanding the attack surface significantly [LSR+20].

Even attacks across virtual machine boundaries have shown that memory isolation provided by virtualization can be broken by Rowhammer. Xiao et al. showed that it is practicable to gain access to the full physical memory from within a guest virtual machine [XZZT16].

2.2 Rowhammer Defenses

Various efforts were undertaken to prevent Rowhammer attacks, using different approaches. A selection of these is briefly outlined here. [HF15; IES16] introduce static analysis of binary code, trying to detect the characteristics of code that is exploiting microarchitectural attack vectors.

Corbet introduced a method to detect and mitigate Rowhammer attacks in software through analysis of cache misses using performance-monitoring units inside processors [Cor16].

Even software-only defenses isolating kernel and user memory parts in physical memory are presented as an effective defense against the Rowhammer attack [BDG+17].

A memory error detection system, namely *CSI:Rowhammer* uses cryptographic message authentication codes, and a software-based memory correction system prevents not only Rowhammer but, in general, memory corruption errors. Notable in this work is that it can prevent memory corruption even if multiple bit values are corrupted [JLK+23].

Target Row Refresh is a technique inside DRAM chips to refresh victim rows before bits are flipped there. TRR identifies aggressor rows and refreshes the neighboring victims rows. This is only feasible when the aggressor rows are easy to identify and are following typical access patterns. To work around this attempt to mitigate the Rowhammer bug, different and less predictable access patterns can be used, where it is still vulnerable [JVF+22].

Attacks and defenses have been investigated to both compromise and improve system security. Mutlu and Kim provide a more comprehensive review of attacks and defenses in [MK19].

At the time of writing, no Rowhammer attacks on encrypted memory have been evaluated.

2.3 Cryptographic Building Blocks

This section gives an introduction to cryptographic block ciphers. In particular, we elaborate on the three encryption modes *Cipher Block Chaining (CBC)*, *Counter Mode (CTR)* and *Xor-Encrypt-Xor (XEX)*, which are explained below.

2.3.1 Encryption

Transforming data into a format that is not readable by any attacker is called encryption and provides the security property *confidentiality*. Adding a tag to the data makes sure that it is not altered by an attacker, providing the integrity of the data. Tags are computed using a *Message Authentication Code (MAC)* in the form of:

$$T = MAC_K(D)$$

Where T is the tag, MAC_K the function and D the data. MAC functions achieve authenticity by computing the MAC function again and comparing it with the stored tag. This ensures neither the data nor the tag is altered [Aum17].

Block Ciphers. A block cipher is an encryption scheme which works on blocks of a fixed size of data by providing a function for both encryption and decryption

$$C = E_K(P),$$

$$P = D_K(C),$$

where C is the ciphertext, E_K the encryption function (with key K as additional input), P the plaintext and D_K the decryption function. The decryption function is the inverse of the encryption function. For an attacker, it should not be possible to compute either $E_K(P)$ or $D_K(C)$ if the key K is unknown. The output of a sophisticated encryption function should be indistinguishable from a *pseudorandom permutation* [Aum17].

Mode of Operation. Block ciphers in the most primitive form can only operate on data which is exactly one block at a time. To break this limitation, a more sophisticated combination of cryptographic functions is required. These are called *modes of operation*. This work covers *Cipher Block Chaining (CBC)*, *Counter Mode (CTR)* and *Xor-Encrypt-Xor (XEX)*.

2.3.2 Authenticated Encryption

As encryption only provides confidentiality, message tampering cannot be detected, therefore authenticated encryption must be used. To combine the security properties integrity and confidentiality, both an encryption function and a Message Authentication Code (MAC) can be combined into a uniform construction. Encryption schemes with integrity verification allow detection of message tampering by generating not only the ciphertext but an additional authentication tag, used for integrity verification. This leads to an obvious overhead in storing the authentication tag alongside the real data, reducing the effective usable memory. Various other contributions [Gue16; YEP+06; TSB18; SNR+18; GSC+03] use this approach to effectively provide these security conditions over large amounts of memory. The overhead of full confidentiality and integrity caused developers to create further optimizations.

2.3.3 Cipher Modes

Cipher modes, or modes of operation, define how cryptographic primitives process data larger than a single block by combining plaintext blocks with previous blocks and possible initialization values to produce secure ciphertexts. Different constructed cipher modes offer different properties in security, performance, parallelizability or ease of hardware implementation.

