PowerHooK: Enabling Software-Based Power Side Channels Against AMD SEV Technologies via Transient-Execution Replay

Mathias Oberhuber[1] Martin Unterguggenberger[1] Martin Wistauder[1]
Andreas Kogler[2*] Rishub Nagpal[1] Stefan Mangard[1]

[1] *Graz University of Technology*
[2] *Graz University of Technology Alumni*

Abstract

Confidential computing technologies, such as AMD SEV, enable secure execution of cloud workloads on shared physical hardware. AMD SEV technologies implement the VM trust model through AMD SEV-ES, encrypting memory and CPU register state, and AMD SEV-SNP, providing integrity protection for VM memory. While AMD SEV provides heavyweight architectural isolation, it remains unclear whether it is susceptible to power side channels.

In this paper, we present PowerHooK, a new attack on AMD SEV technologies that enables software-based power side channels by speculatively replaying victim code paths via transient execution. Specifically, we repurpose page-fault-based transient replay to establish a transient-execution replay hook for power measurements. PowerHooK allows a malicious hypervisor to re-execute vulnerable victim code paths, thereby enabling continuous collection of power traces, reducing significant system noise. This capability allows the attacker to perform power analysis attacks on denoised datasets.

We demonstrate PowerHooK's methodology by recovering AES key bytes across all AMD SEV defenses. Here, the attacker only needs to consider 1320 samples to perform a CPA on AES-NI executed in AMD SEV-SNP running in an experimental setting. We systematically analyze architectural, speculative, and transient power leakage across different AMD CPU generations and evaluate how AMD's virtualization levels affect PowerHooK. Moreover, we present a real-world AES key byte recovery attack targeting VM-isolated cloud workers that run OpenSSL's constant-cycle AES-NI CBC implementation. Thereby, we demonstrate that transient replay gadgets are present in the OpenSSL library, showcasing that PowerHooK effectively enables the extraction of secrets.

1 Introduction

AMD Secure Encrypted Virtualization (SEV) [6] enables strong architectural isolation of virtual machines (VMs) that are broadly utilized by major cloud service providers to operate their cloud computing environments. Cloud computing comprises a comprehensive VM trust model, not only requiring the isolation of mutually untrusted guest VMs from each other but also the isolation of guest VMs from a potentially malicious hypervisor. In this VM trust model, it must be infeasible for the hypervisor to gain unauthorized access to the tenant's private data. For this purpose, AMD has introduced technologies, like SEV, SEV Encrypted State (SEV-ES), and SEV Secure Nested Paging (SEV-SNP) [44] that protect memory resources, secure the register state, and ensure confidentiality and integrity of VM resources in the cloud [26].

Prior work demonstrated various attacks on AMD SEV technologies [22, 23, 25, 35, 37, 45, 50–53, 62, 64, 65]. CIPHERLEAKS [37], for instance, introduced an attack exploiting ciphertext properties of AMD SEV VMs. CacheWarp [64] demonstrated software-based fault attacks on AMD SEV technologies that enable reverting cache lines of guest VMs to prior states. In addition, EntrySign [24] highlights that malicious CPU microcode can compromise the confidentiality and integrity of AMD SEV-SNP.

Beyond these targeted attacks, software-based power side-channel attacks, which enable the extraction of secret data across traditional security boundaries such as process or VM isolation, are a difficult-to-mitigate class of attacks and remain largely unexplored. Prominent examples include PLATYPUS [38], Hertzbleed [58], and Collide+Power [33]. These attacks reveal secret information, such as cryptographic keys, by analyzing software-accessible interfaces related to the system's power consumption. For example, PLATYPUS [38] performs Correlation Power Analysis (CPA) [11] using low-resolution power readings from the Intel RAPL interface [29].

