Battering RAM: Low-Cost Interposer Attacks on Confidential Computing via Dynamic Memory Aliasing

Jesse De Meulemeester
jesse.demeulemeester@esat.kuleuven.be
COSIC, KU Leuven
Leuven, Belgium

David Oswald
david.f.oswald@durham.ac.uk
University of Birmingham & Durham University
United Kingdom

Ingrid Verbauwhede
ingrid.verbauwhede@esat.kuleuven.be
COSIC, KU Leuven
Leuven, Belgium

Jo Van Bulck
jo.vanbulck@cs.kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

***Abstract*—Confidential computing, powered by trusted execution environments (TEEs) like Intel SGX/TDX and AMD SEV-SNP, is now widely available from major cloud providers. At the core of these technologies is hardware-level memory encryption to protect against privileged attackers and physical threats such as bus snooping and cold boot attacks. Recent extensions add access-control checks to defend against software-based ciphertext manipulation and aliasing attacks. In this work, we challenge the protection modern memory encryption technologies offer against physical adversaries by building a low-cost (<$50) DDR4 interposer that dynamically tampers with address lines to bypass aliasing checks in current TEEs.**

We demonstrate how the runtime nature of our interposer bypasses boot-time firmware mitigations introduced by AMD and Intel in response to software-based memory aliasing attacks. Using our interposer, we present the first attack on Scalable SGX's single-key domain, achieving arbitrary plaintext read/write access and extracting SGX's platform provisioning key, thereby dismantling trust in remote attestation. We further re-enable a full attestation breach on up-to-date AMD SEV-SNP platforms, bypassing recent firmware defenses against static aliases. Our results challenge core assumptions about encrypted memory security and highlight critical shortcomings in the performance-security trade-offs of current confidential computing systems. Costing orders of magnitude less than commercial DRAM interposers, our device underscores the need for stronger protections against low-cost physical attacks in scalable TEE designs.

1. Introduction

The increasing reliance on cloud infrastructure has intensified concerns over data confidentiality, especially in the face of insider threats, privileged software vulnerabilities, and coercive legal regimes. These risks, highlighted by the Snowden revelations and reinforced by evolving privacy regulations, call for stronger security guarantees. Confidential computing addresses this need by enabling the processing of sensitive data within hardware-isolated Trusted Execution Environments (TEEs) where both code and data are protected from the surrounding system, including the privileged operating system and hypervisor. With the advent of commercial technologies like Intel's Software Guard Extensions (SGX) and Trust Domain Extensions (TDX), AMD's Secure Encrypted Virtualization (SEV), and Arm's Confidential Compute Architecture (CCA), confidential computing is now widely supported by major cloud providers.

A key building block of confidential computing is efficient and scalable hardware-level memory encryption. Unlike alternative, less practical data-in-use technologies like homomorphic encryption, confidential computing allows computations directly on plaintext within the CPU package. Strict hardware-level access control mechanisms ensure that plaintext data in CPU caches or registers cannot be accessed from outside the protected *enclave* environment. Meanwhile, a dedicated memory encryption engine, integrated into the CPU's memory controller, transparently encrypts and decrypts all data moving between the CPU and untrusted Dynamic Random-Access Memory (DRAM). This mechanism provides critical protection against untrusted cloud providers who may attempt physical attacks, such as cold-boot attacks [1] or memory bus snooping [2].

Given its central role in confidential computing, security analysis of encrypted memory has received considerable attention in recent years. Most research has centered on *software-level* threats, uncovering vulnerabilities such as ciphertext side channels [3, 4] and memory aliasing attacks [5]. In response, hardware vendors have deployed firmware-level mitigations, such as Intel's `MCHECK` and Alias Checking Trusted Module (ACTM) [6] and AMD's `ALIAS_CHECK` [7] and `CiphertextHiding` [8] features to eliminate the software attack surface.

In contrast, the resilience of modern memory encryp-

tion schemes against *physical* adversaries—those with direct hardware access—remains less well understood. Early designs, such as Intel's Client SGX [9], offered strong integrity and freshness guarantees against physical adversaries, but were limited to small protected memory regions (typically 128–256 MB). To enable lift-and-shift deployment of unmodified applications, newer architectures like Intel Scalable SGX and TDX, AMD SEV, and Arm CCA all have scaled encryption to cover the full DRAM. However, this scalability comes at the cost of relaxed cryptographic integrity protections, increasing potential exposure to physical attacks. These risks have traditionally been considered impractical, as successful exploitation was assumed to require high-cost equipment such as active interposers priced upwards of $100,000. Reflecting this common belief, AMD describes physical attacks as "very complex and requir[ing] a significant level of local access and resources" [10], and Intel characterizes the shift in Scalable SGX as a "performance vs. security trade-off" while asserting that "critical benefits are still maintained" [6].

In this paper, we challenge these assumptions, considering the following fundamental research questions:

What level of protection do today's memory encryption technologies offer against physical adversaries? What are the technical and economic requirements for mounting a successful attack? What are the practical implications of breaking these protections for different TEEs?

To address these questions, we develop a minimal, low-cost memory interposer—built from off-the-shelf components for under $50—that sits between the CPU and DDR4 memory and dynamically reroutes address lines via a programmable switch. Using this setup, we demonstrate for the first time the existence of *dynamic memory aliases*, which can be introduced at runtime. This crucial property allows us to completely bypass Intel's `MCHECK`/ACTM and AMD's `ALIAS_CHECK` firmware mitigations for prior, software-based memory-aliasing BadRAM attacks [5]. In contrast to earlier off-chip TEE attacks such as Membuster [2], which used commercial DDR4 interposers exceeding $100,000 for *passive* side-channel observations, our custom interposer enables much stronger, *active* manipulation at a fraction of the cost. Moreover, our attack is highly practical and accessible, unlike invasive fault attacks like VoltPillager [11], which physically interposes on the CPU's voltage regulation bus to trigger unstable sporadic faults. Our method avoids unpredictable microarchitectural behavior and instead leverages stable, completely deterministic address aliasing effects, posing a significant and previously underestimated threat to modern memory encryption schemes.

Building on our minimal *hardware* interposer, we develop more complex exploitation primitives in *software*. We first examine Intel Scalable SGX, exposing for the first time the practical implications of the weakened memory encryption integrity guarantees introduced several years ago with little public scrutiny or acknowledgment. Specifically, we show that our interposer trivially enables ciphertext replay, capture, and tampering. Moreover, we present an innovative attack technique that exploits Scalable SGX's use of a single memory encryption key shared across all enclaves, combined with the privileged adversary's control over physical memory allocation, to enable arbitrary plaintext reads and writes for arbitrary victim enclaves via an attacker-controlled helper enclave. Armed with this powerful capability, we extract the SGX platform provisioning key, fundamentally undermining the trust model of SGX remote attestation. We then turn to AMD SEV-SNP and successfully reproduce prior BadRAM-style memory aliasing attacks on fully updated platforms, even with AMD's latest `ALIAS_CHECK` firmware mitigation in place, demonstrating full compromise of remote attestation and effectively dismantling trust in the SEV ecosystem.

Our findings provide valuable insights for the design of current and future TEEs, highlighting the fundamental security-performance trade-off inherent in confidential computing for large enclaves—an issue that, until now, has received limited practical security attention and remained mostly theoretical. More broadly, this calls for a nuanced reassessment of the trust placed in confidential computing compared to alternatives like homomorphic encryption, which, though less practical, offer stronger theoretical guarantees for outsourced computation.

Attacker Model and Scope. As per our research questions, we explicitly set out to investigate the level of protection that modern TEEs offer against adversaries with (temporary) *physical* access to the motherboard. While we acknowledge that vendors such as Intel and AMD currently consider "invasive" physical attacks to be out of scope, physical attacks on TEEs remain an active and evolving area of academic and practical research [2, 5, 11, 12, 13, 14, 15, 16]. More importantly, in the context of confidential computing in the cloud, the threat of physical access cannot be dismissed: adversaries may include untrusted cloud operators, rogue personnel, supply-chain providers, or even local law enforcement with temporary access to server hardware. Indeed, the very motivation behind hardware-level memory encryption is to protect sensitive data against such scenarios—particularly against attacks that exploit DRAM access, such as cold boot or memory bus snooping.

In this paper, we focus on widely deployed DDR4 memory systems, which are compatible with modern TEEs such as Intel Scalable SGX and AMD SEV-SNP. Due to the technical complexities introduced in DDR5, in particular multi-cycle commands, our interposer is not currently compatible with DDR5 platforms. As a result, our attack does not apply to Intel TDX, which is only available on DDR5-capable systems. We further elaborate on the threat model and system assumptions in Section 3.1, and reflect on limitations and broader implications in Section 8.

Contributions. In summary, our main contributions are:

- We construct a low-cost hardware interposer to dynamically introduce aliases in DDR4 memory.
- We present the first breach of Scalable SGX, achieving arbitrary plaintext access to enclave memory,

highlighting the limitations of current aliasing mitigations and weakened memory encryption schemes.
- We implement an end-to-end attack that compromises Scalable SGX's remote attestation.
- We bypass AMD SEV-SNP's firmware mitigations to revive BadRAM-style attacks.
- We analyze the broader implications of interposer attacks for other TEEs and DDR5.

Responsible Disclosure. We disclosed our novel DDR4 interposer attack technique, including proof-of-concept exploits that enable reading and writing arbitrary plaintext on up-to-date Scalable SGX platforms, to Intel's Product Security Incident Response Team (PSIRT) on January 29, 2025. Intel acknowledged our findings, which they shared with selected customers, but noted that physical attacks are currently considered out of scope for Intel Scalable SGX and Intel TDX. To better reflect this position, Intel deposited the whitepaper on Scalable SGX, previously removed from the Intel website [6], permanently on arXiv [17]. Intel requested an embargo until September 30, 2025, followed by a public security announcement.

On February 10, 2025, we also disclosed our attacks to AMD's PSIRT, including proof-of-concept exploits that bypass the newly introduced memory-aliasing mitigations on up-to-date SEV-SNP platforms. AMD similarly clarified that physical attacks are currently considered out of scope for AMD SEV-SNP. AMD similarly requested an embargo to prepare a security brief (AMD-SB-3024).