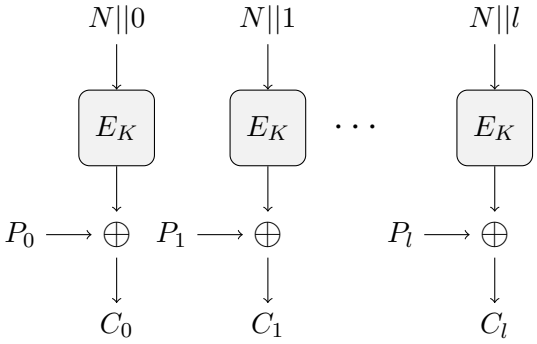


Figure 2.2: Counter Mode (CTR) encryption scheme constructions where E_K is the encryption function, N is a nonce, P_l is the plaintext and C_l is the ciphertext; reproduced from [Aum17]

Counter Mode. This mode splits up the input plaintext, however a nonce N and a block counter is used as input to the encryption function. After encrypting, a *pad* is generated, where the plaintext P is xor'ed, generating the ciphertext C (c.f. Figure 2.2). Note that a decryption function D_K is the same as the encryption function, since xor-ing the ciphertext with the pad yields the original plaintext again. Given another counter or nonce, the counter mode prevents ciphertext repeats. As each ciphertext block is

computed purely based on the nonce, block counter, key and the plaintext block, a modification in one block does not influence any other block [Aum17].

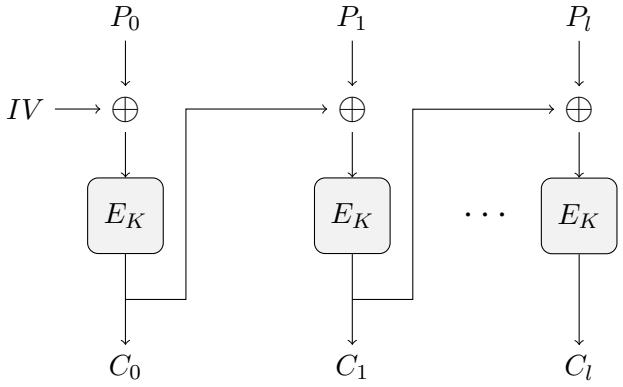


Figure 2.3: Cipher Block Chaining (CBC) encryption scheme where E_K is the encryption function, IV is the initialization vector, P_l is the plaintext and C_l is the ciphertext; reproduced from [Aum17]

Cipher Block Chaining. This mode also processes plaintext in fixed-size blocks, but instead of using a nonce and counter as in Counter Mode, each plaintext block is xor'ed with the previous ciphertext block before being encrypted. Figure 2.3 shows the CBC encryption scheme. The first block uses an initialization vector (IV) as a replacement for the previous block. Each ciphertext block C_l is computed by encrypting the XOR of the plaintext block P_l and the previous ciphertext block C_{l-1}. Unlike in counter mode, the encryption function is not identical to the decryption function, instead this process must be reversed. The combination of an initialization vector and the XOR with the previous block, a ciphertext repeat does not occur given unique IVs are used. As seen, a modification of the plaintext in one block will also modify data in the following block [Aum17].

Xor-Encrypt-Xor (XEX). XEX is a mode that takes a classic block cipher and turns it into a tweakable block cipher using an additional key. The second key is used to derive a per-tweak offset. For a given tweak, an initial offset is computed by encrypting the tweak under the second key and then updated for successive blocks. Each plaintext block is encrypted using the derived offset. The tweak property of XEX allows identical plaintext blocks stored in different locations to encrypt to different ciphertext. As shown in Figure 2.4 the construction uses two keys with added XOR operations [Rog04].

Advanced Encryption Standard (AES). AES is a symmetric block cipher standardized by the National Institute of Standards and Technology (NIST). It operates on fixed 128-bit data blocks and provides strong security against known cryptographic attacks. AES is widely used for securing sensitive data and is generally believed to be secure.

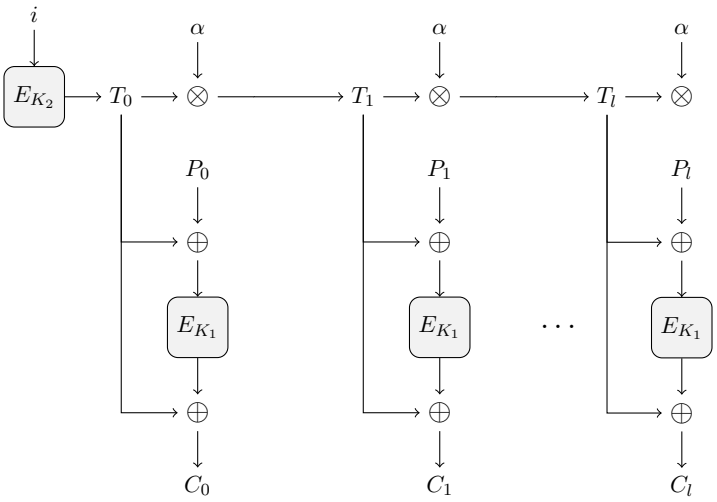


Figure 2.4: XEX encryption mode with two keys. The tweak T_0 is generated using K_2 and evolved per block via multiplication by α. Each block is encrypted using Xor–Encrypt–Xor with key K_1; reproduced from [Wik26]

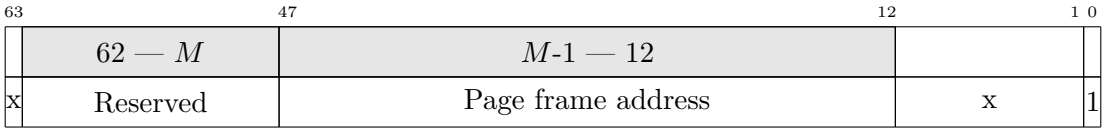
Due to its efficiency in both hardware and software implementations, AES has become a global standard for protecting data. The AES primitives as encryption functions can be used to build the cipher modes described above [DR02].

