FAULT+PROBE: A Generic Rowhammer-based Bit Recovery Attack

Kemal Derya
Worcester Polytechnic Institute
kderya@wpi.edu

M. Caner Tol
Worcester Polytechnic Institute
mtol@wpi.edu

Berk Sunar
Worcester Polytechnic Institute
sunar@wpi.edu

Abstract

Rowhammer is a security vulnerability that allows unauthorized attackers to induce errors within DRAM cells, e.g., to attain elevated user privileges or to extract sensitive information from cryptographic schemes. To prevent fault injections from escalating to successful attacks, a widely accepted mitigation is implementing fault checks on instructions and data.

We challenge the validity of this assumption by examining the impact of the fault on the victim's functionality, as opposed to solely focusing on the victim's erroneous output. Specifically, we illustrate that an attacker can construct a profile of the victim's memory based on the directional patterns of bit flips. This profile is then utilized to identify the most susceptible bit locations within DRAM rows. These locations are then subsequently leveraged during an online attack phase with side information observed from the change in the victim's behavior to deduce sensitive bit values. Consequently, the primary objective of this study is to utilize Rowhammer as a probe, shifting the emphasis away from the victim's memory integrity and toward the examination of the victim's operational behavior.

In the signing victim attack scenario, we show FAULT+PROBE may be used to circumvent the verify-after-sign fault check mechanism, which is designed to prevent the generation of erroneous signatures that leak sensitive information. It does so by injecting directional faults into key positions identified during a memory profiling stage. The attacker observes the signature generation rate and accordingly decodes the secret bit value. This circumvention is enabled by an observable channel in the victim. FAULT+PROBE is not limited to signing victims and can be used to probe secret bits on arbitrary systems where an observable channel is present that leaks the result of the fault injection attempt. To demonstrate the attack, we target the fault-protected ECDSA in wolfSSL's implementation of the TLS 1.3 handshake. We recover 256-bit session keys with an average recovery rate of 22 key bits/hour and a 100% success rate.

1 Introduction

Rowhammer. The discovery of Rowhammer by Kim *et al.* [27] led to a number of attacks, such as gaining kernel privileges by corrupting Page Table Entries [44], bypassing authentication using opcode flipping [15]. Rowhammer has been shown to be a threat in cloud environments [9, 55] and mobile platforms [50]. Rowhammer has also been shown to work from Javascript [12, 16], extending the attack surface to include browsers. To mitigate Rowhammer, detection-based [3, 8, 11, 17, 19, 20, 41, 56], and prevention based [5, 16, 50] countermeasures have been proposed. Gruss *et al.* [15] showed each one of these can be defeated. Similarly, row-refresh-based mitigation attempts implemented in DDR4 chips have been easily bypassed [14, 24, 29]. Cojocar *et al.* [10] showed how to induce bit flips on Error-correcting-code (ECC) memories using the timing side channels.

Attacks on Signature Schemes. Beyond system-level attacks, cryptographic schemes have also been more directly targeted using Rowhammer. For instance, [23] uses Rowhammer faults to recover key bits in the LUOV post-quantum (PQ) signature scheme (a Round 2 competitor in the PQ NIST standard competition). The technique was formalized and named *Signature Correction Attack* (SCA) in [43], which targeted CRYSTALS-Dilithium (PQ NIST Standard). In a nutshell, SCA introduces faults using Rowhammer while signatures are being computed, and by mathematically tracing and correcting the faulty signatures, it is able to recover key bit values. Later, the same attack with modification of the correction mechanisms was shown to work in traditional signatures schemes, e.g., ECDSA, EdDSA, and RSA signatures in [37]. The SCA on Dilithium was further improved by using (non-Rowhammer) instruction skip attacks in [30].

Attacks on KEMs. Similar to signature schemes, Key Encapsulation Mechanisms (KEM), or less formally public key encryption schemes, have also become a target of Rowhammer-enabled attacks. For instance, [13] targeted FrodoKEM (another NIST PQC submission). The attack uses Rowhammer

to increase the error rate during encryption, which results in higher decryption failures, which are then analyzed to deduce key bits. A further study [36] introduced a more generic key-recovery attack on LWE-based KEM schemes using Rowhammer. Similar to most physical attacks targeting lattice-based KEM schemes, the chosen-ciphertext attack (CCA) attack works by running the decapsulation procedure or the plaintext checking oracle multiple times with different ciphertexts while introducing faults that reveal some part of the key. The Rowhammer-enabled attack was demonstrated to reduce the number of plaintext checks on the Kyber and Saber PQ schemes.

Library Countermeasures. To mitigate Rowhammer-enabled fault injection attacks on crypto schemes, e.g. [13, 23, 36, 37, 43], most authors recommend hardware mitigations, e.g. increased refresh rates. Further software-based fault detection techniques are recommended, i.e., SCA papers [23, 37, 43] advocate checking faulty signatures before releasing them to potential adversaries. Without access to the faulty signatures, an adversary cannot run the signature correction step. Further, in the KEM scenario [13] recommends as an active defense for a potential victim to check the distribution of received ciphertexts during the decapsulation process to detect the presence of an attacker while also cautioning on the possible preventive actions. As for industry response, due to the lack of effective and efficient hardware mitigations in DRAM chips, library developers started to patch their own software with application-specific hardening mechanisms, e.g., `SUDO` developers implemented a logic hardening that prevents authentication bypass [1] and WolfSSL developers implemented a *verify-after-sign* check to mitigate signature-correction attacks [37]. In light of the recent advances in Rowhammer attacks, we pose the following questions:

- *Are there any observable channels when combined with Rowhammer faults that might be exploited to recover secrets?*
- *If granted, how can we use such channels along with Rowhammer to probe and recover secret bits from the victim's memory?*
- *Can such an attack be used to bypass existing detection-based defenses?*

1.1 Our Contributions

This work introduces a generic technique that enables an attacker to probe bit values in a victim's memory space using Rowhammer faults. It does so by observing behavioral changes in the victim's execution during fault injection. These behavioral changes confirm a successful fault being injected (or otherwise) and provide a feedback mechanism.

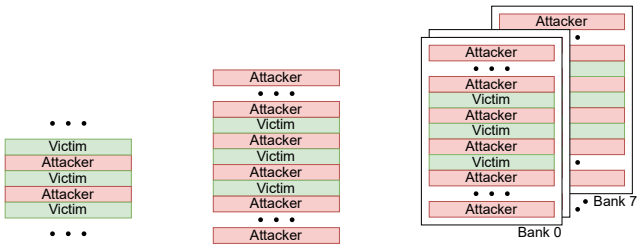
Challenges. To use this side channel for an end-to-end attack, one needs to overcome the following challenges:

- Finding targets with observable feedback mechanisms when a fault is introduced on the secret value,
- Finding physical pages in the memory that produce repeatable flips in a target offset yet do not produce too many flips in other offset locations,
- Mapping the victim to target flippy page and hammering,
- Amplifying the correct prediction probability of the secret bit by reclaiming the same flippy page and repeating the attack.

Contributions. To solve the challenges we explained,

- We introduce FAULT+PROBE, a novel fault-based side-channel leakage mechanism that exploits behavioral changes in a victim program caused by Rowhammer faults to recover secret bits on co-located platforms. (Section 4)
- We introduce three kinds of behavioral changes that serve as feedback mechanisms observable by the adversary:
  1. Fault-checking mechanisms that were put in place to prevent Rowhammer from succeeding may become leakage channels. For example, a signature generation scheme naturally slows down if fault detection is implemented and faults are injected during signing. The adversary can measure the signing rate to deduce whether fault injection attempts succeed or not.
  2. Many protocols report failures to assist diagnosis. For instance, wolfSSL's TLS implementation returns connection error codes, providing information about a fault success, which the attacker might exploit with greater accuracy.
  3. Finally, without any error code or slowdown in the signing rate, the adversary can potentially receive a faulty signature, which fails to be verified by the public verification algorithm, indicating a fault on the victim's side to the adversary[1].
- We provide extensive experimental evidence using a thorough offline profiling phase on DDR4 memory devices that identifies memory pages with stable but isolated directional flips. We show that certain memory locations in DRAM modules have low-noise and repeatable flips, which enables FAULT+PROBE. (Section 4.3)
- We propose a novel method to reclaim flippy pages that enables us to reuse them multiple times for hammering a secret value as explained in Section 4.3.4.
- We demonstrate an end-to-end attack on a TLS 1.3 handshake protocol implemented in wolfSSL. We use wolfSSL's ECDSA implementation, which is specifically protected against fault injections with the *verify-after-sign* check mechanism. We show that FAULT+PROBE can recover 256-bit ECDSA private key from wolfSSL with an average recovery rate of 22 bits/hour and 100% success rate. (Section 6)

[1] Unlike in Signature Correction Attacks, FAULT+PROBE does not need to correct the signature but only runs verify primitive to determine the success or failure of the fault.