In practice, a major challenge of software-based power side-channel attacks is substantial system noise, since only a small fraction of the analyzed signal corresponds to exploitable leakage. Therefore, effective exploitation requires collecting a large number of repeated measurements (often over multiple days) and applying averaging to remove noise impact. To suppress noise, prior work [38, 47, 58] utilized interfaces ca-

*Work done while affiliated with Graz University of Technology.

pable of triggering repeated encryptions, which provide powerful capabilities for an attacker. In a more restrictive setting, prior work [38, 57] demonstrated a zero-stepping approach for instruction repetition to mount software-based power side-channel attacks. While zero-stepping is a powerful technique leveraged by multiple side-channel attacks [13, 22, 62], it is limited in practice for power side-channel attacks [38]. Specifically, noise from the zero-stepping logic and the extended duration required for each measurement hindered successful exploitation, thus preventing leaking constant-cycle AES key bytes using zero-stepping [38,57]. Therefore, the full potential of software-based power side-channel attacks targeting confidential computing technologies (i.e., AMD VMs with SEV, SEV-ES and SEV-SNP enabled) remains unexplored. This raises several key questions: Is it possible to mount software-based power side-channel attacks against AMD VMs with SEV, SEV-ES and SEV-SNP enabled? How can an attacker repeatedly invoke victim code sequences that process secrets in a single run, to mount power-analysis attacks? Do coarse resolution, low sample rate of the power interface, and high noise hinder extraction of fine-grained secrets from constant-cycle algorithms, such as AES, inside a confidential VM?

In this paper, we address these challenges of software-based power side-channel attacks and target VMs with AMD SEV, SEV-ES and SEV-SNP enabled, extracting VM secrets by leveraging transient execution replay. Specifically, we present PowerHooK, a new software-based power side-channel attack that leverages attacker-controlled instruction replay of vulnerable victim code paths in the transient domain, and subsequently denoises resulting power traces. PowerHooK leverages a *transient-execution replay hook*, namely a faulting instruction executed by the victim (e.g., a page-faulting memory load), as in prior transient-execution attacks against Intel SGX [54]. The fault opens a transient window of successive instructions that the hypervisor can repeatedly trigger by leaving the fault unresolved. In more detail, PowerHooK exploits the power consumption of such a transient execution window that occurs after the victim VM accesses a not-present, on-demand mapped page, but before the resulting page fault causes a VM exit. This enables an attacker to consistently trigger victim CPU operations that process secrets in a single execution run, measure their power consumption via internal CPU interfaces such as RAPL or the Hertzbleed effect (observable via commonly available timers), and denoise the collected traces.

To evaluate the effectiveness of PowerHooK, we systematically analyze and compare data-dependent power leakage from constant-cycle instructions executed architecturally, speculatively, and transiently across different AMD platforms (mobile, desktop, and server). To quantify the overhead that confidential-computing technologies add to PowerHooK, we first measure the increase in replay latency while enabling progressively stronger levels of AMD's protection. Additionally, we quantify how these virtualization levels affect power leakage, highlighting that PowerHooK's replay enables effective instruction repetition to mount software-based power side-channel attacks across confidential VM boundaries.

We demonstrate the feasibility of our approach in a leakage analysis of a synthetic first-round AES CPA case study across AMD-V, SEV, SEV-ES, and SEV-SNP. Using a generic filtering step based on observed CPU frequency and latency, an attacker can leak an AES key byte using PowerHooK from a VM running AMD-V in only 3 min (i.e., 300 samples), from SEV in 2450 samples, from SEV-ES considering 3430 samples and from SEV-SNP in 1320 samples.

Finally, we present a real-world AES-CBC key byte recovery attack on OpenSSL, breaking the VM's trust model through PowerHooK. We show that an attacker can leak confidential AES key bytes by observing and replaying secret-processing code paths during a single run of OpenSSL's constant-cycle AES-NI multi-block CBC decryption on a VM-isolated cloud worker. Using PowerHooK, we recover AES key bytes from an AMD SEV-ES protected VM in approximately 3.7 h (i.e., 23 000 samples), highlighting the availability of replay hook gadgets in cryptographic libraries and the effectiveness of such attacks on real cloud workloads.

Contributions. In summary, our key contributions are:

(1) Denoising software-based power side channels. We present PowerHooK, a transient-replay measurement gadget exploiting page-fault-based transient replay for power trace collection. By transiently replaying victim code sequences from a single victim execution within a confidential VM, PowerHooK effectively amplifies and denoises transient power traces.

(2) Systematic analysis of transient power leakage. We systematically analyze data-dependent power leakage of architectural, speculative, and transiently replayed instructions beyond VM isolation boundaries of AMD SEV technologies.

(3) Synthetic AES CPA attack benchmark. We characterize leakage rates of a synthetic first-round AES CPA attack in the transient domain across different levels of virtualization, i.e., AMD-V, SEV, SEV-ES and SEV-SNP.