2.4 Virtual Memory Paging

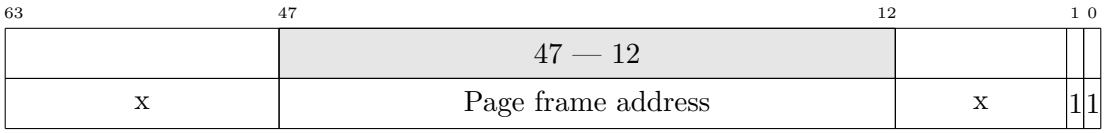
Virtual memory paging is a memory management technique that translates linear (virtual) addresses into physical memory addresses. In Intel x86 architectures, this translation is performed using paging structures defined by the processor specification and configured by the operating system. The operating system initializes and maintains these paging structures, which consist of hierarchical page tables. The virtual address acts as an index in these tables to finally resolve to a physical page frame [Cor26].

On multi-process operating systems, every process operates within its own virtual address space, effectively isolating it from other processes. Through this abstraction, memory management is easier and allows efficient use of physical memory. Furthermore, paging supports fine-grained access control via permissions associated with each page. These mechanisms enforce memory protection and ensure process isolation, which are fundamental for security [Cor26].

The paging structure itself is only accessible to the operating system, as modifying the page tables allows accessing arbitrary physical memory. Seaborn and Dullien already showed that this is exploitable to gain kernel privileges using a Rowhammer approach on page table entries. Once writable access to a paging structure is obtained, full control over the whole physical memory is possible [SD15].



(a) x86 PTE leaf node



(b) ARM PTE leaf node

Figure 2.5: Due to readability constraints, irrelevant fields have been omitted and marked with the letter x.
(a) showing an x86 page table entry, where M represents the value of MAXPHYADDR (maximum physical address bits), which depends on the CPU. It shows a MAXPHYADDR of 47, the maximum value for MAXPHYADDR is 52. Reproduced from [Cor26].
(b) showing a leaf node, on ARM it is called L3 page descriptor (64-bit, 4KB granule) [Lim19; Ban26].

Figure 2.5 shows what a page table entry looks like in memory. Part *a)* prints the x86 page table entry format. In the figure, only the bottom-level page table entry is denoted. The upper levels are designed roughly the same and can be looked up at [Cor26]. The most interesting field is the page frame address, which in the end resolves to the physical memory address. The reserved bits must stay zero for the PTE to be valid. Bit 0, called *present bit* must always be 1 to enable access to the data page. When the *present bit* is zero, the memory management unit will not read the data and instead raise a fault while accessing it. As a result, when this bit ends up zero, the PTE gets unusable [Cor26].

Part *b)* illustrates the ARM page table entry format. The ARM architecture uses the terminology page descriptor, nevertheless it has the same meaning as PTE for x86. Bit 0 (same as in x86) is the *valid bit*, which must have the value of 1 to be valid. Bit 1, called *type bit* specifies the type of the following level (can either be *table* or *block*). In leaf nodes there is no following level, hence this bit must also be set to 1 to form a valid entry. Full specification and definition for other levels can be viewed at [Lim19].

3 Implementing a Rowhammer Attack Simulator

This chapter elaborates the attacker model, including assumptions on the capabilities of the attacker and background information concerning the target. It also outlines the proposed simulator design in detail.

3.1 Attacker Model

We consider an attacker capable of performing a Rowhammer-style attack with a shared but encrypted memory system. The attacker runs on the same physical hardware as the victim, either in a separate user-space process or inside a virtual machine.

The system under consideration employs transparent main memory encryption. However, the encryption scheme provides confidentiality only and does not implement integrity protection. Missing authentication of the corrupted ciphertexts is a crucial property, otherwise a detection of the attack would be obvious. In general, we assume that inducing bit flips outside the accessed memory is feasible. Therefore, the described simulator is explicitly not running a real Rowhammer attack.

However, a fictional attacker would be capable of executing code with regular user privileges on the host system. Physical access to the machine is not required. The attacker can allocate memory and execute carefully chosen memory access patterns to trigger disturbance errors.

We focus the attack on page table entries (PTEs). If it is feasible to attack encrypted page table entries, it is going to be possible to manipulate kernel memory and gain full physical memory access, as it has already been shown [SD15].

3.2 Simulator Design

Typically, full pages of 4 kB of PTEs are stored in memory [Cor26], whereas every PTE is 8 bytes, resulting in a page filled with 512 entries. In contrast to that, our used encryption granularity of 16 B (128 bit, based on the size of the encryption block size) is different from that. In fact, a bit flip in a block can affect more than one PTE, since each encryption block contains two PTEs. This depends on the mode of operation (see section 2.3.1 and section 4.4), but the affected blocks remain deterministic.