(a) Double-sided (b) N-sided (c) N-sided multi-bank

Figure 1: Different Rowhammer hammering configurations

2 Background

Rowhammer. Rowhammer is a security vulnerability discovered by Kim *et al.* [27] and used in an array of attacks, e.g., to gain kernel privileges by corrupting Page Table Entries [44] and using opcode flipping [15] to bypass authentication. Previous work introduced the building blocks of an end-to-end Rowhammer attack, such as finding contiguous physical memory [21, 26], and memory massaging [7, 31] which enabled more precise Rowhammer attacks [45].

Rowhammer has also been shown to be a threat in cloud environments [9, 55] and mobile platforms [50], and to work over network connections [33, 46], and even from to be effective from Javascript [12, 16], increasing the attack surface to include browsers.

Weissman *et al.* [53] showed up to three times faster attack times in heterogeneous FPGA-CPU platforms when Rowhammer is launched from the FPGA. Tobah *et al.* [47] showed Rowhammer can increase the number of potential Spectre-V1 [28] gadgets by 200 times. Recently, Kang *et al.* [26] introduced a multi-bank hammering method that increases the throughput of bit flips up to seven times. Adiletta *et al.* [1] demonstrated that Rowhammer can also be used to corrupt register values temporarily stored in memory.

N-sided Rowhammer. Fig. 1 displays the arrangement of memory in different Rowhammer setups, in which each DRAM row comprises two pages of 4KiB each. In these configurations, it is presumed that the attacker can manipulate pages containing known data, referred to as attacker pages. The rows designated for the victim hold secret data from the victim's process. As depicted in Fig. 1a and 1b, all rows are located within the same bank. However, in Fig 1c, rows are allocated on different banks. The attacker and victim pages are positioned adjacently. Consequently, frequent accesses to the attacker pages can cause bit flips in the victim pages.

Mitigation Techniques. Detection [3, 8, 11, 17, 19, 20, 41, 56], and prevention based [5, 16, 50] countermeasures have been proposed to mitigate Rowhammer attacks. Yet, Gruss *et al.* [15] showed each one of these can be defeated. Cojocar *et al.* [10] showed how to induce bit flips on Error-correcting-code (ECC) memories using the timing side channels. Similarly, row-refresh-based mitigation attempts implemented in DDR4 chips have been easily bypassed by adopting different hammering patterns [14, 24, 29]. Recently proposed mitigations, such as [25, 34, 52], require hardware design changes, and they have not been adopted by DRAM vendors yet.

3 Threat Model

Following the prior work on Rowhammer-based attacks [9, 15, 16, 23, 27, 37, 55] and microarchitectural side-channel attacks [6, 28, 32, 48, 51], attacker and victim are co-located in the same system in our threat model. The adversary has no physical access to the processor or memory system but can run code on the target machine. We assume the system is free of any software vulnerability, and all the protections against microarchitectural side-channel attacks are deployed. We assume the attacker can access a mechanism that tells if the secret has been corrupted. Such a mechanism can naturally exist in different applications without any attacker effort. For instance, the signature verification algorithm in public key crypto schemes can serve as an oracle if a fault was injected on secret or not [37]. In case the victim does not reveal faulty signatures, the error code that is returned to the attacker can do the same task. In extreme cases where the victim does not return any error code and retries the correct execution, the timing difference caused by extra execution can reveal if the fault was injected successfully.

4 FAULT+PROBE Attack

Earlier Rowhammer attacks depended on introducing faults into the victim's process. The attacks in [23, 37, 43], i.e. applied correction techniques to faulty values, extracting secret bits from the victim's side. Mitigation of these attacks has been attempted through check mechanisms such as the *verify-after-sign* method, aiming to prevent the generation of faulty signatures [54]. Nevertheless, this work demonstrates that the FAULT+PROBE attack can still access secret bits even when such check mechanisms are in place. This enables unprivileged attackers to read secret bits by monitoring the flip rate of Rowhammer-induced bit flips at specific memory locations.

4.1 Attack Overview

The FAULT+PROBE attack exploits the reproducibility of bit flips on DRAM rows through meticulous profiling. The secret bits are probed by repetitively flipping them. Fig. 2 shows the attack overview. Our attack is divided into the steps below.

❶ We find a contiguous memory block and hammer it to find flippy bit locations. This stage is referred to as *offline memory*

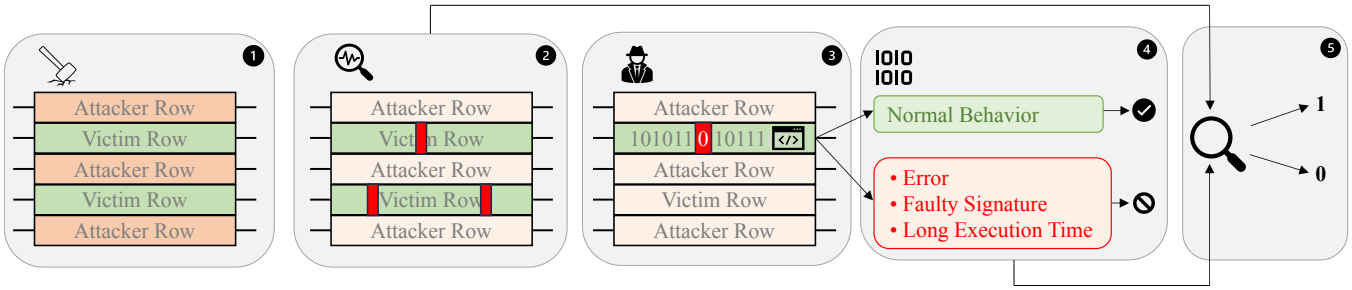


Figure 2: FAULT+PROBE Attack overview. ❶ The attack starts by hammering adjacent rows for the offline memory profiling phase. ❷ The most sensitive bit offsets are profiled. ❸ The victim process is located on the profiled row. ❹ The process is run and hammered during the online attack phase for a sufficient number of time. ❺ The value of the targeted offset is found by comparing the results from the offline memory profiling phase and the online attack phase.

profiling. This stage is performed until a sufficient number of victim pages for the online attack is found. (Section 4.3)

❷ We analyze the victim pages to observe the directional bit flips. We construct a profile based on the bit flips. The most sensitive bit locations within victim pages are identified. (Section 4.3.3)

❸ We choose a victim page from the profile. This victim page has reproducible bit flips on specific bit locations. Before going into the online attack phase, we search the memory space for the victim page with its corresponding attacker pages. We allocate dummy pages to place the secret bits into the victim page for the online attack phase. We manipulate the Linux Buddy Allocator and unmap the dummy pages to map the secret bits to the victim page [31, 37]. We target the secret bits using Rowhammer for the online attack phase. (Section 6.2)

❹ We look at the changes in the victim's behavior. We record the observable results after repeatedly running the victim process and hammering the victim page. (Section 6.2)

❺ We compare the victim page's profile with the operational behavior of the victim. Then, we probabilistically determine the original secret bit. (Section 6.2)

Reproducible Bit Flips. Reproducible bit flips refer to the ability to consistently induce specific changes in memory content through controlled Rowhammer attacks. By carefully targeting and repeatedly accessing specific rows of DRAM, attackers can trigger predictable bit flips, potentially leading to unauthorized access or system compromise [1]. Our approach achieves as low as single-bit reproducible flips within the same row. We fine-tune the attack parameters to minimize the impact on neighboring bits. This work shows the precision and control of Rowhammer attacks for improved bit probing.

4.2 Utilizing Reproducible Bit Flips

This section details the method for utilizing reproducible bit flips to indirectly infer the contents of the victim's bits, bypassing the need to mathematically trace the fault directly.