On June 16, 2025, we reached out to Arm PSIRT regarding the potential applicability of our attack to upcoming Arm CCA processors that are not yet publicly available. Arm responded that physical attacks are beyond the security guarantees provided by their intellectual property, including CCA, but shared our findings with their licensees.

Open Science. The hardware design for our custom interposer and corresponding microcontroller firmware, along with the attack scenarios described in this paper, are available at https://github.com/batteringramattack/batteringram.

2. Background and Related Work

2.1. Trusted Execution Environments

Trusted Execution Environments (TEEs) are hardware-enforced execution environments that ensure isolation of data and computations from the rest of the system, including privileged software such as the operating system or hypervisor. They are relevant in cloud scenarios where the provider or infrastructure may not be fully trusted. TEEs aim to provide confidentiality and integrity against malicious co-tenants, compromised hosts, and limited physical attackers.

One of the first commercial TEEs, Intel's Client Software Guard Extensions (SGX), enabled isolated memory regions, or enclaves, within applications on commodity Intel processors [18]. Intel later extended SGX to servers with Scalable SGX [6]. Intel's latest TEE, Trust Domain Extensions (TDX) [19], protects entire Virtual Machines (VMs), also referred to as Confidential VMs. AMD, similarly, provides Secure Encrypted Virtualization (SEV) [20]. Subsequent versions introduced confidentiality for the register file with SEV Encrypted State (ES) [21], and memory layout integrity with SEV Secure Nested Paging (SNP) [10]. Finally, Arm recently announced Confidential Compute Architecture (CCA), protecting VMs on Arm processors [22]; however, no hardware supporting CCA is currently available.

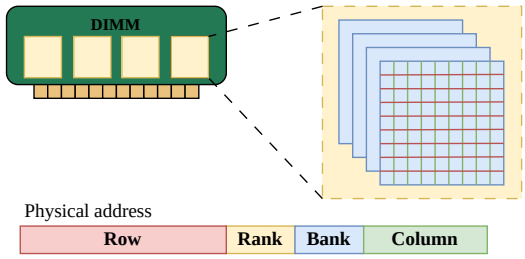


Figure 1. Simplified visualization of DRAM addressing.

Memory Encryption. All current TEEs implement memory encryption to protect data in external memory. Client SGX, one of the early TEEs, used strong cryptographic primitives to provide confidentiality, integrity, and freshness [9]. However, through these constructs, it was limited to protecting only up to 128 or 256 MB. To avoid this bottleneck, newer "scalable" TEEs, including Scalable SGX, TDX, SEV, and CCA, trade off some of these guarantees to protect the entire memory range. They employ a deterministic encryption scheme with an address-dependent tweak to ensure scalability while still maintaining confidentiality. Scalable SGX, for instance, builds upon Total Memory Encryption (TME), which uses a single key to encrypt the entire memory range using AES-XTS [23]. Similarly, TDX relies on Total Memory Encryption-Multi-Key (TME-MK), with a dedicated key for each Trust Domain (TD) [24]. SEV encrypts data using AES-XEX with a separate key per guest [20]. Finally, CCA employs either AES-XEX or QARMA.

2.2. DRAM Addressing

Dynamic Random-Access Memory (DRAM) stores data as electrical charge in capacitors arranged into a two-dimensional grid, called a bank. To access data, the corresponding row must first be activated by copying it into the row buffer. Once activated, read and write instructions can be issued by specifying the column within that row. Only one row per bank can be active at any given time.

In DDR4, each DRAM chip typically consists of 16 banks organized into four bank groups. Optionally, high-density chips may use 3-Dimensional Stacking (3DS), where multiple dies are integrated into a single package. Several DRAM chips operate in parallel to form a rank. Typically, DDR4 Unbuffered Dual In-line Memory Modules (UDIMMs) contain 8 or 16 DRAM chips per rank, while

	$\overline{CS}$	$\overline{ACT}$	BG	BA	CID	A17	A16	A15	A14	A13	A12	A11	A10	A9-0
ACT	L	L	Bank		CID	Row								
MRS	L	H	Reg		V	RFU	L	L	L	OP				
REF	L	H	V	V	CID	V	L	L	H	V	V	V	V	V
PRE	L	H	Bank		CID	V	L	H	L	V	V	V	L	V
PREA	L	H	V	V	CID	V	L	H	L	V	V	V	H	V
RFU	L	H	RFU				L	H	H	RFU				
WR	L	H	Bank		CID	V	H	L	L	V	$\overline{BC}$	V	AP	CA
RD	L	H	Bank		CID	V	H	L	H	V	$\overline{BC}$	V	AP	CA
ZQCL	L	H	V	V	V	V	H	H	L	V	V	V	H	V
ZQCS	L	H	V	V	V	V	H	H	L	V	V	V	L	V
NOP	L	H	V	V	V	V	H	H	H	V	V	V	V	V
DES	H	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 2. DDR4 command encoding [27]. Values can be either high (H), low (L), valid (V), or don't care (X). BA, BG, and CA refer to bank group, bank address, and column address, respectively. Chip ID (CID) is only defined for 3DS devices, otherwise valid. Note that some columns represent multiple lines (*i.e.*, BG, BA, CID, and A9-0).

Registered Dual In-line Memory Modules (RDIMMs) contain one or two additional chips that store Error-Correcting Code (ECC) bits. DIMMs are organized in channels. Upon accessing a certain location, the memory controller translates the physical address into the corresponding DRAM address consisting of channel, rank, bank, column, row, and optionally Chip ID (CID) bits. Figure 1 provides an overview. In practice, physical address bits are not mapped one-to-one with DRAM addressing bits to improve performance [25, 26].

Communication between the memory controller and DRAM occurs over the Command/Adress (CA) bus. This bus conveys commands such as row activation (`ACT`), read (`RD`), write (`WR`), and refresh (`REF`). On DDR4, the CA bus consists of 26 lines, including 18 address lines (`A17-0`) [27]. These bits are used to encode both the memory addresses (e.g., row or column), command types, and additional options, depending on the specific command. The command encoding is standardized by JEDEC and shown in Figure 2.

RDIMMs, commonly used in server platforms, include a Registering Clock Driver (RCD) between the processor and DRAM chips [28]. The RCD buffers data and commands, reducing the electrical load on the bus. It also performs parity checking on the CA bus to detect transmission errors.

2.3. Off-Chip Physical-Access Attacks

Memory Bus Snooping. Lee et al. employed a commercial DDR4 interposer, costing approximately $170,000, to passively snoop the memory address bus [2]. Since addresses are transmitted unencrypted, their interposer enabled side-channel attacks by observing secret-dependent enclave memory access patterns. As Scalable SGX had not yet been released, their analysis was limited to passive address side channels targeting Client SGX.

Although expensive commercial interposers are capable of capturing DRAM traffic, they are, to our knowledge, limited to *passive* observation and would thus only support hardware-aided ciphertext side-channel analysis on Scalable SGX, similar to software-based ciphertext side-channel attacks on AMD SEV-SNP [3]. In contrast, Battering RAM demonstrates much more powerful, *active* manipulation of the DRAM bus with a low-cost setup, ultimately enabling arbitrary plaintext access to Scalable SGX enclave memory.

Memory Aliasing. De Meulemeester et al. manipulated the configuration data in a DIMM's Serial Presence Detect (SPD) flash chip to cause aliasing in the physical address space, circumventing CPU access control mechanisms [5]. They leveraged this "BadRAM" attack primitive, typically requiring one-time physical access, to mount attacks on AMD SEV-SNP and Client SGX. However, they found that newer Intel TEEs, including Scalable SGX and TDX, already mitigate such attacks via boot-time alias checks [6]. In response, AMD introduced a similar firmware mitigation [7]. Hence, it is believed that major commercial TEEs are no longer vulnerable to such *static* memory aliasing.

Focusing on hardware trojans beyond TEEs, Hopkins et al. constructed an FPGA-based interposer to redirect addresses on DDR3 memory [29]. Their FPGA-based interposer modifies DDR3 commands by downclocking the bus to 400 MHz and extending SPD latencies. DDR4, however, has a minimum clock frequency of 800 MHz, which poses much stricter timing constraints. While high-end FPGAs could theoretically meet these timing constraints, their cost and the engineering challenges, including timings, impedance matching, and signal integrity, would considerably complicate the design. In contrast, we achieve similar functionality with a simple and low-cost setup.

Fault Injection. Developing low-cost custom hardware, Chen et al. showed that by injecting malicious control packets into the bus between CPU and external voltage regulator, faults can be injected into enclaved computations in Client SGX [11]. These faults can then be leveraged to, e.g., recover cryptographic keys from SGX enclaves, depending on the nature of the target code. Buhren et al. adopted a similar approach to fully break AMD SEV, installing custom firmware on AMD's security co-processor [13].

Techniques that target the physical data, address, and control lines between a processor and its external memories have been explored in the past to bypass security features: a jailbreak of the Nintendo Wii gaming console is based on shorting pins on the memory bus to create memory aliasing [30]. To enter the (normally disabled) bootloader of embedded devices such as LTE routers, Dixon used needles to short memory data lines to cause failure during the boot process [31]. Beyond physically connecting to signals, other works have shown memory manipulation attacks with proximity: Cui use electromagnetic pulses to corrupt data in memory and bypass TrustZone-based secure boot on an Arm processor [12]. Along similar lines, Mishra et al. characterize and exploit electromagnetic faults on

the memory bus to attack TrustZone applications [16]. In contrast to our work, all of the aforementioned attacks rely on application-specific, probabilistic faults and are mostly limited to TrustZone.

To facilitate the analysis of Rowhammer effects, Cojocar et al. developed a low-cost DDR4 interposer to filter out refresh commands on the memory bus [32]. Gloor et al. recently proposed a similar platform for DDR5 memory [33]. However, these interposers only filter refresh commands and thus cannot be used to bypass TEE security.

3. Memory Interposer for Dynamic Aliasing

In this section, we present a minimal, low-cost DDR4 interposer that enables *dynamic* memory aliasing at runtime, bypassing existing boot-time alias checks and reactivating a potent class of memory-aliasing attacks.

3.1. Attacker and System Model

We assume an adversary with root privileges on the target system, consistent with the standard TEE threat model. Additionally, the attacker has (temporary) physical access to the device—sufficient to install an interposer. Our minimally invasive interposer design does not require permanent modifications and could be deployed, for example, in a brief supply chain compromise to extract the Scalable SGX attestation key (cf. Section 5).