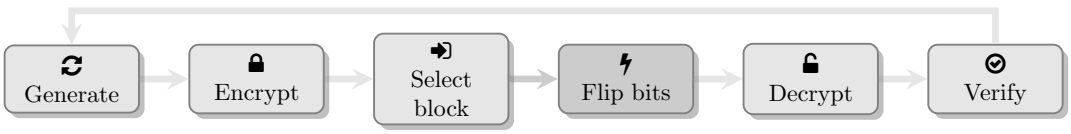


Figure 3.1: The Rowhammer Attack simulator round.

As an overview, one simulation round consists of the steps seen in Figure 3.1.

The simulator tests a total of 10 million induced bit flips in encrypted memory. Randomness is obtained from a ChaCha pseudo-random number generator to also have predictable and seedable randomness [Ber+08]. After the random generation, the PTEs are modified in order that they are valid, in respect of the current simulated hardware properties (e.g. MAXPHYADDR). In practice, this means the present bit and reserved bits are set accordingly (see fig. 2.5 for reference).

The same PTEs are then encrypted with one of the three supported encryption modes: CBC, CTR, or XEX. After 32 runs on the same PTEs, the simulator generates a new table of PTEs. In addition, it regularly generates a fresh random encryption key.

The simulator chooses the bits that should be flipped randomly. A range of 16 bytes, which is the encryption granularity, gets chosen. This range is always aligned with the encryption block, so there are never two bit flips outside a block. However, the bit flips could be in different PTEs because an encryption block contains two PTEs. Within this range a defined number of bits is chosen and flipped. Chosen bits are unique within this set, meaning the same bit is not flipped twice, which would result in the same value as it originally had. We verified that the bit flips are evenly distributed over the data, even outside the 16 byte range.

After introducing the bit flips into the ciphertext, the system decrypts the ciphertext and verifies the affected modified PTEs. Unmodified PTEs are not considered.

4 Evaluation

This chapter evaluates the results generated by the simulator. Results of all three cipher modes, CBC, CTR, and XEX, are discussed and reasoned. Every section starts with a general inspection of how bit flips in ciphertext modify the plaintext after the decryption.

4.1 Cipher Block Chaining

Fault Propagation. Rowhammer-induced faults in CBC-encrypted PTEs or data in general appear in Figure 4.1. The figure shows a hexadecimal representation of two chunks of data: the original data on the left and the modified data on the right. Bytes with differences are highlighted. As shown in the figure, a bit in the byte at position *0x1b* has flipped. Because of the strong diffusion properties of AES [Aum17], this bit flip changes the entire block in which it occurs. The CBC construction also propagates the bit flip to the same position in the next block (see address *0x2b*, values *0x83* vs. *0x81*). The output therefore shows that exactly one bit flip occurs within this chunk. Due to this property, a single bit flip in one block may affect up to three PTEs (since one encryption block contains two PTEs).

```
0x00  d1fa 49de 23d2 8ecb | d1fa 49de 23d2 8ecb
0x08  1132 d6b0 ac1e 682b | 1132 d6b0 ac1e 682b
0x10  49ce 9686 d8b6 ff20 | 34ba dc16 4b5b 4cee
0x18  37e1 b8be 2230 dbde | 7730 a99f a5f1 656b
0x20  2148 280f 1996 3632 | 2148 280f 1996 3632
0x28  7548 b281 264f 618d | 7548 b283 264f 618d
0x30  f36b 00b0 e6bf 6685 | f36b 00b0 e6bf 6685
0x38  81d0 d9ba cc89 dad3 | 81d0 d9ba cc89 dad3
```

Figure 4.1: An example hexdiff of PTEs with CBC mode after decryption, showing the fault propagation of bit flips. Left side is the original, right side is the modified data, which also shows different bytes highlighted.

Simulation Results. The gathered results are summarized and plotted in Figure 4.2. The figure shows that the effectiveness of a Rowhammer attack on CBC-encrypted PTEs mainly depends on the MAXPHYADDR value. The results range from approximately 20% to 65% valid PTEs after decryption. In general, fewer flipped bits lead to a slightly higher percentage of valid PTEs, peaking at 8% difference when MAXPHYADDR is 40. The results become less dependent on the number of flipped bits as the MAXPHYADDR

value increases. Taking the layout of a PTE (see: Section 2.4) into account, the results are reasonable. A higher MAXPHYADDR value reduces the number of reserved bits and therefore increases the probability that all reserved bits remain zero, resulting in a valid PTE. In CBC mode, each flipped ciphertext bit also propagates to the same bit position in the next block. Consequently, a lower number of flipped bits reduces the probability that the modified bits affect reserved fields within the affected PTEs. As the number of reserved bits decreases with increasing MAXPHYADDR values, the exact number of flipped bits becomes less relevant to the overall attack effectiveness.