Activating Attacker Pages. Executing a hammering routine on pages controlled by the attacker leads to interference within the victim's pages. The hammering routine plays an important role in reproducible bit flips. The number of bit flips is affected by the number of attacker pages. Moreover, we observe that the position of the victim page among the attacker pages affects the number of bit flips on the victim page. Even the data stored in the attacker pages affects the distribution of bit flips on the victim page. Therefore, we need to activate the attacker pages by using a hammering routine that generates precise and reproducible bit flips.

Deducing Secret Bits. Prior to the attack, victim pages undergo a profiling stage to ascertain the flipping tendencies of individual bits. A victim page that shows precise and reliable bit flips is chosen from the profiling stage. Then, the online attack is started by locating the victim process on the victim page, which is hammered repetitively.

The efficacy of the attack is monitored by examining the disturbance in the victim's processes. Given that bit flips do not occur with every attempt, there is a calculable probability associated with the success of the attack. Concurrence between the outcomes of the profiling phase and the online attack iterations allows for the inference of the probed bit.

4.3 Offline Memory Profiling

This section describes the memory profiling techniques to perform FAULT+PROBE attack explained in Section 4.

4.3.1 Finding Physically Contiguous Memory Pages

To position attacker rows in proximity to the victim rows, our approach starts by identifying contiguous memory addresses. One technique to find contiguous physical memory involves the use of huge pages, ensuring that sequential virtual addresses correlate directly with contiguous physical addresses. However, this method necessitates unique system configurations. Operating under the constraint of standard configurations, we leverage DRAMA [42] on DRAM modules. For

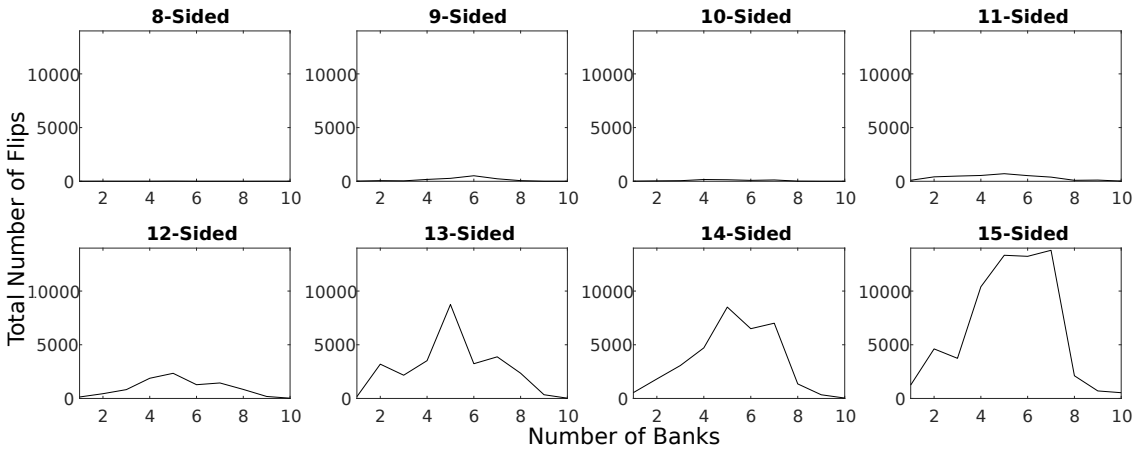


Figure 3: Optimal configuration search for the most number of bit flips.

an in-depth understanding of the DRAMA exploit, we direct readers to the dedicated section detailing its implementation. Other alternatives for finding contiguous physical memory segments from user space include [21] and [26].

4.3.2 Finding Adjacent Rows on DRAM

We start by allocating a memory chunk from the system. On this chunk of memory, we search for the memory space that is located on the same bank. Depending on the number of banks we need to allocate, we gather each memory space associated with its bank into a memory array. The memory array contains the physical addresses of memory chunks located in different banks. We use the method presented in [42] to find the memory chunks located in the same bank.

Once memory blocks on different banks have been allocated, our next step involves identifying pages that are adjacent within the same DRAM bank. To achieve this, we employ the method in [42], which reveals the row number of physical addresses in memory chunks on different banks. In each bank, we look for physical addresses that are either located in the same row or in the next row. The number of adjacent rows might be different for each bank. We truncate the number of adjacent rows to the minimum number of rows found in a bank. If we do not find any adjacent rows in the same bank, we start the process from the beginning by mapping a memory chunk from the system.

4.3.3 Profiling the Contiguous Memory

Upon successfully arranging rows within the same DRAM bank of a contiguous physical memory block, we executed an effective Rowhammer attack. In systems equipped with DDR3 DRAM, we employed a double-sided Rowhammer technique, hammering each victim row with an attacker row on both sides. This approach, however, is rendered ineffective on DDR4 systems due to Target Row Refresh (TRR) countermeasures. Consequently, we adapted to an N-sided Rowhammer strategy [14, 24, 26], aiming at multiple victim rows with a sequential pattern of attacker and victim rows. Subsequently, we pinpointed the rows experiencing bit flips by persistently hammering the attacker rows and monitoring for changes in the victim rows' data.

Multi-bank Hammering. The number of banks used in hammering has a direct impact on the quantity of flips observed. Our observations reveal that employing a multi-bank approach generates a higher number of bit flips compared to a single bank. Kang *et al.* [26] introduced the multi-bank hammering method. We use their method to observe the effect of multi-bank hammering. We utilize Linux's huge pages to target the same DRAM pages for a more fair comparison between the configurations. We need to target the same pages so that we would have more understanding of the number of bit flips with different numbers of attacker rows and different numbers of banks. Fig. 3 shows the total number of bit flips with different numbers of attacker rows and multi-bank hammering on the same DRAM pages. The number of bit flips refers to the number of total flips from $1 \rightarrow 0$ and $0 \rightarrow 1$ flips. Fig. 3 indicates that increasing the number of banks increases the number of flips observed until eight banks in most cases. It decreases dramatically after using more than eight banks.

Memory Profiling. We set the attack window, which is dictated by the number of rows and the number of banks. We sweep contiguous chunks of memory on each bank by using the attack window, which contains the attacker and victim rows. Firstly, we populate the victim rows with 0s and the attacker rows with 1s. Then, we hammer the attacker rows by 500K times to observe the bit flips on the victim rows. Secondly, we swap the values of attacker and victim rows and hammer them to observe the bit flips. If we find any bit flips on the victim rows, we use them for memory profiling. If we do not observe any bit flips during this phase, we pass to the

next attack window on the contiguous memory chunk on each bank.

The data pattern within the victim row influences the bit flip rate at specific positions. Typically, bit configurations of 1-0-1 and 0-1-0 are more prone to flips than those of 1-1-1 and 0-0-0 [27] when the central bit is the victim bit.

To simulate realistic flip rates, the victim rows are filled with random data during the profiling phase. The process begins by setting the attacker rows to all 1s. Subsequently, the victim rows, now containing random data, are hammered. This is followed by reinitializing the attacker rows to all 0s and, once again, hammering the victim rows with the same random values. This cycle is repeated 200 times. For each cycle, we note down the bit flips on the victim rows. These bit flips are analyzed in a later stage of the attack.

In the online attack scenario, the flip rates might be different than the offline profiling phase because of the presence of the victim process in the memory. To achieve more realistic flip rates, the offline profiling phase can be performed with an attacker's copy of the victim process located at the victim rows. (Section 6.1)

4.3.4 Amplifying Leakage

The pages with page offsets that flip at every hammering attempt are rare. Therefore, a single Rowhammer session can only reveal the secret value with a certain probability. Moreover, the number of noisy bit flips within the page, as well as the other probabilistic factors, such as memory massaging, lowers the probability of correctly predicting the secret bit. Hence, we need to amplify the probability by repeating the attack on the same secret value multiple times. Fig. 4 shows that there is a high variance in how many times a bit flip will be observed in a given Rowhammer iteration. The profiled page in this figure shows a flip characteristic where a flip in a given target offset is more likely to occur than any other offset. We observe that after 200 iterations in the memory profiling, the variance in the flip rates of both the target and other offsets converge. Therefore, we use 200 iterations in the following experiments for the memory profiling and online attacks.

Once the attacker profiles the memory and finds relatively less noisy pages, these pages are unmapped so that the secret in the victim process is placed on the same flippy page. Yet, reclaiming the same physical page without any privilege is a challenging task due to virtual to physical address translations, which are opaque to users.