At the system level, we focus on platforms with DDR4 memory, which remains widely deployed and is compatible with both Intel Scalable SGX and AMD SEV-SNP. We assume systems are fully up to date with the latest firmware and hardware mitigations, including boot-time alias checks such as Intel's `MCHECK` and ACTM [6], and AMD's `ALIAS_CHECK` [7].

3.2. Interfering with DRAM addressing

While data sent between the CPU and DRAM is encrypted, the commands are not, allowing a physical attacker to interfere with them. By inserting themselves between the CPU and DRAM, adversaries can, theoretically, arbitrarily modify the sent commands, modifying, dropping, or replaying them at will. In practice, however, building a full interposer capable of real-time command manipulation at DDR4 speeds is highly complex and poses significant engineering and cost challenges. As a result, such attacks are largely considered impractical. Instead, a more practical attack may consider only tampering with a single or limited number of lines. In this section, we investigate the possibility of creating dynamic aliases in the CPU's physical memory view by manipulating the CA lines on DDR4 DIMMs.

Command Encoding. To reduce the number of required data lines between the CPU and DIMM, the different lines of the CA bus may take on different semantic roles depending on the command being sent. Figure 2 on page 4 summarizes the DDR4 command encoding.

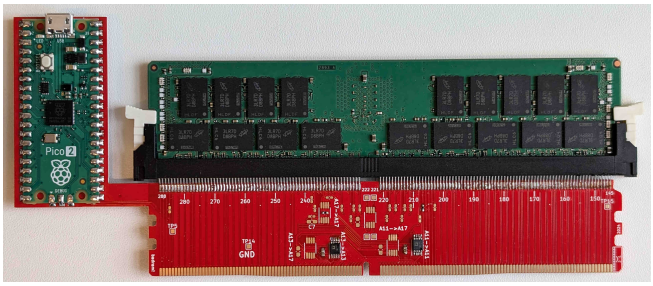


Figure 3. Our DDR4 interposer, containing two analog switches (bottom center) controlled by a microcontroller (left). The switches can dynamically either pass through the command signals to the connected DIMM or connect the respective lines to ground.

To introduce aliases, the attacker must modify or interfere with the DRAM address (*i.e.*, the row, column, or bank bits) being transmitted over the memory bus. However, modifying a single bit in a single command is challenging due to the high frequency at which DDR4 operates. Such manipulations require very precise control logic operating at multiple GHz. Instead, a more rudimentary solution is to simply ground one (or more) CA lines. For instance, from Figure 2, we can see that all lines except `A16–14`, `A12`, and `A10` have no semantic role outside of the `MRS` command other than containing bank, chip ID, and row or column bits. Outside of the commands that require these bits, these lines only need to hold arbitrary (but valid) values. This makes them interesting targets for injecting faults; tying one of these lines to, e.g., ground, will effectively alter the address received by the DIMM without impacting the other commands. This creates aliases: two distinct system addresses—one where the bit is set to zero and another where it is set to one—end up mapping to the same physical location in DRAM. Since this interference occurs after the CPU has issued its commands, the CPU remains unaware of the manipulation.

While an attacker could consider targeting bank-group or bank-address lines, these bank bits are usually mapped to the lower bits of the physical address [25, 26], making controlling the aliases difficult. Likewise, faulting one of `A9–0` would affect both a row and column bit, complicating the aliasing. Additionally, CID bits are only defined for 3DS devices, restricting the impact of the interposer to only high-density DIMMs. As a result, we target the row address bits `A11` and `A13`. These are present on all DDR4 DIMMs and, aside from the `MRS` command, are only used to define the 11th and 13th row address bit. Since the `MRS` command is only used to configure the DRAM registers during boot, this will not create any conflicts at runtime.

Physical Interposer. To facilitate the manipulation of `A11` and `A13`, we designed a DDR4 interposer that sits between the CPU and DIMM, shown in Figure 3. The interposer places analog switches on either of the two aforementioned row address lines, allowing it to either pass the values undisturbed or break the connection and ground the line to

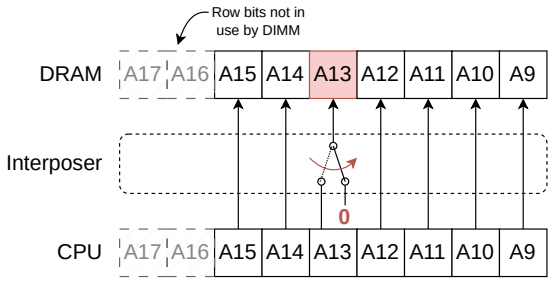


Figure 4. Interfering with DDR4 addressing. A switch, controlled by a microcontroller, dynamically pulls a DRAM line to ground.

the DIMM. A schematic overview of the operation of this interposer is given in Figure 4. During normal operations, all CA lines are directly connected to the CPU. However, through a switch, the attacker can force either `A11` or `A13` to ground, creating aliases in the CPU's physical address space. When active, the DRAM chips will always see this bit as low, regardless of the actual value sent out by the CPU. Note that the resulting aliasing only works in one direction; only the DRAM location with the bit not set can be accessed while the interposer is active.

The switches are controlled by a microcontroller (Raspberry Pi Pico 2), which enables their activation only for the duration of the attack. This microcontroller is connected to a second, attacker-controlled device, providing an interface to toggle the switches dynamically at runtime. This enables the interposer to bypass any boot-time alias checking as present in Scalable SGX and SEV-SNP [6, 7].

A complete bill of materials is provided in Table 1. Our design consists mainly of off-the-shelf components. The only custom part, the Printed Circuit Board (PCB), is a standard four-layer board that can be manufactured inexpensively, even in low quantities. The total cost of our interposer is less than $50, many orders of magnitude cheaper than existing attacks that employ commercial signal analyzers to snoop the memory bus [2].

TABLE 1. BILL OF MATERIALS FOR OUR DDR4 INTERPOSER.

Item	Part Number	Qty.	Cost
Printed Circuit Board	N/A	1	$18.49
DIMM connector	CONN-DDR4-288-SM	1	$16.00
Microcontroller	Raspberry Pi Pico 2	1	$5.00
Analog switch	ADG902BRMZ	2	$8.00
Voltage regulator	LD1117S25TR	1	$0.65
Miscellaneous resistors/capacitors			$0.50
Total			$48.64

3.3. Bypassing RCD Parity Checking

Unlike UDIMMs, RDIMMs contain an RCD chip to reduce the electrical load on the memory bus. In addition to buffering commands, the RCD optionally performs parity checking on the CA bus. This (even) parity signal is sent out by the memory controller and covers all CA bits to

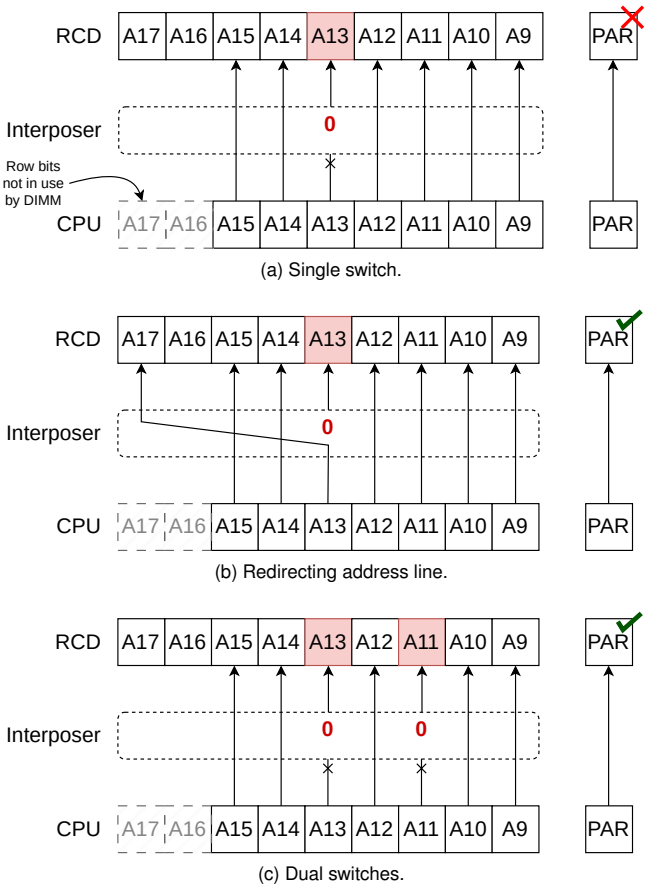


Figure 5. Bypassing RCD parity checks.

detect transmission errors between the CPU and DIMM. As a result, faulting a single bit on RDIMMs will cause a parity error that will result in the command being discarded, as shown in Figure 5a. Additionally, upon detection, the RCD will notify the CPU through the `ALERTn` line. To avoid triggering parity errors, we identified three practical techniques to bypass this verification.

Disabling RCD Parity. The RCD only performs parity checking if it is explicitly enabled in the `F0RC0E` configuration register. In theory, the RCD is connected to the SMBus, exposing these registers to attackers. In our experiments, however, we found the I^2C communication of the RCD to be disabled on both of our test platforms, preventing this. While the I^2C interface should be enabled by default, the BIOS may choose to disable it through the `F0RC2x` register. However, interestingly, we found that our AMD platform exposed the parity configuration through the BIOS settings.

Redirecting Data Lines. While the RCD ensures the integrity of the command lines, it is agnostic to the actual size of the connected DRAM chips. As a result, it will perform the parity calculation over all received bits, regardless of whether they are connected to the DRAM chips. Since the parity calculation is simply an XOR of all CA bits,

TABLE 2. OVERVIEW OF EVALUATION SYSTEMS USED IN THIS PAPER.

System	TEE	Mainboard	CPU	DIMM(s)	DRAM
AMD_1	SEV-SNP	ASRock ROMED8-2T	EPYC 7313P	1×MTA36ASF8G72PZ-3G2F1 1×MTA36ASF4G72PZ-3G2R1	DDR4 RDIMM
$Intel_1$	Client SGX	Dell D11S001	Core i5-6500	1×HMA41GU6AFR8N	DDR4 UDIMM
$Intel_2$	Scalable SGX	Supermicro X12DPi-NT6	Xeon 6330	16×HMAA4GR7AJR8N-XN	DDR4 RDIMM

swapping or redirecting bits will produce the same parity output. Specifically, instead of only tying an address bit to ground, we can redirect its original value, sent by the CPU, to `A17`, the most significant row address bit, as shown in Figure 5b. If this bit is not in use, it will simply be ignored by the DRAM chips, similar to the principle used in [5].