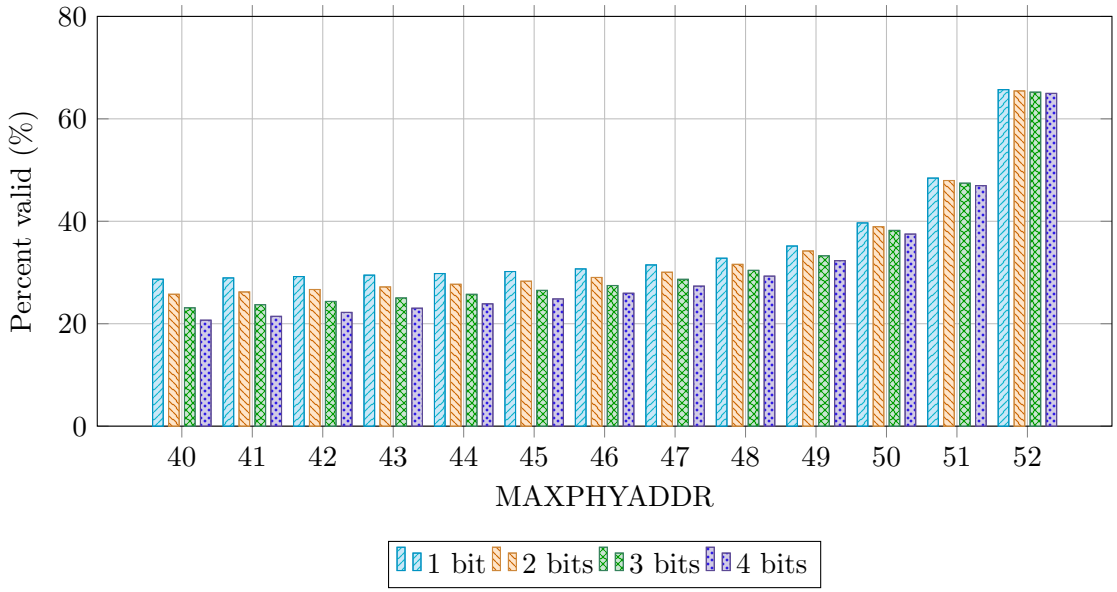


Figure 4.2: Percent of valid PTEs (after decryption) based on parameters MAXPHYADDR and number of flipped bits in the ciphertext. Mode of operation: CBC.

4.2 Counter Mode

Fault Propagation. Figure 4.3 shows Rowhammer-induced faults in CTR-encrypted PTEs and encrypted data in general. The figure presents a hexadecimal representation of two data chunks, with the original data on the left and the modified data on the right. It highlights all bytes that show a difference. As shown in the figure, three bits have flipped in the bytes at positions *0x14*, *0x1c*, and *0x1f*. Because of the construction of counter mode (see Section 2.3.1), decryption flips only the corresponding bits in the plaintext when ciphertext bits are flipped. The output therefore shows that exactly three bits have flipped in this chunk. Due to this property, a single bit flip in a block will only change one PTE. Multiple bit flips might change up to two PTEs, depending on the distribution

of the flipped bits. At a maximum, two PTEs could be modified, as the counter mode construction does not change neighboring encryption blocks.

```
0x00   55d7 1d6d ac1d d694 | 55d7 1d6d ac1d d694
0x08   df1e d966 b585 d568 | df1e d966 b585 d568
0x10   6be1 fcd7 e8e0 7ad7 | 6be1 fcd7 eae0 7ad7
0x18   c720 dc9f 2be6 65c3 | c720 dc9f 6be6 65e3
0x20   271f 17d3 f8e6 9025 | 271f 17d3 f8e6 9025
0x28   6bfc 6c83 04e5 aab2 | 6bfc 6c83 04e5 aab2
```

Figure 4.3: An example hexdiff of PTEs with CTR mode after decryption, showing the fault propagation of bit flips. Left side is the original, right side is the modified data, which also shows different bytes highlighted.