Here, we propose a novel method to reclaim the flippy pages without using Linux `pagemap`, which requires root privileges or huge pages that require special system configurations.

- After an iteration of the online attack is completed, the victim program releases the flippy page.
- Then, we allocate a large buffer of pages. As the page frame cache provides the recently used pages to the new allocations, the flippy page is highly likely to come back within the buffer.
- We cannot see the physical address of the flippy page to search within the buffer, Therefore, we run another round of Rowhammer on the same attacker rows that we used for the online attack and search for a bit flip in the large buffer.
- Since we know the original flippy page offsets in the previously released page, we can then compare those flippy offset numbers with the actually flipped offset in the buffer.
- If we cannot see a flip in the buffer in the first trial, we try hammering more. If it still fails, we increase the size of the buffer by mapping more pages and repeating the process until we can reclaim the victim page.

In our experiments on the DDR3 system, we observed that if the flippy page is in the allocated buffer, we can detect it by seeing a flip in the first few trials of Rowhammer, if not in the first trial.

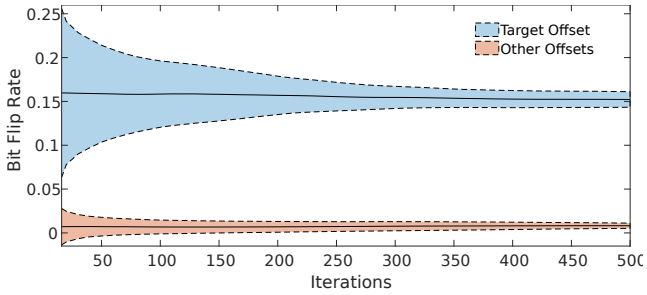


Figure 4: The variance and mean of bit flip rates for the target offset and other offsets in a single page for an increasing number of iterations. The separation between the target and other offsets allows us to easily distinguish the two based on the observed rate.

5 Experiment Results

Experiment Setup. To execute our online attack, we employ offline memory profiling on the target system to pinpoint the most effective pages for the attack. This experiment is performed on a Ubuntu 20.04.6 LTS system with Intel Core i5-8400 CPU@2.80 GHz with Corsair DDR4 DRAM module with part number CMU64GX4M4C3200C16.

Table 1 shows the top 10 most number of flips observed on different configurations. Increasing the number of banks increases the number of pages hammered and the number of flips observed. Moreover, the number of attackers significantly affects the number of observed flips on the same memory region. Thus, it increases the throughput of the multi-bank hammering process.

We profile the memory with different configurations on the number of attacker rows and the number of banks, i.e., **(R, B)**. We choose **(15, 7), (15, 5), (15, 4), (12, 7)** for the offline

Table 1: Impact of Hammering Configuration on Bit Flips.

# Attacker Rows	# Banks	# Flips
15	7	13796
15	5	13333
15	6	13234
15	4	10398
13	5	8750
14	5	8508
14	7	7008
14	6	6496
14	4	4709
15	2	4616

memory profiling phase based on Table 1. **(15, 7)** and **(15, 5)** configurations result in similar number of bit flips while **(15, 4)** configuration has less number of bit flips. We also use **(12, 7)** configuration to compare the result from a configuration that is not shown in Table 1.

The offline memory profiling phase is performed as explained in Section 4.3.3 with chosen **(R, B)** configurations. Table 2 shows the number of flippy pages under different profiling configurations. A page is referred to as a flippy page if at least one bit-flip is found during the offline memory profiling phase.

We observe that profiling the same physical address with the same hammering configuration results in the same bit-flip pattern. The number of bit flips might differ in different offline profiling phases, whereas the most flippy bit location stays the same. Furthermore, the bit flip pattern on a physical address is affected by the hammering configuration. The most flippy bit location changes for the same physical address with different hammering configurations. Thus, we can use the same physical address for probing different bits by using different hammering configurations. Some of the flippy pages in Table 2 are the same physical addresses with different flip characteristics.

We profile more memory area with **(15, 7)** configuration than **(15, 5)** configuration. However, the number of flippy pages per MB is lower in **(15, 7)** configuration compared to **(15, 5)** configuration. Different parts of memory are allocated for offline memory profiling in different configurations. We allocate memory area which is less prone to have bit flips in **(15, 7)** configuration.

We profile 42% more area with **(15, 4)** configuration than **(12, 7)**. We find 69% more flippy pages in **(15, 4)** configuration. This also lays emphasis on the memory area allocated for the offline profiling phase.

Post-processing analysis is then applied to the collected bit flip data, producing Table 3 that illustrates the number of bit flips for random 10 flippy pages that were found in the profiling phase. Table 3 shows the most flippy bit location on each profiled DRAM page, where each DRAM page consists

Table 2: Analysis of flippy pages in various hammering configurations that highlights how specific configurations impact the likelihood of inducing flippy pages.

Configuration (R, B)	Profiled Area (MB)	# Flippy Pages	Flippy Page / MB
(15, 7)	340.38	29,085	85.44
(15, 5)	296.25	27,270	92.05
(15, 4)	46.50	4,788	102.96
(12, 7)	32.70	2,827	86.45

Table 3: Distribution of bit flips across physical addresses, highlighting the most flippy bit location and the comparative number of flips on the identified flippy bit versus other bits.

# Physical Address	Most Flippy Bit Location	# Flips on Target Offset	# Flips on Other Offsets
2c0046000	2218	116	0
2bfa62000	7210	73	0
1ca224000	522	45	0
1e7598000	11347	49	17
135dd0000	7201	62	18
1cd7fc000	981	41	37
2c0ae1000	26034	42	28
2c1c8c000	1349	80	605
2c1d12000	9049	172	1153
2c1d10000	16076	154	2861

of 4KB, which is 32768 bits. The distribution of bit flips on DRAM pages varies, which is indicated by the number of total bit flips on the other bits.

A detailed analysis of the distribution of flips on 4 different pages is shown in Fig. 5. Some pages have distinctive bit flips on certain bit locations. It should be noted that we see the same bit flip direction on an offset while hammering it with 0 and 1. This would make it possible to probe the targeted bit regardless of the hammering values. Fig. 5 shows the total number of bit flips that occurred while hammering each page with 0 and 1. Fig. 5a and 5b show reliable bit flips on consistent bit offsets, while Fig. 5c and 5d have bit flips distributed over different offsets. Later on, we split profiled pages into different categories based on the number of bit flips. For our attack, we utilize pages that show similar bit flip patterns as shown in Fig. 5a and 5b.

6 Recovering wolfSSL TLS Server Key

Our demonstration of the attack uses wolfSSL TLS 1.3 handshake protocol shown in Fig. 6. It is important to note that in a practical attack scenario, offline memory profiling should cease once appropriate pages for the attack are found. Subsequently, the focus should shift to the online phase of the attack. For our online attack, we extensively profiled the memory setup to locate optimal pages for the attack.

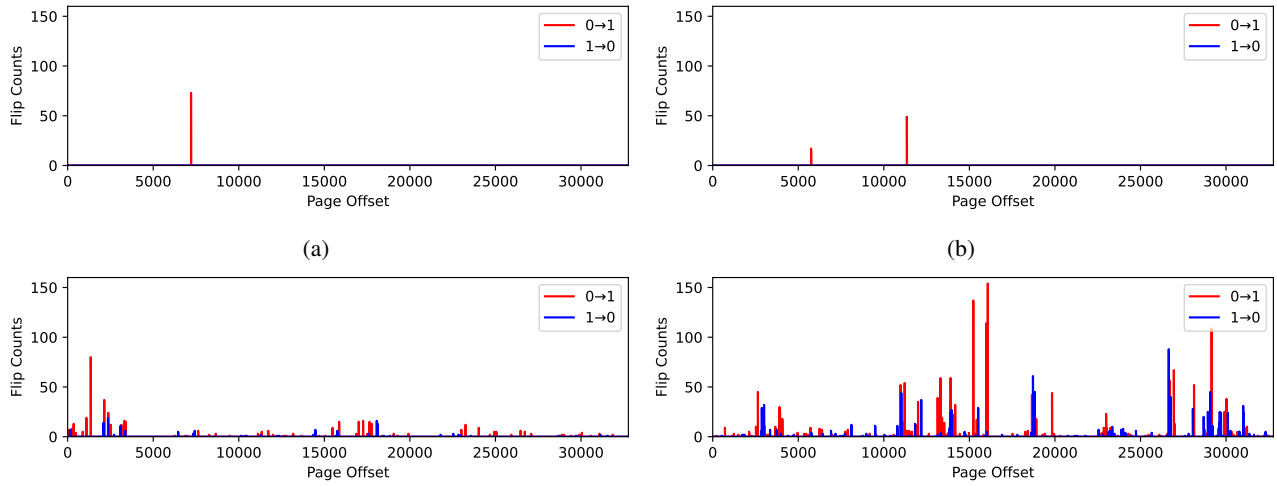


Figure 5: Location and direction of bit flips on different pages. (a) and (b) are examples of *reliable* pages, (c) and (d) are examples of *unstable* and *unusable* pages, respectively.