Multiple Switches. Since the parity function XORs all bits together, tampering with two bits will result in the same parity bit. If both bits switch their value simultaneously, the changes cancel out, and parity is preserved. More precisely, both `A11` and `A13` can be fitted with a switch to simultaneously ground these lines, shown in Figure 5c. Memory locations where these two lines are high will be aliased with a different location where these bits are low, assuming the switches are active. Crucially, this technique only preserves parity when both bits have the same value. When only a single address line is high, grounding both will result in a parity mismatch. Therefore, the attacker must carefully control the memory layout to ensure that the system does not use these addresses. This aligns with the TEE threat model, in which the adversary is assumed to have full control over the host operating system. As such, the attacker can control the memory allocation to only allocate pages that will not crash the system.

4. Interposer Evaluation

We first evaluate the effectiveness and stability of our custom DDR4 interposer before mounting end-to-end attacks on Intel Scalable SGX (Section 5) and AMD SEV-SNP (Section 6).

4.1. Experimental Setup

We evaluated our interposer on three systems, listed in Table 2 and encompassing AMD SEV-SNP, Intel Scalable SGX, and Intel Client SGX. In each system, a single DDR4 DIMM is equipped with our custom interposer. Figure 6 shows the interposer installed in the AMD SEV-SNP and Intel Scalable SGX platforms.

Memory Layout. To avoid unintended aliasing of critical memory regions (e.g., kernel memory, ACPI memory-mapped I/O, etc.) via the interposer, the attacker must carefully control the memory layout. These sensitive regions typically reside within the lower 4 GB of the physical address space.

On AMD_1, we leveraged BIOS options to disable memory interleaving. This allowed us to map the entire lower physical address space to an unaffected DIMM. On $Intel_2$, a dual-socket system, the memory regions for each socket are logically separated. Since we install a single interposer on the second socket, the lower physical address space associated with the first socket remains unaffected. Furthermore, the Linux kernel `memmap` parameter can be used to reserve the entire physical memory range corresponding to the second socket, preventing unintended allocations to the interposer-connected DIMM that may affect system stability.

Unstable Regions. On RDIMMs, the interposer may introduce unstable memory regions due to RCD parity checking, as discussed in Section 3.3. When using two switches (cf. Figure 5c), parity checks fail if only one of the two associated address lines is active, resulting in a system crash. To alleviate this, the `ALERTn` line can be disconnected, preventing the DIMM from reporting the parity error to the CPU [32]. Alternatively, since the physical-to-DRAM address mapping is deterministic, the affected regions can be reserved using the Linux kernel's `memmap` parameter to prevent the OS from allocating them.

To identify the unstable regions on our platform, we reverse-engineered the physical-to-DRAM mapping of the two affected row address bits `A11` and `A13` by observing which physical address ranges trigger system crashes when only a single switch is active. These crashes correspond to cases where the respective CA line is driven high. On AMD_1, the affected row address bits `A11` and `A13` correspond to physical address bits 28 and 30, respectively. On $Intel_2$, they correspond to bits 31 and 33, respectively.

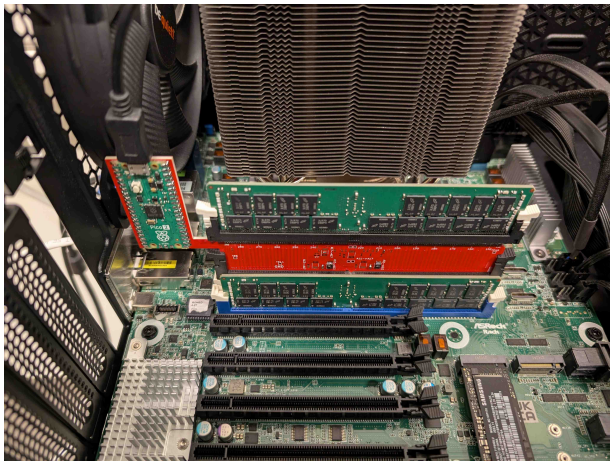
4.2. Evaluation of RCD Parity Check Bypassing

Single Switch. On $Intel_1$ (Client SGX with UDIMM without RCD parity checking), a single switch on either `A11` or `A13` reliably introduced aliases. These bits correspond to physical address bits 28 and 30, respectively.

On both SEV-SNP and Scalable SGX server systems equipped with RDIMMs, pulling a single address line causes a system crash when accessing affected addresses due to RCD parity checking (cf. Figure 5a). Interestingly, the BIOS on our ASRock motherboard (AMD_1) provides an option to disable RCD parity checking. We experimentally verified that disabling RCD parity checking this way indeed suppresses parity errors and enables aliasing.

Redirecting Data Lines. To bypass RCD parity checking, we experimented with redirecting `A11` or `A13` to the unused

(a) Intel_2.



(b) AMD_1.

Figure 6. Battering RAM DDR4 interposer installed in our test systems (cf. Table 2).

row bit `A17` (cf. Figure 5b). This technique was successful on AMD_1, but failed on our Intel system. We suspect this is because the Intel system explicitly disables the `A17` input buffer through an RCD configuration register (`F0RC08`), preventing it from participating in the parity calculation.

Multiple Switches. Finally, we evaluated both RDIMM platforms using interposers with two switches (cf. Figure 5c). We confirmed that pulling two bits simultaneously indeed bypasses RCD parity checking and successfully introduces aliasing. This approach may still introduce unstable memory regions, where only one of the two bits is set, which must be carefully mapped out to avoid crashes, as explained above.

As this technique is the most general, we use an interposer with two switches on both Intel_2 and AMD_1 for the remainder of the paper.

Stability. To evaluate the impact of the interposer on system stability, we performed memory stress tests using MEMTEST86+ and `stress-ng`. Due to unstable memory regions, MEMTEST86+ can only be run when the interposer is inactive. We tested each system at its default memory speeds (i.e., DDR4-3200 for AMD_1, DDR4-2133 for Intel_1, and DDR4-2933 for Intel_2). During these memory stress tests, we did not observe any crashes with the interposer, whether inactive or active.

We did, however, occasionally observe random crashes when accessing aliased Enclave Page Cache (EPC) pages when prototyping the Scalable SGX attacks on Intel_2. Although the precise cause of these occasional crashes remains undiagnosed, they only required a system reboot and did not prevent successful memory aliasing attacks.

4.3. Discussion

Feasibility. As demonstrated by the memory stress tests, the interposer remains stable at the default DRAM clock frequency. As such, our interposer has no discernible performance overhead and remains completely invisible to the operating system.

Additionally, as the physical-to-DRAM address mapping is fixed, the aliasing is fully deterministic. If the victim location and its alias are known, its contents can be captured and replayed with 100% accuracy.

Limitations. To enhance performance, modern systems may interleave memory across multiple DIMMs. As a result, on systems with multiple DIMMs, a single interposer can only observe a fraction of the memory lines belonging to a page. For instance, on Intel_2, a dual-socket system with 16 DIMMs, a single interposer can access only $1/8$th of the cache lines for a given page within the socket it is attached to. While installing an interposer on every DIMM would eliminate this limitation, we found that in practice, a single interposer combined with careful memory allocation was sufficient to carry out our attacks.

5. Breaking Scalable SGX

In this section, we describe how the interposer can be used to break the integrity and confidentiality of Scalable SGX, Intel's second-generation SGX for server environments. While Scalable SGX comes with built-in countermeasures against aliasing attacks, making it immune to BadRAM attacks [5], we show how the dynamic nature of the aliases introduced by the interposer, along with carefully allocated EPC pages, can bypass these mitigations. Additionally, we show how the interposer breaks Scalable SGX's confidentiality, relying on the static nature of the memory encryption. Finally, we describe a severe end-to-end attack that extracts the provisioning key from the Provisioning Certification Enclave (PCE), allowing an attacker to generate valid quotes without access to the system.

The ability to perform these attacks stems from the deterministic encryption provided by Scalable SGX. While the

original Client SGX provided a strong Memory Encryption Engine (MEE) [9], providing cryptographic confidentiality, integrity, and freshness, Intel has moved away from these strong protections due to their inherent limitations: integrity and freshness require cryptographic Message Authentication Codes (MACs) and an integrity tree, creating performance and storage overhead and severely limiting the protected memory size [6]. Scalable SGX, instead, relies on TME for confidentiality, using a single key for the entire SGX-reserved memory range [6]. TME encrypts the entire memory range using AES-XEX, a static memory encryption scheme with an address-based tweak. Contrary to Client SGX's MEE, TME does not provide integrity or freshness and instead relies on additional access control mechanisms to prevent access to the ciphertexts.

5.1. Aliasing Mitigations

Due to the deterministic nature of the memory encryption, additional measures are required to avoid ciphertext side channels [3, 4] and ciphertext replay attacks [5] via memory aliasing. Intel distinguishes between two types of aliases: outside-in and inside-in aliases, where an EPC page aliases with another page outside or inside the EPC, respectively [6]. To this end, Scalable SGX features two defenses to prevent aliasing: the owner bit and a boot-time alias check.

Owner Bit. To prevent outside-in aliasing, SGX repurposes one of the ECC bits to store an owner bit, which indicates whether the cache line belongs to enclave or regular memory. The memory controller sets this bit on every memory write, depending on the current execution context. On every read, it checks whether it matches the current domain. Upon a mismatch, the CPU will either return a fixed pattern of all zeros (when reading an EPC cache line from a non-secure address) or disable SGX in the other case.

Boot-Time Alias Check. The owner bit is only effective against aliases that cross the security domain. Any aliases between two EPC locations will remain undetected. To prevent inside-in aliases, `MCHECK`, or the Alias Checking Trusted Module (ACTM) starting with 4th generation Xeon, performs an alias check at boot time [34]. If any aliases are detected, SGX is disabled. This check can effectively mitigate attacks such as BadRAM [5], where the modified SPD causes the aliases to be present from the start. However, any aliases introduced *dynamically* at runtime, such as with our interposer, will remain undetected.