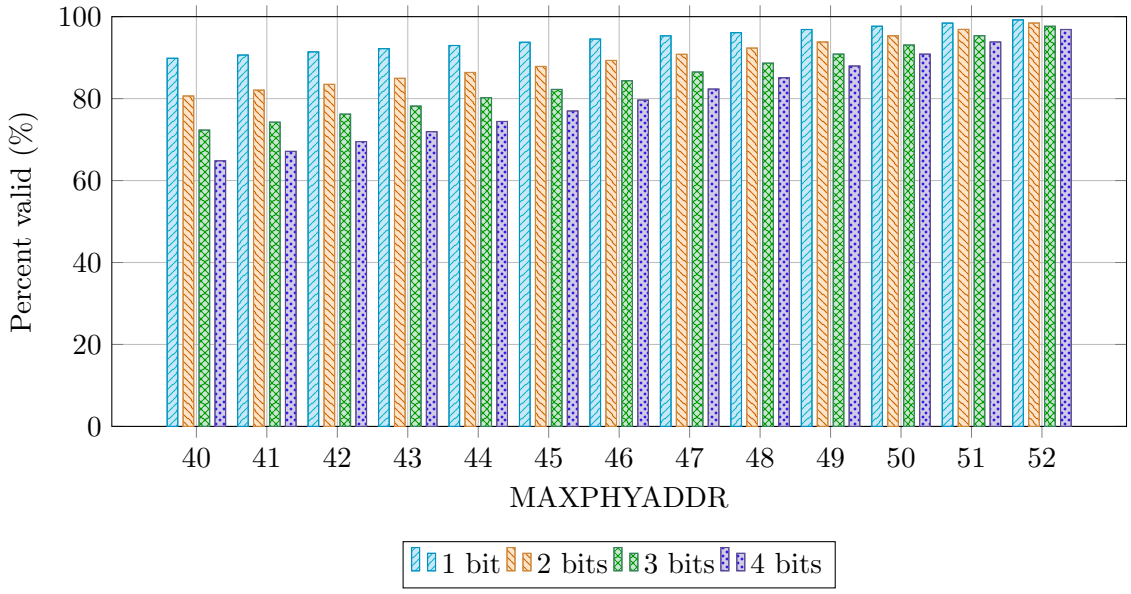


Figure 4.4: Percent of valid PTEs (after decryption) based on parameters MAXPHYADDR and number of flipped bits in the ciphertext. Mode of operation: CTR.

Simulation Results. Figure 4.4 summarizes and plots the collected results. The figure shows that the effectiveness of a Rowhammer attack on CTR-encrypted PTEs mainly depends on the number of bits flipped if the value of MAXPHYADDR is low. When the MAXPHYADDR is becoming bigger, the number of bits flipped becomes less important. The figure shows a wide spectrum ranging from as low as 64% to 99% of valid PTEs after decryption. Generally speaking, the lower the number of flipped bits is, the higher is the percent of valid PTEs. Taking the layout of a PTE (see: Section 2.4) into account, the

results are reasonable. A higher MAXPHYADDR leads to a smaller number of reserved bits, which leads to a higher probability that the chosen flipped bit is outside the section, which would make the PTE invalid. About the same reasoning can be applied to the number of flipped bits: when fewer bits are being flipped, fewer bits can end up making the PTE invalid.

4.3 Xor-Encrypt-Xor

Fault Propagation. Figure 4.5 illustrates faults induced by a Rowhammer attack on XEX-encrypted PTEs and encrypted data in general. The figure shows a hexadecimal representation of two data chunks: the original data on the left and the modified data on the right. Bytes with differences are highlighted. As shown in the figure, bits in the block starting at position *0x10* have flipped. Because of the strong diffusion properties of AES [Aum17], the entire block containing the flipped bit changes. In this example, a total of four bits have flipped, although this is not directly visible in the figure. Only the affected block changes after decryption, meaning that a single bit flip may affect up to two PTEs.

```
0x00  e97a 9208 8a57 d0f1 | e97a 9208 8a57 d0f1
0x08  fd25 fbc9 0de8 5cb8 | fd25 fbc9 0de8 5cb8
0x10  29a2 97aa aaca b12e | 425f 4bfa a367 5391
0x18  9728 e84e 6d5a d6e0 | bfd9 47c1 9ce6 7a6d
0x20  2308 2c96 e51a 8d67 | 2308 2c96 e51a 8d67
0x28  b3c5 ac1a 3a73 b297 | b3c5 ac1a 3a73 b297
```

Figure 4.5: An example hexdiff of PTEs with XEX mode after decryption, showing the fault propagation of bit flips. Left side is the original, right side is the modified data, which also shows different bytes highlighted.

Simulation Results. Figure 4.6 visualizes the collected results for XEX mode. The figure shows that the effectiveness of a Rowhammer attack on XEX-encrypted PTEs mainly depends on the MAXPHYADDR value. The results range from less than 0.1% to 50% valid PTEs after decryption. In contrast, the number of flipped bits has little influence on the outcome. For the same MAXPHYADDR value, the difference between different numbers of bit flips always remains below 0.1%. These results align with the PTE layout described in Section 2.4. A higher MAXPHYADDR value reduces the number of reserved bits and therefore increases the probability that all reserved bits remain zero, resulting in a valid PTE. Due to the diffusion property, the number of flipped bits does not significantly affect the results.