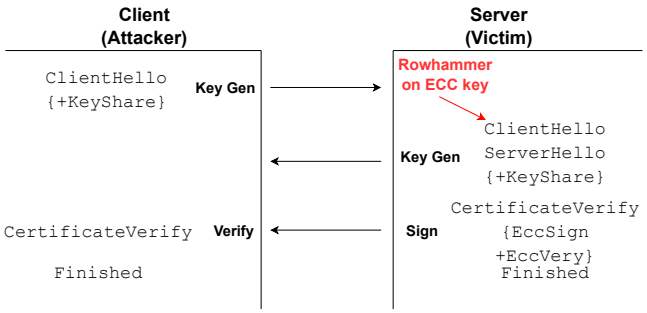


Figure 6: Rowhammer attack on TLS 1.3 handshake

Previously, wolfSSL addressed a Rowhammer attack that leveraged defective signatures to compromise the server's private key. The signature correction method applied to these signatures made it feasible to deduce the secret 256-bit ECDSA key [37]. As a countermeasure, wolfSSL added an additional verification step to the handshake protocol, involving a pre-send verification of signatures [54]. This step ensures that if a signature is found to be defective, the handshake, and consequently the SSL connection, does not occur. This step is shown as **EccVery** on the server side in Fig. 6. This method is referred to as the *verify-after-sign*. However, this fix inadvertently opens up ways for alternative attack strategies. Now, it is conceivable to infer the state of any targeted key bit by monitoring the failure rates of SSL connections during Rowhammer attacks. TLS 1.3 handshake protocol is summarized below.

- The client starts the handshake by sending `ClientHello+KeyShare` to the server.
- The server responds the client by sending `ServerHello+KeyShare` to the client. It also calculates the signature of a handshake and sends it to the client under `CertificateVerify` step. Optionally, it verifies the signature before sending it to the client. Thus, any defective signature is not sent to the client before the termination of handshake. Signature verification is optional and added as a countermeasure against a prior attack [37].
- Once the client receives the response, it calculates its signature and verifies it. Then, the handshake is completed and the secure connection is settled between the client and the server.

If an attacker successfully injects fault into the ECDSA key, there are two possible ways to recover the secret bit.

- If the attacker gets the faulty signature from the server, a signature correction method can be used to recover a bit from the ECDSA key [37].
- If the attacker does not get the faulty signature and observes that the handshake is terminated by the server, it is possible to recover a secret bit by repetitively sending handshake requests and injecting faults to the ECDSA key. Then, the attacker needs to observe the connection handshake termination rate to recover a bit from the ECDSA key.

6.1 Memory Profiling Phase

To provide a comprehensive summary of the offline memory profiling phase, we divide flippy pages found in Table 2 into different categories based on the distribution of flips. Before categorizing the pages, we take sample pages from the flippy pages, which have bit flips distributed as shown in Fig. 5. Then, we observe the handshake failures over different pages and categorize them to determine a criterion for pages that are useful for the online attack.

Table 4: Classification of memory pages into reliable, unstable, and unusable categories based on the distribution of bit flips across different hammering configurations.

Configuration (R, B)	Reliable Pages	Unstable Pages	Unusable Pages
(15, 7)	685	2076	26324
(15, 5)	218	570	26482
(15, 4)	24	56	4708
(12, 7)	33	128	2666

We define δ as the most number of bit flips observed on a single bit offset on a flippy page. We define σ as the total number of bit flips observed on all bit offsets on a flippy page except the most flippy bit location.

Some flippy pages have a very low number of bit flips, i.e., $\delta \leq 10$ and $\delta \geq \sigma$. On the other hand, some flippy pages have a higher number of bit flips where $\sigma \geq 80$. We observe that it is not possible to inject a reproducible fault on the ECDSA key on these pages. We either see no bit-flips on the target offsets or lots of bit-flips on the other bit offsets. Thus, we refer to these pages as **unusable pages**.

We use flippy pages where $\delta \geq 10$ and $\delta \geq \sigma$ and perform an online attack. We observe that the handshake is terminated, and bit flips occur on the targeted bits on these pages. We categorize these flippy pages where $\delta \geq 10$ and $\delta \geq \sigma$ as **reliable pages**.

We use flippy pages where $\delta \geq 10$ and $\sigma \leq 80$ to perform the online attack. We observe that these pages can be used for the attack. However, they do not always result in consistent bit flips on the ECDSA key. Thus, it might give us a wrong conclusion of the real bit value. We categorize these pages as flippy pages where $\delta \geq 10$ and $\sigma \leq 80$ as **unstable pages**.

Table 4 shows the profiled pages under different categories. There are more **reliable** and **unstable** pages in **(15, 7)** configuration compared to the **(15, 5)** configuration. This shows that **(15, 7)** configuration profiles pages that have consistent and reproducible bit-flips as shown in Fig. 5a and 5b.

(15, 4) configuration have less number of **reliable** and **unstable** pages compared to the **(12, 7)** configuration. **(15, 4)** configuration profiles more flippy pages. However, it either induces a very low number of bit-flips or a very high number of bit-flips on flippy pages, which are referred to as **unusable**. Therefore, we get more useful pages for the online attack from the **(12, 7)** configuration.

This analysis aids in identifying potential pages for targeting specific bit offsets, which we then repeatedly hammer in our online attack phase.

Extending the Offline Profiling for Higher Confidence. The presence of wolfSSL TLS server in the online attack causes some of the profiled pages to have lower flip rates than the offline profiling phase. This difference decreases the confidence in our probing process. Having low confidence in probed bits may still result in full key recovery on some schemes, such as on RSA with specialized algorithms [31]. Since other targets do not have specialized key recovery algorithms from noisy bits, we need to increase the confidence in the recovered bits. For this reason, we extend the offline profiling phase.

We profile the **reliable** and **unstable** pages while an attacker's copy of the wolfSSL TLS server is located at them. The pages are loaded with an ECDSA key in which the probe offset is set to 0, and they are profiled as explained in Section 4.3.3. We do not need to connect to the server, differently from the online attack phase. After profiling with a 0 value on the probe offset, we restart the server on the same pages and load them with an ECDSA key in which the probe offset is set to 1. Then, we profile them to observe the bit flips on the probe offset. If the probed bit offset is still flippy, we use this page in the online attack phase.

Table 5: The effect of wolfSSL TLS server on the flip rates. If the probe offset is still flippy, the page is suitable.

Physical Addr.	Offline Profiling	ECDSA Key Bit	# of Bit Flips	Suitable?
2bf774000	$0 \rightarrow 1$: 17 $1 \rightarrow 0$: 0	0 1	$0 \rightarrow 1$: 6 $1 \rightarrow 0$: 0	✓
1d0220000	$0 \rightarrow 1$: 12 $1 \rightarrow 0$: 0	0 1	$0 \rightarrow 1$: 0 $1 \rightarrow 0$: 0	✗

Table 5 shows two pages to probe the same bit on the ECDSA key. The page at `2bf774000` has 17 instances of $0 \rightarrow 1$ bit flips at the probe offset during profiling without the wolfSSL TLS server. We set the ECDSA key bit to 0 by loading the proper ECDSA key, and we profile the page. We observe 6 instances of $0 \rightarrow 1$ bit flips, which is compatible with the profiling without the wolfSSL TLS server. Then, the ECDSA key bit is set to 1, and we profile the page again. We do not observe any bit flips this time, as expected. Since the probed bit offset is still flippy, this page is suitable for the online attack phase.