5.2. Interposer Attack

As discussed in the previous section, existing SGX countermeasures are insufficient to prevent interposer-based aliasing attacks. The boot-time alias checks can be bypassed by only activating the switches during the attack. Similarly, the owner bit only provides logical isolation; aliases between two EPC pages are not detected. We exploit these limitations to build an end-to-end attack on Scalable SGX.

Allocating Aliasing Pages. To perform an aliasing attack within the EPC, the attacker needs fine-grained control over the EPC memory allocation to ensure attacker and victim reside at aliasing locations. To achieve this, we extended the out-of-tree Linux SGX driver to provide precise control over the allocated pages. We expose a new method in the driver's `ioctl` interface that allows the attacker to specify the desired physical address for the next EPC allocation(s). This is facilitated through minimal changes in the memory allocator; instead of returning the first page in the free-page list, we iterate through the list until the physical address is found. If no specific address is specified, the modified driver returns pages that are unaffected by the interposer (*i.e.*, where both `A11` and `A13` map to zero).

To lay out aliasing victim and attacker enclaves, the attacker performs the following steps: (1) create the victim enclave, which will be allocated to unaffected memory pages if no address is specified; (2) identify the physical address containing the secret and compute its alias; (3) configure the modified driver to allocate the attacker's buffer at the alias.

5.3. Arbitrary Plaintext Access

Scalable SGX relies on TME to protect data in DRAM. Unlike SEV and TME-MK, which use a different encryption key for every confidential VM, TME uses a single key, generated randomly at boot-time, for the entire memory range. While this provides confidentiality against single-read attacks, such as cold boot attacks, it cannot defend against repeated reads or replay attacks. Furthermore, the use of a single key enables arbitrary plaintext access when capturing and replaying ciphertexts through aliases. Replayed ciphertexts, whether decrypted by the victim or attacker, will yield valid plaintext since the encryption function only relies on the physical address.

Arbitrary Plaintext Reading. Figure 7 provides a schematic overview of accessing plaintext data from victim enclaves. First, the attacker launches the victim enclave and sets up its own attacker enclave with a buffer that aliases with the victim secret (①). At this point, both enclaves are still independent and cannot yet interfere with each other. Next, the attacker enables the interposer (②). Because the attacker enclave was set up to alias with the victim, both enclave buffers will now point to the same location in DRAM. From here, the attacker can capture the victim's ciphertext by reading its buffer and storing it in a second, non-aliasing buffer (③). This access is allowed by the memory controller because it is performed from another enclave, thus matching the owner bit, which indicates enclave memory. Afterward, the attacker disables the interposer and terminates the victim (④). At this point, the attacker cannot yet access the plaintext because it was accessed from the alias and thus decrypted using a different tweak value.

To retrieve the plaintext secret, the attacker launches a second attacker-controlled enclave, using our modified driver to allocate it at the same physical address as the victim (⑤). After enabling the interposer (⑥), the attacker can

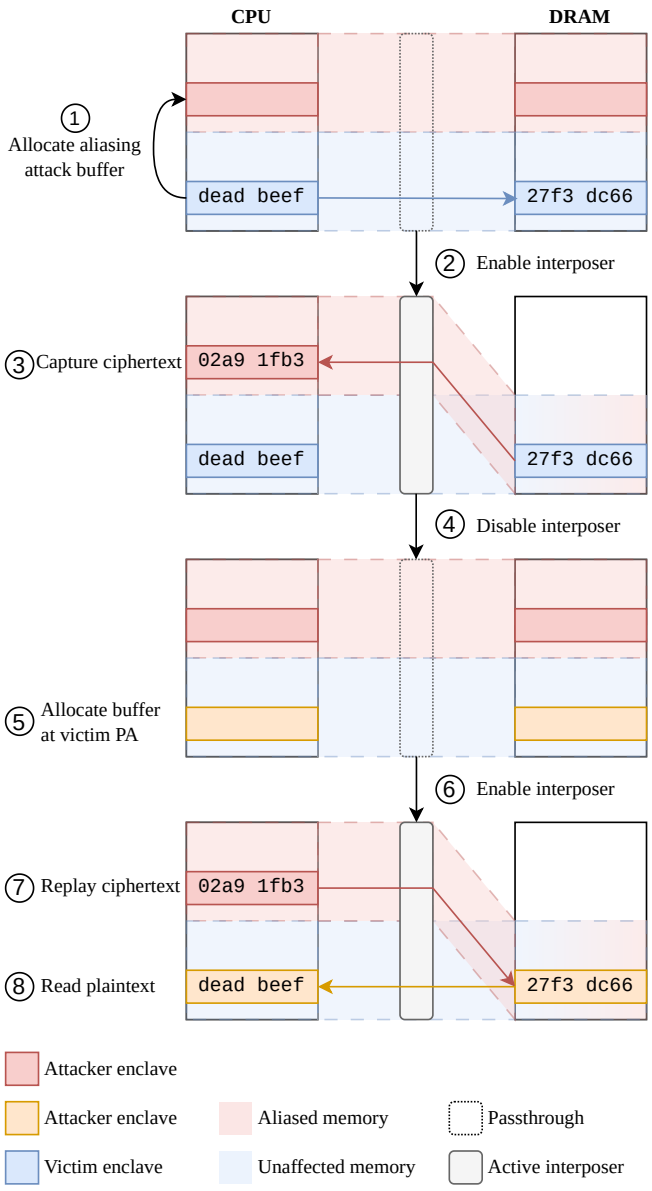


Figure 7. Overview of interposer-enabled arbitrary plaintext access in Scalable SGX. A single key is used to encrypt all enclave memory, with the physical address of the access applied as a tweak. By aliasing victim enclave pages, a Battering RAM adversary can capture ciphertexts in a first attacker enclave and later replay them into a second attacker enclave mapped to the victim's original physical address, recovering the plaintext.

now replay the captured ciphertext into the second attacker enclave (⑦). Because this second enclave was allocated at the same physical address as the original victim, reading from this location will yield the original plaintext as it is now decrypted with the correct tweak (⑧).

Arbitrary Plaintext Writing. The same methodology can also inject arbitrary plaintext into victim enclaves. Instead of capturing ciphertext from the victim, the attacker first writes controlled plaintext into an attacker-controlled enclave and captures the corresponding ciphertext from an alias using the interposer. Afterward, the attacker launches the victim, ensuring it is allocated at the same physical address, activates the interposer, and replays the previously captured ciphertext from the alias. Like before, TME's use of a single key means that the encryption relies solely on the physical address. When the victim reads the modified location, it will see the attacker's plaintext as it was allocated to the same physical address using our modified driver.

Evaluation. To demonstrate the feasibility of this attack, we implemented a basic proof-of-concept on Intel_2, a dual-socket system containing a total of 16 DIMMs. Instead of installing 16 interposers, we instead install a single interposer on the second socket. While this limits the aliasing to certain physical addresses, it allows us to reserve the entire physical memory range corresponding to the second socket, preventing unintended allocations to the interposer DIMM that may affect system stability.

Note that the eight DIMMs belonging to one socket are interleaved in the physical address space to improve performance. As a result, a single interposer only affects one out of eight cache lines within a physical page. The smallest contiguous size that could be aliased with a single interposer on our system is 256 B. While installing eight interposers would solve this problem, we found that a single interposer is sufficient to carry out our attacks.

We create a dummy victim enclave that allocates a buffer and initializes it with a known value. Using our modified SGX driver, we ensure that all victim pages are allocated on unaffected memory pages (*i.e.*, `A11` and `A13` not set). We then interrupt the victim and retrieve the physical address of the buffer, from which we compute the aliasing location. After enabling the interposer, we capture the ciphertext through the attacker enclave. We then terminate the victim and launch the second attacker enclave, assigning its buffer to the same physical address as the victim's buffer using our modified SGX driver. Finally, we enable the interposer and replay the ciphertext into our attacker buffer. We verified that data read by the second attacker enclave matches the value set by our dummy victim, thus successfully extracting plaintext from SGX enclaves.

5.4. Extracting Provisioning Key

To demonstrate the practicality of interposer-based aliasing attacks on Scalable SGX, we perform a destructive end-to-end attack that fully breaks SGX's remote attestation guarantees. This serves as a convincing demonstration that the platform is fundamentally compromised—akin to prior high-impact attacks that extracted SGX attestation keys via transient execution CPU vulnerabilities [35, 36, 37, 38, 39], which have since been mitigated through microcode updates and Trusted Computing Base (TCB) recovery. While those earlier attacks targeted Client SGX's Enhanced Privacy ID (EPID)-based attestation model, we, for the first time, break Scalable SGX's newer data-center third-party attestation framework [40]. First, we describe the attestation flow when

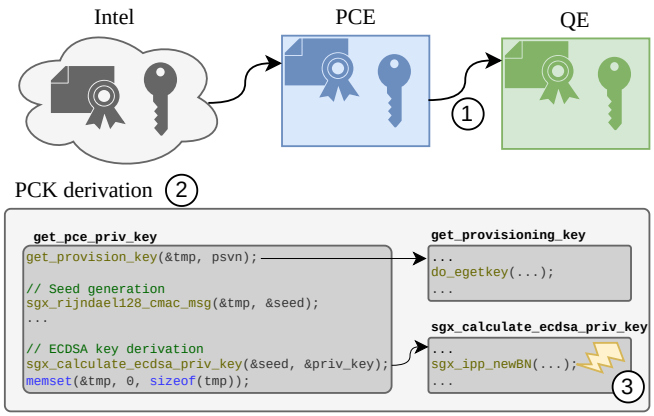


Figure 8. Overview of the PCE attack. The attacker requests the PCE to certify an enclave (①). This ECDSA key is generated from the hardware provisioning key (②). While this key is on the stack, the attacker interrupts the PCE (③), and retrieves it using our plaintext primitive (cf. Figure 7).

using a third-party Quoting Enclave (QE). During this process, the third-party QE is certified by the PCE using a hardware-provided key. Using the interposer, we show how a physical attacker can extract this hardware key, allowing them to certify arbitrary QEs, even without access to the system.

QE Certification. To enable attestation in scenarios where SGX's original EPID-based attestation is unsuitable, such as offline environments, Intel introduced the Datacenter Attestation Primitives (DCAP) [40]. DCAP introduces a range of tools and online services that enable third parties to create their own attestation infrastructure. In particular, DCAP enables third-party QEs, allowing local attestation and quote generation instead of relying on Intel's QE.