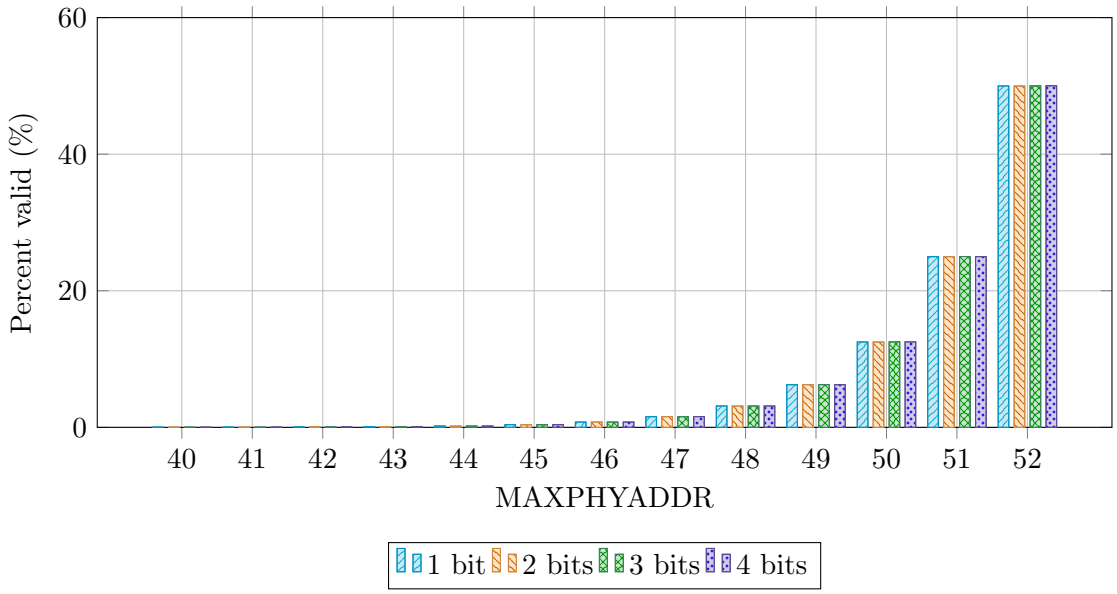


Figure 4.6: Percent of valid PTEs (after decryption) based on parameters MAXPHYADDR and number of flipped bits in the ciphertext. Mode of operation: XEX.

4.4 Evaluation Comparison

The evaluation of the three cipher modes reveals clear overall trends. The MAXPHYADDR parameter has the strongest impact on the attack effectiveness. An increase in MAXPHYADDR consistently leads to a higher proportion of valid PTEs after decryption, which improves the overall attackability. The reduced number of reserved bits in the PTE layout causes this behaviour, which increases the likelihood that flipped bits do not invalidate the entry. Consequently, systems with larger physical address spaces, such as those typically found in server and data center environments, are more vulnerable to this attack.

In addition, the importance of the number of flipped bits decreases as MAXPHYADDR increases. While a lower number of bit flips generally correlates with a higher probability of producing valid PTEs, this effect becomes less relevant for higher MAXPHYADDR values.

When comparing the cipher modes, clear differences in attackability become evident. The design of the Counter Mode (CTR) allows precise control over bit flips, since faults only affect the corresponding bits without propagating to other blocks, making it the best attackable mode. CBC is the second most attackable mode. Due to fault propagation into the next block, more PTEs are affected, which also increases the number of valid PTEs. Precise control over the flipped bits may be used to attack the next block while ignoring the current one. This approach could be used to get about the same results as with the Counter Mode. In contrast, the Xor-Encrypt-Xor (XEX) mode shows the lowest

attackability. Strong diffusion properties and no fault propagation to neighboring blocks reduce the probability of maintaining valid PTEs.

Overall, as a common ground, all evaluated encryption schemes remain vulnerable to fault attacks because they do not provide integrity protection. Although even without integrity protection and authentication, pure encryption as investigated here may be used as a relatively cheap defense against Rowhammer attacks, even though not providing 100% security.

5 Future Work and Related Targets

This thesis focused on evaluating Rowhammer effects under encrypted memory assumptions, but a few possible research paths remain open. First, the experiments should be extended to real hardware platforms, as results in this work rely on simulations. Validation on real hardware would help confirm whether simulated attacks are useful in practice.

Second, future studies should investigate biased bit flips rather than random bit flips. Evidence shows that Rowhammer faults can be controlled to a certain degree, allowing specific bits to be flipped. Considering different encryption schemes, targeted bit flips at specific locations may improve attack effectiveness. In particular, counter mode encryption could be more exploitable, where predictable modifications in ciphertext produce predictable modifications in plaintext. In contrast, other modes of operation likely offer fewer opportunities for meaningful exploitation (see Chapter 4, Fault Propagation).

Additionally, one left-out topic in this work is the investigation of how the attack behaves on non-leaf nodes in page table entries. Corrupting higher-level page table structures may be possible using the exact same approach but might offer more powerful attacks with a broader application scope. Understanding the feasibility of such an attack could broaden the threat model.

Our Rowhammer attack outlined above will very likely also work on ARM and other architectures using the same design. Taking the PTE layout from Section 2.4 into account, this attack has a slightly lower probability of succeeding than on x86 with the biggest MAXPHYADDR possible. ARM needs at least two bits to be a fixed value, whereas x86 only needs one. Compared to lower values of MAXPHYADDR it will generally be better attackable than x86. These assumptions should also be verified.

While there are different targets for Rowhammer attacks, PTEs are the data structure with the highest impact on system security. Generally speaking, every data structure is attackable, some will be easier, some harder. One interesting topic could be cryptographic keys. For example, attacking RSA [Aum17] keys by modifying the private key bits could lead to either an invalid key or a key which is very easy to factorize and the security properties are no longer met.