The page at `1d0220000` has 12 instances of $0 \rightarrow 1$ bit flips on the probe offset during profiling without the wolfSSL TLS server. We repeat the same steps as the previous page. Since the probed bit offset is not flippy, this page is not suitable for the online attack phase. All **reliable** and **unstable** pages are profiled in this manner to find the suitable pages needed to perform the online attack.

6.2 Online Attack Phase

Allocating the Useful Pages. Once we identify pages that are useful to our attack, the next step involves allocating these pages for use in our online attack. For experimental purposes, during the profiling stage, we record the physical addresses of the victim pages and their corresponding attacker pages. To reallocate these pages, we utilize the `mmap` system call,

allocating various segments of memory. We then verify the physical addresses of these allocated memory pages. Pages with matching memory addresses are retained for use in the attack, while those that don't match the required physical address values are released.

It is important to note that in a full-scale, end-to-end attack, this memory allocation step would not be necessary. Should an appropriate page be identified during the profiling stage, the next action would be to immediately proceed with the online phase of the attack.

Mapping Private Key to Victim Page. To successfully flip bits in the ECDSA key, it is necessary to modify the memory arrangement, ensuring the key is positioned in one of the vulnerable rows identified previously. This is achieved by initially unmapping the areas prone to flips, after which the client either generates or loads the private key. Leveraging the characteristics of the Linux Buddy Allocator [4], which allocates newly freed physical pages from the page frame cache in a first-in-last-out manner, the private key gets mapped onto the vulnerable row.

Once the server initiates the TLS 1.3 handshake process, the ECDSA key is loaded from the key certificate to the server memory. In our attack scenario, we impersonate a client to establish a connection with the server, which is in a state of awaiting connection requests. A crucial aspect of the attack is the timing: it is imperative not to begin hammering the victim page until the key has been allocated to it. To address this timing or synchronization challenge, the client (in this case, the attacker) should delay the connection request, allowing sufficient time for the key to be generated and placed on the victim page. As an attacker, we have control over the timing of the connection request, enabling us to synchronize our actions with the server's key generation process.

Once we achieve the necessary synchronization, we proceed to hammer the attacker pages that were allocated prior to initiating the attack. Subsequently, we monitor the status of the TLS 1.3 handshake to determine whether a fault has occurred in the key. It is important to note, however, that the connection status does not always provide direct insight into the state of the secret bit. There are two scenarios to consider:

- First, the secret key might not be located on the victim page we targeted. In such instances, our bit flipping may impact other variables within the wolfSSL framework rather than the intended key.
- Second, there's a possibility that the bits being flipped belong to memory allocations of other processes, not just wolfSSL. In such cases, a failure in the connection does not necessarily imply a compromise of the secret bit.

This underscores the complexity of the attack and the need for careful analysis of the results to accurately interpret the impact of our actions.

ASLR Effect on the Key Allocation. Address Space Layout Randomization (ASLR) randomizes the victim memory space as a caution against buffer overflow attacks. If it is disabled by the system, we observe that the physical address of the page for the server key stays the same. This would increase the flips observed on the victim's system since we always target the same victim row. If ASLR is enabled, our chance of placing the server key into the victim page will decrease. However, it will only increase the time of our attack since it requires more iterations to eliminate the noise.

Shifting the Page Offset of Key. Our attack depends on the flippy bit offset on a page found in the offline memory profiling phase. Thus, it is vital to match the flippy bit offset with the bit offset of the server key. If any mismatch occurs, we would target a bit that is different than intended or belongs to the other variables in the process. We observe that the page offset of the server key always stays the same since ASLR does not randomize the page offset. In this case, we would limit our attack to the profiled pages that have bit-flips only on a constant 256 bit-space over a 4KB page. To show the viable attack scenario, we utilize an input-dependent `malloc` size on a variable from the victim's side. This changes the page offset of the server key and enables us to place it into the desired offset on the victim page.

In case where there is no attacker-controlled `malloc` before the secret, it is still possible to extend the offline memory profiling phase to find flippy pages for each bit offset. This would only prolong the offline attack time.

Listing 1: An example of dynamic memory allocation for server TLS 1.3 application from wolfSSL 5.6.3

```
1  int main(int argc, char** argv)
2  {
3    int ret = 0;
4  #ifdef WOLFSSL_TLS13
5    struct sockaddr_in servAddr;
6    struct sockaddr_in clientAddr;
7    socklen_t size = sizeof(clientAddr);
8    char buff[256];
9    size_t len;
10   int size = atoi(argv[2]);
11   char *reply = (char*) malloc(sizeof(char) *size);
12   int on;
13   /* declare wolfSSL objects */
14   WOLFSSL_CTX* ctx = NULL;
15   WOLFSSL* ssl = NULL;
16   ...
17   return ret;
18 }
```

On code snippet List. 1, `reply` variable is allocated by using `malloc`. This would affect the bit location of the ECDSA key, which is stored in `ctx` variable. As the size of `malloc` increases by changing **size** variable, the location of the ECDSA key within the DRAM page shifts toward the next DRAM page. We illustrate in Fig. 7 that an attacker who is in control of the size of a `malloc` can indeed shift the secret variable around the page boundaries, spanning all possible page offsets. This would allow us to utilize every **reliable** and **unstable**

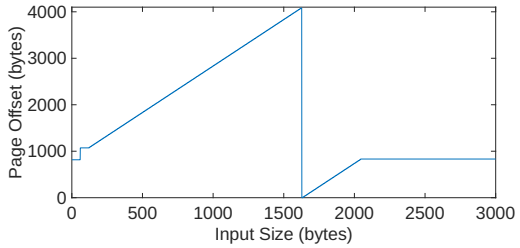


Figure 7: Shifting page offset of the secret using the attacker-controlled input size

Table 6: Summary of online attack outcomes on a fixed ECDSA key, showing the relationship between offline profiling, connection failures, and bit value predictions versus actual values.

Pages	Physical Address	Offline Profiling	Number of Connection Failures	Probed Bit Value	Real Bit Value
Reliable	2bf774000	$0 \rightarrow 1 = 17$	6	0	0
			0	1	1
	1cc3fc000	$1 \rightarrow 0 = 14$	0	0	0
			6	1	1
	2c01db000	$1 \rightarrow 0 = 13$	0	0	0
			6	1	1
Unstable	1b1cc5000	$0 \rightarrow 1 = 20$	4	0	0
			0	1	1
	2c06ad000	$0 \rightarrow 1 = 10$	8	0	0
			0	1	1

pages found in Section 6.1.

The `malloc` system call allocated memory that is 128-bit aligned, it allows us to shift `ctx` variable by 128-bit on the same page. This gives us the opportunity to probe congruent bits in the ECDSA key i.e. 0^{th} - 128^{th}, 1^{st} - 129^{th} bits, and so on. Hence, we only need 128 pages to probe the entire 256-bit ECDSA key.

Performing the Online Attack. Each page used for the online attack is hammered with the attacker pages set to 1s, and any connection failure is observed by connecting to the server as a client. This cycle is repeated 200 times. Then, the page is hammered with 0s to see if any connection failure happens. We also repeat this cycle 200 times. Then, we shift the ECDSA key 128-bit and repeat the attack cycle to probe the congruent bit on the key. In total, each page is used 800 times for the online attack to probe two bits on the ECDSA key.

6.3 Online Attack Results

We perform the online attack as explained in Section 6.2. Table 6 shows the online attack results on several examples of **reliable** & **unstable** pages.

For example, the page located at `2bf774000` has 17 instances of 0→1 bit flips during the offline profiling. We perform the online attack, and we observe 6 connection failures and probe the bit value as 0. Then, we perform the online attack on the *shifted* ECDSA key and do not observe any connection failures. Thus, we probe the bit value as 1 since we expect to see instances of 0→1 bit flips that hinder the connection.

The page located at `1b1cc5000` has 20 instances of 0→1 bit flips during the offline profiling. We perform the online attack and observe 4 connection failures. Thus, we probe the bit value as 1 since we expect to see 0→1 bit flips that prevent the connection. On the *shifted* ECDSA key, we do not observe any connection failures and probe the bit value as 0. After the attacks, we check the real bit values and validate our probing results.

Overall, the attack is performed on a 256-bit ECDSA key, and we achieve an average recovery rate of 22 bits/hour with a 100% success rate.