Central to the DCAP attestation flow is the PCE, an Intel-signed architectural enclave responsible for certifying third-party QEs. The PCE issues certificates to QEs by signing their public keys with a platform- and TCB-specific ECDSA key, the Provisioning Certification Key (PCK). The PCE, in turn, is signed by Intel's root certificate, creating a verifiable certificate chain, enabling the authentication of the quote and platform state at the time of quote generation.

The PCK is derived from the provisioning key, a 128-bit AES key rooted in hardware, and only accessible to Intel-signed enclaves with the `ProvisionKey` attribute, such as the PCE. When certifying a new QE, the provisioning key is retrieved using `EGETKEY` and is used to derive the PCK, after which the key is cleared from memory. The derivation ensures that the PCK is unique for a given platform and TCB and does not require persistent storage to store the private key.

Provisioning Key Extraction. The PCE uses the PCK, derived from the provisioning key, to sign and certify enclaves. If an attacker were to obtain the provisioning key or the PCK, they could forge certificates for arbitrary enclaves, even without access to the victim platform. This could, for instance, be exploited in a supply-chain attack. The extracted key would remain valid until a TCB recovery or SGX factory reset. This compromises the certification chain and, thereby, the attestation in Scalable SGX.

We extract the hardware provisioning key using the plaintext access discussed before, Figure 8 provides a summary. We prepare the PCE by allocating its pages into aliasing memory through the modified SGX drivers. Using the arbitrary read and write primitive described before, we are able to extract the provisioning key while it is loaded in memory. However, the provisioning key is only present on the stack during PCK derivation and is cleared afterwards. This derivation is performed each time the PCE certifies a new QEs (①). In order to extract the key, the attacker, therefore, has to interrupt the PCE during key derivation (②). More specifically, we modify the page tables to induce a page fault at the code page containing `sgx_ipp_newBN` (③), a helper function in Intel's Integrated Performance Primitives (IPP) library used during the ECDSA key generation. This page fault transfers control back to the attacker while the provisioning key remains on the stack, enabling the attacker to capture it from the alias.

With the PCE interrupted, the attacker can set up the attacker enclave with a buffer at the aliasing address to the provisioning key's stack location. After enabling the interposer, the attacker can capture the ciphertext containing the provisioning key by reading from the buffer set up in the attacker enclave. The attacker then disables the interposer, terminates the PCE, and launches the second attacker enclave with a buffer at the same physical addresses as the original PCE. By enabling the interposer and replaying the ciphertext, the attacker can now get a plaintext memory dump of the PCE by reading from the second buffer. By capturing the stack contents using this technique, the attacker can locate and extract the provisioning key. In practice, the end-to-end PCE attack completes within seconds and executes deterministically, with no noise (aside from occasional non-disruptive crashes that require a reboot).

While the PCE theoretically employs stack randomization, we found that the provisioning key is consistently located at offset `0x1c_eeaa`. This offset was determined by locating the constant string `PAK_KEY_DER`—used as input to AES-CMAC for key derivation—which is located at offset `0x20` from the provisioning key. We verified the correctness of the extracted key by matching it against valid signatures generated by the genuine PCE. The extracted key allows an attacker to forge attestation certificates for *arbitrary* QEs, fundamentally undermining SGX's remote attestation guarantees, essentially enabling active man-in-the-middle attacks on *any* application enclave, or even spoofing valid responses from the compromised platform without requiring further access to it.

6. Breaking AMD SEV-SNP

In this section, we demonstrate how our interposer can bypass the boot-time alias check introduced by AMD

in response to BadRAM [7]. This alias check is designed to restore the integrity guarantees to SEV-protected VMs by disabling the system if aliases are detected. The result of the alias check is communicated through the `ALIAS_CHECK_COMPLETE` field of the guest attestation report and platform info.

6.1. Bypassing Alias Checks

While the alias check effectively mitigates SPD-based aliasing attacks, the interposer enables the introduction of aliases at runtime. Since this check only runs once at boot time, these aliases cannot be detected.

Setup. We evaluated our DDR4 interposer on AMD_1, running the latest available Platform Initialization (PI) firmware v1.0.0.D and SEV firmware v1.55.29. We experimentally verified that this latest PI detects spoofed SPD contents, refusing to boot when creating memory aliases using BadRAM [5]. To ensure system stability, we equipped our system with two DDR4 DIMMs, one for regular memory and one module, installed on the interposer, for introducing aliases. This setup is also shown in Figure 6b. We attached the microcontroller controlling the interposer to a separate machine, enabling us to synchronize the activation and deactivation of the switches on the interposer. While the BIOS on our system allowed us to disable RCD parity, we opted to leave it enabled, instead falling back to adding switches to both `A11` and `A13` (cf. Figure 5c).

To avoid unintended interference with kernel or firmware memory regions, we ensured all allocations were performed to either the first, unaffected DIMM, or to the safe regions of the interposer DIMM (*i.e.*, where `A11` and `A13` are zero). To achieve this, we disabled memory interleaving in the BIOS and placed the unaffected DIMM in the lower address range, covering the first 4 GB of physical memory. Additionally, we relocated the private memory regions (*i.e.*, those used by the Secure Processor (SP), System Management Unit (SMU), and CC6), which typically reside at the top of physical memory, to the unaffected DIMM through BIOS options. Finally, to prevent accidental aliasing, we reserved all memory regions affected by the interposer (*i.e.*, those with either `A11` or `A13` set).

Evaluation. We verified that the alias check was completed successfully by querying the `ALIAS_CHECK_COMPLETE` bit in the guest attestation report. While the switches remain deactivated, the interposer passes through the CA signals undisturbed at boot time, making it impossible to detect.

To test runtime aliases, we grounded both `A11` and `A13` on the interposer. We then manually scanned the memory region, identifying aliasing regions and regions where the system crashed. On our system, we successfully identified aliases and found that `A11` and `A13` corresponded to bit 28 and 30 of the physical address, respectively, after subtracting the PCI bus offset due to memory hoisting [41]. As a result, the alias is found by XORing the offset physical address with `0x50000000`.

6.2. Attestation Attack

We experimentally demonstrate that interposer-based aliases re-enable the attacks outlined in BadRAM. We reproduce their attacks, including their full attestation breach.

Attack. During remote attestation, the SEV SP takes a launch measurement of the VM and stores this digest in the guest context. This launch digest is later used to verify the VM's integrity by comparing it with the owner-provided ID block or dynamically validating it inside the guest VM. While the guest context is encrypted, the same key is used for all guests. Because the employed AES-XEX encryption scheme is deterministic, this enables replay attacks on the guest context.

BadRAM exploited this by first launching a genuine, unmodified guest and capturing the launch digest from the guest context through the alias [5]. Next, the attacker terminates the genuine VM and prepares to launch a modified VM. By ensuring the guest context is allocated at the same physical address, the encryption tweak will be identical. Next, the previously captured launch digest is replayed before finalizing the VM. Due to the static memory encryption, this will result in a valid launch digest that matches the genuine digest, even though the VM has been modified.

Evaluation. We successfully reproduced BadRAM's end-to-end attack on our fully patched AMD system, showing that the mitigations introduced to stop SPD-based aliasing are ineffective against runtime aliases. As a result, a physical adversary can bypass SEV's integrity claims to launch arbitrary VMs, breaking SEV's attestation feature. In practice, the end-to-end attestation attack requires booting two VMs and completes within seconds, executing fully deterministically without noise.

6.3. Discussion

Besides re-enabling ciphertext replay attacks, the aliases introduced by the interposer also re-enable attacks architecturally mitigated in SEV-SNP. For instance, by changing the ciphertext through the alias, attackers can launch fault injection attacks on cryptographic implementations [42]. Similarly, aliases enable attackers to modify the critical Reverse Map Table data structure, which was introduced in SEV-SNP to ensure memory layout integrity [5]. By maliciously altering these mappings, an attacker can decrypt arbitrary memory pages [43, 44].

Ciphertext Hiding. The latest, 5th generation EPYC processors introduce ciphertext hiding [8], which aims to isolate guest and host memory. When enabled, the host cannot access guest-encrypted memory regions and only observes a fixed value. This mitigates ciphertext side channels, which rely on observing the changes in ciphertext to deduce information about the processed data [3, 4, 45, 46].

While the ABI specification does not provide implementation details, we hypothesize that the ciphertext hiding

TABLE 3. VULNERABILITY OF COMMERCIAL TEES TO MEMORY ALIASING ATTACKS, INDICATING CIPHERTEXT READ (RD), WRITE (WR), OR REPLAY (RPL) OR PLAINTEXT (PT) ACCESS ON PLATFORMS WITHOUT THE LATEST MITIGATIONS (◐) OR UP TO DATE (●).

		BadRAM [5]				This work			
TEE	**Crypto**	*Rd.*	*Wr.*	*Rpl.*	*Pt.*	*Rd.*	*Wr.*	*Rpl.*	*Pt.*
Scalable SGX (§5)	AES-XTS	○	○	○	○	●	●	●	●
SEV-SNP† (§6)	AES-XEX	◐	◐	◐	○	●	●	●	○
Client SGX (§7.1)	AES-CTR	●	○	○	○	●	○	○	○
TDX (§7.2)	AES-XTS	○	○	○	○	○	○	○	○
Arm CCA‡ (§7.3)	AES-XEX/ QARMA	–	–	–	–	–	–	–	–

†BadRAM mitigated in AMD firmware update. ‡No commercial hardware available.

feature is similar to SGX/TDX's owner bit, splitting the memory into host and guest partitions. Under this assumption, aliasing attacks may circumvent this feature, similar to our Scalable SGX attack (cf. Section 5.2). However, as ciphertext hiding is available only on 5th generation EPYC processors, which only support DDR5, we cannot experimentally verify its effectiveness under interposer-based aliasing attacks.

7. Interposer Attacks on Other TEEs

Previous sections have investigated how our interposer can undermine the security guarantees provided by Scalable SGX and SEV-SNP. In this section, we investigate other commercial TEEs, including Client SGX, TDX, and Arm CCA. Table 3 provides an overview.