6 Conclusion

In this thesis we provided a simulated Rowhammer attack against unauthenticated, encrypted memory. The results showed that memory encryption alone is not a sufficient mitigation to protect systems against Rowhammer attacks on PTEs. Encryption changes the way bit flips propagate in memory, but it remains exploitable with considerable probability depending on the encryption mode used. Therefore, this work also demonstrated that Rowhammer remains a relevant threat on modern systems.

For x86 systems, this thesis evaluated the probability of obtaining valid corrupted PTEs under different encryption modes. The results showed a wide range of attack success probabilities, mainly depending on the physical address space. In CBC mode, valid PTEs were observed with probabilities between 20% and 65%. CTR mode showed even higher probabilities, ranging from 64% to 99%. XEX mode performed better overall but still remained exploitable with probabilities between 0.1% and 50%. These findings demonstrated that the choice of encryption mode had a significant influence on the feasibility of Rowhammer attacks, but none of the evaluated modes fully prevented attacks.

Although this thesis focused on the x86 architecture, similar attacks are likely possible on other architectures as well. Due to differences in page table structures and memory management implementations, we cannot provide concrete probabilities. Overall, the results indicated that pure memory encryption is not a viable mitigation against Rowhammer attacks targeting page tables, especially in systems with a large physical address space. Effective mitigation still requires either proper integrity protection mechanisms or other dedicated Rowhammer defenses in order to provide reliable protection against these attacks.

Bibliography

[Aum17] Jean-Philippe Aumasson. *Serious Cryptography*. No Starch Press, 2017.

[Ban26] Laveesh Bansal. *ARM64 Page Tables*. 2026. URL: https://kernel-internals.org/arch/arm64/page-tables.

[BDG+17] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. "CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory". In: *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 117–130. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/brasser.

[Ber+08] Daniel J Bernstein et al. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 1. Lausanne, Switzerland. 2008, pp. 3–5.

[BHSG15] Kuljit S Bains, John B Halbert, Suneeta Sah, and Zvika Greenfield. *Method, apparatus and system for providing a memory refresh*. US Patent 9,030,903. 2015.

[Cor16] Jonathan Corbet. *Defending against Rowhammer in the kernel*. 2016. URL: https://lwn.net/Articles/704920/.

[Cor25] Intel Corportation. *Intel Architecture Memory Encryption Technologies*. Revision 1.6. 2025.

[Cor26] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. 2026.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4.

[GLS18] Daniel Gruss, Moritz Lipp, and Michael Schwarz. "Another Flip in the Row: Bypassing Rowhammer Defenses and Making Remote-Rowhammer Attacks Practical". In: 2018.

[GMM16] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2016, pp. 300–321. DOI: 10.1007/978-3-319-40667-1_15.

[GSC+03] B. Gassend, G.E. Suh, D. Clarke, M. van Dijk, and S. Devadas. "Caches and hash trees for efficient memory integrity verification". In: *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* 2003, pp. 295–306.

[Gue16] Shay Gueron. *A memory encryption engine suitable for general purpose processors.(2016)*. 2016.

[HF15] Nishad Herath and Anders Fogh. "These are not your grand daddys CPU performance counters–CPU hardware performance counters for security". In: *Black Hat Briefings* 47 (2015), p. 48.

[IES16] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. "MASCAT: Stopping microarchitectural attacks before execution". In: *Cryptology ePrint Archive* (2016).

[JLK+23] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. "CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 1702–1718.

[JVF+22] P. Jattke, V. Veen, P. Frigo, S. Gunter, and K. Razavi. "BLACKSMITH: Scalable Rowhammering in the Frequency Domain". In: (2022), pp. 716–734.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ACM SIGARCH Computer Architecture News* 42.3 (2014), pp. 361–372.

[Lim19] Arm Limited. *Armv8-A Address Translation*. 2019.

[LSR+20] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *2020 IEEE European Symposium on Security and Privacy Workshops*. 2020, pp. 710–719. DOI: 10.1109/EuroSPW51379.2020.00102.

[MK19] Onur Mutlu and Jeremie S Kim. "RowHammer: A Retrospective". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.8 (2019), pp. 1555–1571.

[Rog04] Phillip Rogaway. "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC". In: *Advances in Cryptology - ASIACRYPT 2004*. Ed. by Pil Joong Lee. 2004.

[SD15] Mark Seaborn and Thomas Dullien. "Exploiting the DRAM rowhammer bug to gain kernel privileges". In: *Black Hat* 15.71 (2015), p. 2.

[SNR+18] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. "Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories". In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, pp. 416–427.

[TSB18] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. “VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures”. In: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2018.

[Wik26] Wikipedia. *Disk encryption theory*. Online; accessed 01-February-2026. 2026. URL: `http://en.wikipedia.org/w/index.php?title=Disk%5C%20encryption%5C%20theory&oldid=1323276525`.

[XZZT16] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. “One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation”. In: *25th USENIX security symposium (USENIX Security 16)*. 2016, pp. 19–35.

[YEP+06] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. “Improving Cost, Performance, and Security of Memory Encryption and Authentication”. In: ISCA ’06. IEEE Computer Society, 2006, pp. 179–190. ISBN: 076952608X.