7 Comparison to Related Work

Rowhammer-based KEM Attacks. The works in [13] and [36] both target PQ KEM schemes using Rowhammer fault injection. The former uses Rowhammer to increase the error rate during encryption in FrodoKEM, which results in higher decryption failures, which are then mathematically analyzed to deduce key bits. Similarly, [36] achieves key recovery from LWE-based KEM schemes using Rowhammer by running the decapsulation procedure or the plaintext checking oracle multiple times with different ciphertexts while introducing faults that reveal parts of the key. Both techniques require extensive analysis. In contrast, FAULT+PROBE can directly fault and probe bits without any extensive analysis given an observation channel.

Signature Correction Attacks. SCAs such as LUOV [38], Dilithium [22] and EC/DSA [37], on the other hand, recover key bits by correcting faulty signatures obtained by Rowhammer fault injection. Our attack does not require access to faulty signatures/ciphertext or any complex mathematical postprocessing to recover the secret key bit. For instance, for signature correction to work, the faulty bit needs to be mathematically *traceable* to the faulty signature, allowing signature correction. Instead, the presented FAULT+PROBE attack can directly recover bit values either by observing signing rates (if fault checking is implemented) or by recovering error-dependent change in the behavior of the protocol.

RAMBleed. Kwong *et al.* [31] proposed RAMBleed to extract bits using Rowhammer. Our attack is fundamentally different in terms of how the secret bit is deduced. In RAMBleed, the attacker exploits the data dependency of the aggressor rows. FAULT+PROBE exploits the observable effect of a bit flip on the secret value. Moreover, RAMBleed has three fundamental limitations that FAULT+PROBE does not have.

First, in RAMBleed, the attacker has to co-locate two copies of the victim in the same rows and force them to

hammer the attacker's row. In our attack, however, the victim can exist in any row as long as the attacker rows cause a bit-flip on the victim row.

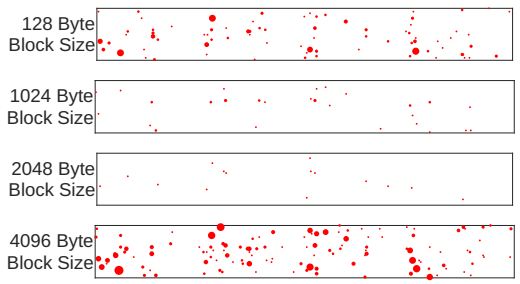


Figure 8: Flip characteristics of a single page with different attacker row data patterns. Larger dots indicate more repeatable bit flips in a certain data pattern in DDR4.

Second, RAMBleed can only deduce the bits with a certain probability. Probabilistic information is enough to extract a full RSA key using a variant of Henninger-Shacham algorithm [18, 40], yet it does not work on other schemes such as ECDSA. Since FAULT+PROBE can deduce bits with 100% accuracy, it can be used as a generic attack on any secret that can be flipped using Rowhammer and reveals fault information.

Finally, single-sided or double-sided RAMBleed would not cause bit flips on DDR4 systems because of the TRR mechanism. In our systems, the best configuration requires 15 attacker rows. Requiring 15 copies of the secret around the sampling row successfully is much less realistic. More importantly, the RAMBleed attack was based on the assumption of data dependency of bit flips on DDR3 memory chips. It assumes that a bit flip strongly depends on the bit value of the same offset in the immediate neighbors. However, in our experiments on DDR4, we have observed that this is not the case. To show the effect of diagonal memory cells on a flippy page offset, we tried different aggressor row patterns. We set the value of the hammering rows as 111..1000..0111..1 where we call running 1s or 0s a *block*. Fig. 8 demonstrates how a single page shows different flip characteristics when we change the size of the block. We observed that different values on the attacker rows change these flip characteristics of the victim row entirely. Therefore, without knowing the actual value of the whole victim row, it would not be possible to comment on the flips seen on the attacker-controlled "sample page". Knowing every bit of the secret value prior to the attack would be an unrealistic assumption. In conclusion, a profiling-based attack like RAMBleed would not work on DDR4 systems.

8 Countermeasures

In this section, we discuss the viability of several countermeasures against FAULT+PROBE that can be implemented on both software and hardware.

Mitigating Faults. In FAULT+PROBE, we rely on bit corruption using Rowhammer. Therefore, any defense mechanism that prevents targeted Rowhammer attacks also prevents FAULT+PROBE.

Reducing the DRAM refresh interval constrains the attack's potential by limiting the available time for inducing bit flips. However, this measure leads to increased power consumption and decreased system performance since memory cannot be accessed during the refresh operation, thereby making it an impractical solution for widespread adoption.

Researchers have proposed alternative row refresh techniques that aim to balance effectiveness against performance and energy penalties. One notable example is the Hidden Row Activation (HiRA) [2], which introduces a method for parallelizing row refreshes. HiRA capitalizes on the architectural feature that different rows within the same bank can be connected to distinct charge restoration circuits, enabling simultaneous refresh operations. This method not only diminishes the window available for Rowhammer attacks by embedding refresh operations into the normal row access cycles but also mitigates the associated latency issues. Although HiRA offers a promising approach to mitigating Rowhammer attacks, its integration into actual products is still forthcoming.

Further advancements are represented by the work of Wang *et al.* [52], which advances the Probabilistic Adjacent Row Activation (PARA) [27] defense mechanism through the development of Discreet-PARA. This approach integrates disturbance tracking with a dedicated cache for recently accessed rows (PARA-cache), facilitating precise management of refresh operations for rows at risk of Rowhammer attacks. By refining the process of monitoring row accesses and refreshes, Discreet-PARA has successfully reduced the performance overhead typically associated with such mitigation techniques. Since PARA requires changes in the memory controllers or DRAM chips, it is not possible to implement them on the current systems [39].

Although Error Correcting Code (ECC) can correct single-bit flips, Rowhammer attacks capable of generating multiple-bit flips can circumvent ECC protection, as demonstrated in [10]. This limitation underscores the necessity for Rowhammer-specific countermeasures even in systems equipped with ECC.

Mitigating Probes. For extracting the secret bit value, we rely on a feedback mechanism that is observable by the adversary. Eliminating or blinding this channel would mitigate FAULT+PROBE as well.

In server/client scenarios where a malicious client collects faulty signatures, verifying the signature before sending it

to the client would prevent the client from deducing secret bits *only by using the signature verification.* Yet, as we explained in the previous sections, this alone is not a sufficient mitigation. Even if the server implements verify-after-sign as a security measure, how it handles an unverified signature scenario is also important for FAULT+PROBE. Libraries need to prevent error codes reveal information to the clients if there is a fault in the secret values. This requires a thorough review of the error codes.

Finally, not releasing any error code to the client and correcting the fault on the fly can be the last resort. However, an error handling mechanism needs to be implemented to be constant-time (or, more precisely, fault-independent). For example, repeating the signing operation when the verify-after-sign fails can potentially cause a visible delay in the connection, and the malicious client can deduce that the fault went through successfully. To prevent that, two copies of the signatures can be generated at the same time, and whichever is verified can be sent to the client. This would cause computational overhead, but if implemented constant-time, the client would not see the effect of the fault.

9 Exploiting ASLR as an Attack Vector

When the number of reliable pages that can generate reproducible bit flips with low noise is limited, we need to rely on shifting the secret value within a page so that we can reuse the same pages to probe multiple bits. In Section 6.2, we explained that this can be done using an attacker-controlled `malloc` size.

Previous works [1, 47] demonstrated flipping variables stored in stack memory. In case the secret value is stored in stack memory instead of heap memory, changing the `malloc` size will not affect the page offset of the secret. Yet, ASLR implemented in the Linux kernel randomizes page offset by default. Here, we propose to exploit address randomization that is implemented to improve performance by *"avoiding L1 evictions by the processes running on the same package"* [35, 49] to enable FAULT+PROBE against the secret variables that are stored in stack memory. This is critical when the number of reliable pages is not enough to cover the secret size.

Fig. 9 shows an overview of the ASLR shift idea. Note that regardless of the size of the variable, the last 4 bits of the address are not affected by the offset randomization. Yet, the remaining 8 bits of the page offset are randomized. Let's say a variable has a page offset `addr` in one run and `addr+0x10` in the second run. Since the last 4 bits of the address are not affected by the randomization, if the size of the variable is less than 16 bytes, all bits in the variable will always be mapped into unique locations on the physical page. Yet, if the variable is larger than 16 bytes, `addr` and `addr+0x10` will have overlapping bits for different runs.