7.1. Client Intel SGX

Intel's first-generation commercial TEE, Client SGX, features a Memory Encryption Engine (MEE) that provides strong cryptographic protections for data stored in external memory, including confidentiality, integrity, and freshness [9]. To ensure confidentiality, SGX encrypts cache lines using AES-CTR, with the counter incorporating spatial and temporal components. This randomizes the ciphertexts on each encryption and makes SGX immune to, for instance, ciphertext side channels. To protect against replay attacks, the counters are included in a Merkle tree, which is updated on every write and verified on every read, ensuring the value read was the latest value that was written. As counters are updated on every write, replaying a value results in a counter mismatch, which is detected. Additionally, the integrity of each ciphertext is protected by a cryptographic MAC.

While SGX's MEE provides robust cryptographic protections against attackers with arbitrary read and write access, it does not conceal the memory access pattern. Prior work has shown that observing the ciphertexts reveals the memory write pattern, potentially leaking information in non-constant-time applications [5]. Because the counters are updated on every write, the ciphertext will change, even if the plaintext remains constant. As a result, by observing which ciphertexts change, the attacker can determine which physical addresses the victim enclave wrote to.

Evaluation. We reproduce BadRAM's attack on Client SGX [5], using our interposer to introduce the aliases instead of SPD-based aliasing. We equipped our Intel_1 system with an interposer on its single DIMM. Since Client SGX was implemented on consumer processors, which use UDIMMs, the attacker does not need to take into account parity; grounding a single bit is sufficient. We confirm the ability to observe the EPC through the aliases introduced by the interposer and reproduce the basic BadRAM proof-of-concept, which uses a dummy enclave to write to a random memory location. We verified that the EPC changes corresponded to the write pattern of the dummy enclave.

7.2. Intel TDX

Intel's latest generation TEE, Trust Domain Extensions (TDX), shifts the protection from individual enclaves towards entire VMs, or Trust Domains (TDs). In contrast to SGX, TDX allows these TDs to utilize the entire physical address range. To protect memory contents against physical attackers, TDX encrypts data stored to external memory using AES-XTS. However, in contrast to Scalable SGX, which uses a single memory encryption key for the entire physical address range, TDX supports TME-MK [24], providing each TD with a dedicated encryption key.

However, all commercial TDX machines only support DDR5, making them incompatible with our DDR4 interposer. While this prevents direct experimental verification of TDX's countermeasures, we investigate the impact of a DDR5 interposer on TDX below. We discuss DDR5-based interposers in Section 8.1 but leave the exploration and design of such interposers to future work.

Similar to SEV-SNP and Scalable SGX, TME-MK features a deterministic encryption scheme that is, without additional safeguards, vulnerable to attacks such as ciphertext side channels and ciphertext replay through aliases. To address this, TDX relies on the Alias Checking Trusted Module (ACTM) to perform a boot-time alias check covering the entire physical address space. Additionally, similar to Scalable SGX, TDX provides logical integrity by repurposing one ECC bit to store an owner bit. Furthermore, TDX adds cryptographic integrity via a 28-bit MAC in ECC bits [19, 47].

While the logical integrity could be bypassed by aliasing between two different TDs, as demonstrated in Section 5, the cryptographic integrity remains robust against simple aliasing attacks. This is because, while an interposer enables replay of the data bits containing the ciphertext, it cannot be used to replay the ECC bits, which store the cryptographic MAC. Replaying both data and ECC bits, while theoretically possible, would require a full-fledged interposer capable of intercepting and replaying the data contents. Such an interposer poses significantly higher engineering challenges.

7.3. Arm CCA

Confidential Compute Architecture (CCA) is Arm's latest TEE, enabling confidential VMs, called Realms, in the cloud. CCA is realized through the Realm Management Extension (RME), which introduces Realms and partitions the system into four physical address spaces: Non-secure, Secure, Realm, and Root [48]. The RME enforces memory isolation using the Granule Protection Table (GPT), which tracks ownership and state of memory blocks called granules [49]. Upon every memory access, a Granule Protection Check (GPC) verifies the access rights based on the GPT.

CCA ensures the confidentiality of DRAM contents through memory encryption with address-based tweak, implementing either QARMA or AES-XEX [22]. Additionally, vendors may choose to provide additional security features (e.g., by providing additional integrity and/or freshness). Critical data structures, such as the GPT, are either stored in on-chip memory or protected through additional integrity mechanisms when stored in external memory to limit exposure. Direct ciphertext access is blocked through GPC checks, preventing ciphertext side channels.

Similar to Scalable SGX and SEV, CCA's deterministic encryption scheme may make it vulnerable to interposer attacks, potentially enabling ciphertext read and replay attacks through aliases. Additionally, in systems without Memory Encryption Context (MEC), all Realms share a single encryption context, possibly enabling plaintext access similar to Scalable SGX (cf. Section 5.3). However, no CCA-capable hardware is currently available, and emulation platforms [50] do not include memory encryption, making it impossible to assert the provided security guarantees against physical memory interposers.

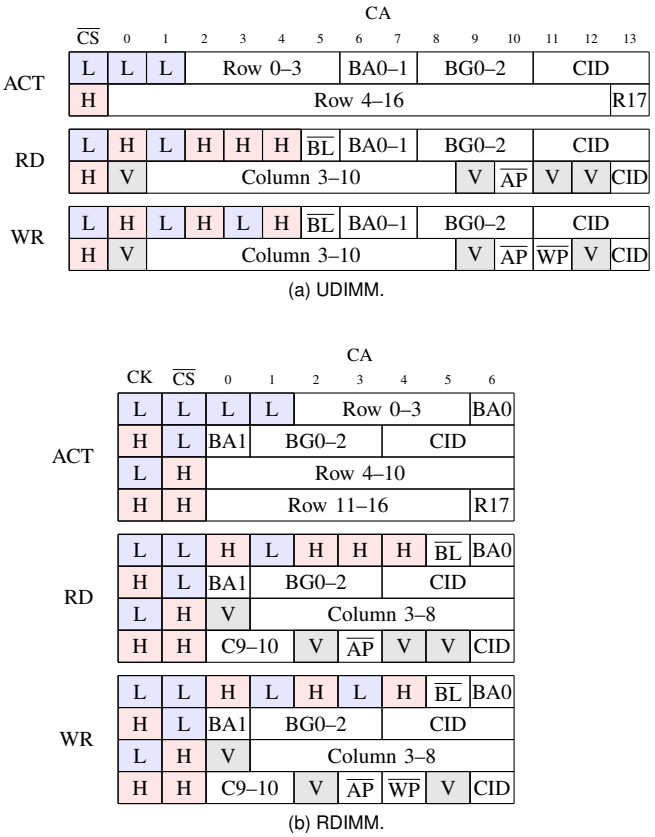


Figure 9. Activate, read, and write command encoding for DDR5 UDIMM and RDIMM [51]. CA values can be either high (H), low (L), valid (V), or don't care (X). BA and BG refer to bank group and bank address, respectively. CID is only defined for 3DS devices, otherwise valid.

8. Discussion and Mitigations

8.1. Interposer Attacks on DDR5

DDR5 introduces several architectural changes compared to DDR4, including the introduction of subchannels. To support this, the width of the CA bus has been reduced from 26 bits for DDR4 to 14 bits for DDR5, with commands now multiplexed over two consecutive cycles. Figure 9a shows a subset of the command encoding.

This multiplexing complicates manipulation, as each switch will now affect two logical bits. RDIMMs further complicate the matter, featuring an RCD that halves the CA width to only 7 bits. Each command is transmitted to the RCD in 7-bit chunks on a double data rate bus, as shown in Figure 9b, along with a parity signal that ensures command integrity. The RCD combines these to produce the full 14-bit CA encoding that is sent to the DRAM chips.

Pulling a single CA line will thus affect four logical command bits. However, each 7-bit chunk is accompanied by a parity bit, causing the RCD to detect the manipulation. If RCD parity cannot be disabled, as we observed on our DDR4 Intel server, the attacker would need to manipulate two lines simultaneously.

As a result, introducing aliases on DDR5 RDIMMs using the primitive from Section 3 is impossible without affecting other critical command bits. While not introduced as a security feature, these changes significantly complicate the ability to meddle with the bits using rudimentary switches. Introducing aliases on these systems using an interposer would either require the attacker to bypass the RCD parity checking or manipulate the command lines at DDR5 speeds. While this leaves TDX out of scope for this work, DDR4 RDIMMs, compatible with Scalable SGX and SEV-SNP, remain widely deployed in cloud environments.

8.2. Mitigations

The interposer introduces aliases after the addresses have been sent out by the memory controller. Furthermore, their dynamic nature makes detecting them from software or firmware impossible. As a result, robust countermeasures thus require hardware changes.

Memory Layout. Successful exploitation of the interposer-based attacks requires positioning the victim's pages to alias with the attacker's. This requires precise control over the memory allocation on the target system. Reducing the ability

of an adversary to perform this precise memory allocation would make exploitation of the attack considerably more challenging.

Compact CA Encoding. A compact encoding (such as in DDR5 RDIMMs) with commands spread over multiple clock cycles makes it more difficult to introduce aliases successfully using a passive interposer. Instead of affecting a single bit, these multiplexed schemes result in multiple faults, increasing the probability of corrupting other commands. Additionally, the presence of a parity signal, such as on RDIMMs, significantly increases complexity as faulting a single line will be detected. While these are not principled countermeasures and are mostly introduced to improve performance and reliability, they significantly increase the complexity of these attacks.

Integrated Memory. Integrated memory, such as high-bandwidth memory, is typically integrated into the package using 2.5D or 3D stacking. As a result, the command and data bus are internal to the package and thus inaccessible to a physical adversary. This effectively eliminates our interposer-based attacks. However, such integrated memory is less flexible, more expensive, and limited in capacity, making it unsuitable for systems requiring large or upgradable memory.

Strong Crypto. The attacks in this paper are enabled by the migration of TEEs to deterministic encryption schemes that omit integrity and freshness in return for larger protected memory sizes. Strong cryptographic memory encryption schemes, such as Intel's MEE in Client SGX [9], mitigate any physical tampering with the memory bus, including ciphertext replay attacks, though side-channel attacks are still possible [5]. These protections, however, come at a significant cost: performance overhead due to integrity and freshness checks, storage overhead to store the MACs, and limited protected memory sizes due to integrity tree scaling limitations. While recent academic works have improved upon this design [52, 53, 54, 55], they have yet to be adopted by commercial TEEs.

More recently, Intel's TDX re-introduces cryptographic integrity by embedding a 28-bit MAC in the ECC bits. While this does not offer freshness, it does mitigate alias-based replay attacks since those cannot replay ECC bits. It thus offers partial replay protection while maintaining low overhead and a large encrypted memory size.

9. Conclusion

Scalable TEEs employ deterministic memory encryption to protect data in DRAM. To prevent breaches through memory aliasing, they feature additional safeguards, such as boot-time alias checks. In this paper, we introduced a DDR4 memory interposer that enables dynamic aliasing, bypassing these mitigations. We demonstrated how our interposer enables arbitrary plaintext access to Scalable SGX enclaves, by exploiting TME's single key domain. We used this capability to extract the hardware attestation key, breaking the SGX attestation chain. Furthermore, we showed how the interposer bypasses recently introduced mitigations on SEV-SNP, re-enabling previously mitigated attacks. Our $50 attack shows that low-cost physical adversaries form a practical, yet underestimated, threat. While DDR5 increases CA complexity, preventing our primitive at present, it does not eliminate the underlying issue. We argue that memory encryption and TEE designs should consider the impact of these threats and provide strong, robust mitigations against low-cost physical adversaries.

Acknowledgements

This work was supported by the Research Fund KU Leuven, the Research Foundation – Flanders (FWO) via grant #1261222N, and the Flemish Government (Cybersecurity Research Program) via grant VOEWICS02. In addition, this work is supported by the European Commission through Horizon 2020 (ERC #101020005 BELFORT). This research was also partially funded by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/X03738X/1, EP/V000454/1, and EP/R012598/1. The results feed into DsbDtech. Jesse De Meulemeester is funded by an FWO fellowship (11PFE24N).

References

[1] S. F. Yitbarek, M. T. Aga, R. Das, and T. M. Austin, "Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 313–324.

[2] D. Lee, D. Jung, I. T. Fang, C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium*, 2020, pp. 487–504.

[3] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, "CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel," in *30th USENIX Security Symposium*, 2021, pp. 717–732.

[4] M. Li, L. Wilke, J. Wichelmann, T. Eisenbarth, R. Teodorescu, and Y. Zhang, "A systematic look at ciphertext side channels on AMD SEV-SNP," in *43rd IEEE Symposium on Security and Privacy (S&P)*, 2022, pp. 337–351.

[5] J. De Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede, and J. Van Bulck, "BadRAM: Practical memory aliasing attacks on trusted execution environments," in *46th IEEE Symposium on Security and Privacy (S&P)*, May 2025.

[6] S. Johnson, R. Makaram, A. Santoni, and V. Scarlata, "Supporting Intel SGX on multi-socket platforms," https://web.archive.org/web/20220822150148/https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf, Intel, White Paper, ID 843058, 2021.

[7] AMD, "Undermining integrity features of SEV-SNP with memory aliasing," https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3015.html, December 2024.

[8] ——, "SEV secure nested paging firmware ABI specification," AMD, Tech. Rep. 56860, Rev. 1.57, January 2025.

[9] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptology ePrint Archive*, 2016.

[10] AMD, "AMD SEV-SNP: Strengthening VM isolation with integrity protection and more," January 2020.

[11] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. F. Oswald, and F. D. Garcia, "VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface," in *30th USENIX Security Symposium*, 2021, pp. 699–716.

[12] A. Cui, "BADFET: Defeating modern secure boot using Second-Order pulsed electromagnetic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Aug. 2017.

[13] R. Buhren, H. N. Jacob, T. Krachenfels, and J. Seifert, "One glitch to rule them all: Fault injection attacks against AMD's secure encrypted virtualization," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 2875–2889.

[14] Z. Chen and D. Oswald, "PMFault: Faulting and bricking server CPUs through management interfaces or: A modern example of halt and catch fire," *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, vol. 2023, no. 2, pp. 1–23, 2023.

[15] X. M. Saß, R. Mitev, and A.-R. Sadeghi, "Oops..! i glitched it again! how to Multi-Glitch the Glitching-Protections on ARM TrustZone-M," in *32nd USENIX Security Symposium (USENIX Security 23)*, Aug. 2023, pp. 6239–6256.

[16] N. Mishra, A. Chakraborty, and D. Mukhopadhyay, "Faults in our bus: Novel bus fault attack to break ARM trustzone," in *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

[17] S. Johnson, R. Makaram, A. Santoni, and V. Scarlata, "Supporting Intel SGX on multi-package platforms," 2025. [Online]. Available: https://arxiv.org/abs/2507.08190

[18] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *2nd ACM International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[19] Intel, "Architecture specification: Intel trust domain extensions (Intel TDX) module," Intel, Specification 344425-005US, Feb. 2023.

[20] T. W. David Kaplan, Jeremy Powell, "AMD memory encryption," October 2016.

[21] D. Kaplan, "Protecting VM register state with SEV-ES," 2017.

[22] Arm, "Arm CCA security model," Tech. Rep. Arm DEN 0096, Version 1.0, August 2021.

[23] Intel, "Intel hardware shield – Intel total memory encryption," Intel, White Paper.

[24] ——, "Intel architecture memory encryption technologies," Intel, Manual 336907-004US, Rev. 1.4, August 2022.

[25] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-CPU attacks," in *25th USENIX Security Symposium*, 2016, pp. 565–581.

[26] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A knowledge-assisted tool to uncover DRAM address mapping," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[27] JEDEC, "DDR4 SDRAM," JEDEC, Standard JESD79-4B, Jun. 2017.

[28] ——, "DDR4 registering clock driver definition (DDR4RCD02)," JEDEC, Standard JESD82-31A.01, Jan. 2023.

[29] B. D. Hopkins, J. Shield, and C. North, "Redirecting DRAM memory pages: Examining the threat of system memory hardware trojans," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 197–202.

[30] Wii Brew Wiki, "Tweezer Attack," https://wiibrew.org/wiki/Tweezer_Attack, 2023.

[31] B. Dixon, "pin2pwn: How to root an embedded Linux box with a sewing needle," DEF CON 24, 2016.

[32] L. Cojocar, K. Loughlin, S. Saroiu, B. Kasikci, and A. Wolman, "mFIT: A bump-in-the-wire tool for plug-and-play analysis of rowhammer susceptibility factors," 2021.

[33] S. Gloor, P. Jattke, and K. Razavi, "REFault: A Fault Injection Platform for Rowhammer Research on DDR5 Memory," in *1st Microarchitecture Security Conference (μASC '25)*, Feb. 2025.

[34] Intel, "Intel Xeon scalable processors: NEX eagle stream platform, Intel platform security," Intel, Tech. Rep. 784473, August 2023.

[35] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium*, Aug. 2018, pp. 991–1008.

[36] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[37] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad: Cross-privilege-boundary data sampling," in *26th ACM Conference on Computer and Communications Security (CCS)*, Nov. 2019, pp. 753–768.

[38] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss, and F. Piessens, "LVI: Hijacking transient execution through microarchitectural load value injection," in *41st IEEE Symposium on Security and Privacy (S&P)*, May 2020, pp. 54–72.

[39] S. Van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," 2020.

[40] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for Intel SGX with Intel data center attestation primitives," 2018.

[41] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: Rowhammer attacks on AMD Zen-based platforms," in *33rd USENIX Security Symposium*, 2024.

[42] R. Buhren, S. Gueron, J. Nordholz, J. Seifert, and J. Vetter, "Fault attacks on encrypted general purpose compute platforms," in *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2017, pp. 197–204.

[43] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "SEVered: Subverting AMD's virtual machine encryption," in *Proceedings of the 11th European Workshop on Systems Security (EuroSec)*, 2018, pp. 1:1–1:6.

[44] M. Morbitzer, S. Proskurin, M. Radev, M. Dorfhuber, and E. Q. Salas, "SEVerity: Code injection attacks against encrypted virtual machines," in *IEEE Security and Privacy Workshops (S&P Workshops)*, 2021, pp. 444–455.

[45] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, "Ciphersteal: Stealing input data from TEE-shielded neural networks with ciphertext side channels," in *46th IEEE Symposium on Security and Privacy (S&P)*, 2025.

[46] ——, "Hypertheft: Thieving model weights from tee-shielded neural networks via ciphertext side channels," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[47] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX demystified: A top-down approach," *ACM Computing Surveys*, vol. 56, no. 9, pp. 1–33, 2024.

[48] Arm, "Arm realm management extension (RME) system architecture," Tech. Rep. Arm DEN 0129, Version B.a, November 2023.

[49] ——, "Learn the architecture - introducing Arm confidential compute architecture," Tech. Rep. Arm DEN 0125, Version 3.0, June 2023.

[50] A. Bertschi and S. Shinde, "OpenCCA: An open framework to enable arm cca research," in *8th Workshop on System Software for Trusted Execution (SysTEX 2025)*, 2025.

[51] JEDEC, "DDR5 SDRAM," JEDEC, Standard JESD79-5, Jul. 2020.

[52] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the PENGLAI enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021, pp. 275–294.

[53] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 416–427.

[54] M. Taassori, A. Shafiee, and R. Balasubramonian, "VAULT: Reducing paging overheads in SGX with efficient integrity verification structures," in *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2018, pp. 665–678.

[55] A. Inoue, K. Minematsu, M. Oda, R. Ueno, and N. Homma, "ELM: A low-latency and scalable memory encryption scheme," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2628–2643, 2022.

Appendix A. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper constructs a low-cost hardware interposer (<$50) that enables dynamic aliasing between different pages of memory in DDR4 memory. It uses the interposer to read plaintexts from encrypted enclave memory in both Intel Scalable SGX and AMD SEV-SNP.

A.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Creates a New Tool to Enable Future Science
- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) The paper demonstrates the practicality and ease of physical attacks on DDR4-based Intel Scalable SGX and AMD SEV-SNP.
2) Open sourcing the hardware and software design of affordable DDR4 interposer will offer an important resource for the community.
3) The paper did a responsible disclosure including response from Intel and AMD.

A.4. Noteworthy Concerns

1) It is known that a physical attacker capable of interposing on the DRAM bus can break modern TEE security guarantees, both Intel and AMD consider such physical attacks are out of the scope.
2) The attack is limited to DDR4 memory. For AMD, it only works on Zen3 processors (> 4 years old), and not on Zen4 or Zen5 (DDR5). For Intel, it does not work for 4th generation and above, which includes all CPUs with TDX support.