Assuming that an unprivileged adversary has no prior information about the randomized page offset of a variable in a given run, using our FAULT+PROBE, they can deduce where this flip occurred based on the unique mapping of every bit for variables with a size smaller than 16 bytes. Similar to the technique explained in Section 6.2, since the flip characteristic of the profiled page is known, an error in the program will indicate the original value of the corrupted bit.

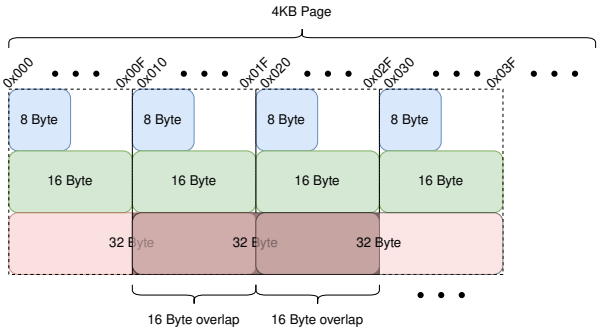


Figure 9: ASLR Shift overview

For variables with a size larger than 16 bytes, the value of the secret bit can be found the same way. However, the overlapping parts decrease the probability of deducing the location of the bit within the variable. For example, for a 32-byte variable, a detected bit flip at offset `0x020` may be located either at the beginning of the first or second half of the variable, resulting in two different options.

To generalize, in a variable with size n bytes, an adversary can tell the location of each corrupted bit on the variable with a $16/n$ probability where $n \geq 16$ and $n \equiv 0 \pmod{16}$.

10 Conclusion

In this study, we presented FAULT+PROBE, a novel methodology that uses the Rowhammer to deduce secret information exploiting the observable changes in a system's behavior following fault injections. Through rigorous experimentation, including a targeted attack on the ECDSA signature scheme in wolfSSL TLS handshake, we have demonstrated the capability of FAULT+PROBE to bypass established fault-check mechanisms.

The introduction of FAULT+PROBE underlines the necessity for a refined defensive strategy against Rowhammer and similar hardware-level threats. Future countermeasures must address the attack vectors that exploit operational behaviors as side channels.

Acknowledgment

This work was supported by the National Science Foundation grant CNS-2026913 and in part by a grant from the Qatar National Research Fund.

References

[1] Andrew J. Adiletta, M. Caner Tol, Yarkın Doröz, and Berk Sunar. Mayhem: Targeted corruption of register and stack variables. In *Proceedings of the 2024 ACM Asia Conference on Computer and Communications Security*, 2023.

[2] Anonymous. Hira: Hidden row activation for reducing refresh latency of off-the-shelf dram chips, 2022.

[3] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices*, 51(4):743–755, 2016.

[4] Daniel P Bovet and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[5] 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)*, pages 117–130, Vancouver, BC, August 2017. USENIX Association.

[6] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant cpus. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 769–784, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Anirban Chakraborty, Sarani Bhattacharya, Sayandeep Saha, and Debdeep Mukhopadhyay. Explframe: exploiting page frame cache for fault analysis of block ciphers. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1303–1306. IEEE, 2020.

[8] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 49:1162–1174, 2016.

[9] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are we susceptible to rowhammer? an end-to-end methodology for cloud providers. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 712–728. IEEE, 2020.

[10] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ECC memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 55–71. IEEE, 2019.

[11] Jonathan Corbet. Defending against Rowhammer in the kernel, October 2016. https://lwn.net/Articles/704920/.

[12] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized many-sided rowhammer attacks from JavaScript. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1001–1018. USENIX Association, August 2021.

[13] Michael Fahr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, and Daniel Apon. When frodo flips: End-to-end key recovery on frodokem via rowhammer. CCS '22, page 979–993, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 747–762. IEEE, 2020.

[15] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261. IEEE, 2018.

[16] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A remote software-induced fault attack in javascript. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 300–321. Springer, 2016.

[17] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[18] Nadia Heninger and Hovav Shacham. Reconstructing rsa private keys from random key bits. In *Annual International Cryptology Conference*, pages 1–17. Springer, 2009.

[19] Nishad Herath and Anders Fogh. These are not your grand Daddys cpu performance counters–cpu hardware

performance counters for security. *Black Hat Briefings*, 2015.

[20] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. MASCAT: Stopping microarchitectural attacks before execution. *IACR Cryptol. ePrint Arch.*, 2016:1196, 2016.

[21] Saad Islam, Ahmad Moghimi, Ida Bruhns, Moritz Krebbel, Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. SPOILER: Speculative load hazards boost rowhammer and cache attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 621–637, Santa Clara, CA, August 2019. USENIX Association.

[22] Saad Islam, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. Signature correction attack on dilithium signature scheme. *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 647–663, 2022.

[23] Saad Islam, Koksal Mus, and Berk Sunar. Quantumhammer: A practical hybrid attack on the LUOV signature scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1071–1084, 2020.

[24] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, volume 1, 2022.

[25] 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)*, pages 1702–1718. IEEE, 2023.

[26] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. Sledgehammer: Amplifying rowhammer via bank-level parallelism. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[27] 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. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[28] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[29] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-double: Hammering from the next row over. In *31st USENIX Security Symposium: USENIX Security'22*, 2022.

[30] Elisabeth Krahmer, Peter Pessl, Georg Land, and Tim Güneysu. Correction fault attacks on randomized crystals-dilithium. Cryptology ePrint Archive, Paper 2024/138, 2024. https://eprint.iacr.org/2024/138.

[31] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading bits in memory without accessing them. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 695–711. IEEE, 2020.

[32] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[33] 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 (EuroS&PW)*, pages 710–719. IEEE, 2020.

[34] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. Rega: Scalable rowhammer mitigation with refresh-generating activations. In *44rd IEEE Symposium on Security and Privacy (SP 2023)*. IEEE, 2023.

[35] Ingo Molnar. Enhancing the linux kernel. https://lkml.org/lkml/2006/7/25/62, Jul 2006. Accessed: 2024-02-07.

[36] Puja Mondal, Suparna Kundu, Sarani Bhattacharya, Angshuman Karmakar, and Ingrid Verbauwhede. A practical key-recovery attack on lwe-based key-encapsulation mechanism schemes using rowhammer. *22nd International Conference on Applied Cryptography and Network Security (ACNS)*, 2024.

[37] Koksal Mus, Yarkın Doröz, M. Caner Tol, Kristi Rahman, and Berk Sunar. Jolt: Recovering tls signing keys via rowhammer faults. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1719–1736, 2023.

[38] Koksal Mus, Saad Islam, and Berk Sunar. Quantumhammer: A practical hybrid attack on the luov signature scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*,

CCS '20, page 1071–1084, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Onur Mutlu. The rowhammer problem and other issues we may face as memory becomes denser. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1116–1121. IEEE, 2017.

[40] Kenneth G Paterson, Antigoni Polychroniadou, and Dale L Sibborn. A coding-theoretic approach to recovering noisy rsa keys. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*, pages 386–403. Springer, 2012.

[41] Mathias Payer. HexPADS: a platform to detect "stealth" attacks. In *International Symposium on Engineering Secure Software and Systems*, pages 138–154. Springer, 2016.

[42] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, Austin, TX, August 2016. USENIX Association.

[43] Islam Saad, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. A signature correction attack on the post-quantum scheme dilithium. In *Proceedings of the IEEE European Workshop on Security and & Privacy*, 2022.

[44] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. *Black Hat*, 15:71, 2015.

[45] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: A surgical precision hammer. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 47–66, Cham, 2018. Springer International Publishing.

[46] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 213–226, Boston, MA, July 2018. USENIX Association.

[47] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. Spechammer: Combining spectre and rowhammer for new speculative attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 681–698. IEEE, 2022.

[48] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *41th IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[49] Arjan van de Ven. Patch 4/6 randomize the stack pointer. *LWN.net*, Jan 2005. Accessed: 2024-02-07.

[50] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689, 2016.

[51] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *S&P*, May 2019.

[52] Z. Wang, W. Liu, and Y. Wang. Discreet-para: Rowhammer defense with low cost and high efficiency. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 1–8. IEEE, 2021.

[53] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):169–195, Jun. 2020.

[54] wolfSSL Inc. wolfssl (5.5.0), Aug 2022.

[55] 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)*, pages 19–35, Austin, TX, August 2016. USENIX Association.

[56] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.