(4) Real-world OpenSSL AES key-byte recovery attack. We present a real-world known-ciphertext AES key-byte recovery using PowerHooK, extracting key bytes from an AMD SEV-ES protected VM in a single invocation of OpenSSL's AES-NI accelerated CBC implementation.

Outline. Sections 2 and 3 present background and threat model. Section 4 presents the PowerHooK attack. Section 5 details the leakage analysis. Section 6 demonstrates the synthetic transient AES-NI leakage analysis, and Section 7 presents a real-world AES key recovery on OpenSSL. Sections 8 and 9 discuss related work and conclude.

2 Background

In this section, we describe background on AMD SEV technologies, transient execution, and power side-channel attacks.

2.1 AMD SEV Technologies

Virtualization technologies (e.g., AMD-V) enable sharing of computing resources between multiple virtual environments on a single physical machine, i.e., by separating a *guest* from the *host* system. This enables efficient utilization of resources for desktop and server applications. Traditionally, the hypervisor has been seen as a trusted element within the security model. However, cloud computing requires a stronger security model, which excludes the hypervisor from the Trusted Computing Base (TCB). Thus, the confidential computing security model relies on strong isolation of workloads against cloud-co-located guests and *a potentially malicious host*.

AMD Secure Encrypted Virtualization (SEV) [6] targets this security model, protecting virtual machines (VMs) from a potentially compromised hypervisor. AMD SEV encrypts DRAM [5], providing cryptographic isolation of VMs and protection against physical attacks (e.g., cold boot attacks [27]). AMD SEV Encrypted State (SEV-ES) [1] encrypts CPU register content during a VM exit (i.e., switch to the hypervisor) [60]. AMD SEV Secure Nested Paging (SEV-SNP) [3] adds memory integrity, preventing encrypted memory content tampering [45, 46, 61]. Thus, AMD SEV technologies, including SEV-SNP and SEV-ES, allow running isolated workloads on the same machine, enabling confidential cloud computing.

2.2 Transient Execution

The microarchitectural design of modern CPUs incorporates techniques and optimizations (e.g., pipelining, speculative execution, and out-of-order execution) to improve system performance, increasing throughput and efficiency. Out-of-order execution [56] allows instructions to be executed as soon as input data and CPU resources (e.g., execution units) are available, instead of strictly following sequential program order. The instruction stream is reordered to optimize hardware utilization and reduce CPU idle times. Subsequent instructions independent of preceding computational results can be executed out-of-order while waiting for the completion of a previous (slower) instruction, such as a load from memory.

The Reorder Buffer (ROB) in an out-of-order pipeline ensures that instructions are retired in program order, i.e., by buffering computational results and either discarding them or committing them *in-order* to the architectural state, while the instructions may be transiently executed *out-of-order*. When an error or fault occurs during the execution of an instruction (or the execution is invalid due to misspeculation), the CPU processes so-called *transient instructions* [39]. In such cases, subsequent results in the ROB are discarded, and architectural state is restored prior to the faulting instruction. However, *transient* instructions, albeit never architecturally committed (i.e., discarded on a fault), leave traces in the microarchitectural state (e.g., in the memory cache). Transient execution attacks [14], such as Meltdown [39] and Spectre [31], exploit these changes in the microarchitectural state to infer secrets, such as cryptographic keys.

2.3 Power Side-Channel Attacks

Power side-channel attacks exploit information leakage, such as the static and dynamic power consumption of CMOS circuits, observable while processing secret data. Traditionally, adversaries target embedded cryptographic devices, collecting power "traces" with physical equipment, subsequently applying statistical analysis to recover confidential data, e.g., cryptographic keys [11, 32, 41].

Simple Power Analysis (SPA) [32] observes changes in the power consumption and matches them to specific operations or data operands processed on the system. Alternatively, Differential Power Analysis (DPA) [32], detects differences in power consumption of computations with controlled inputs to uncover secrets. Correlation Power Analysis (CPA) [11] replaces the differential approach with correlations to extract cryptographic keys. Moreover, CPA can leverage arbitrary power leakage models, such as the *Hamming distance*, i.e., the number of bit positions where two binary values differ, and the *Hamming weight*, i.e., the number of '1' bits in a binary value, to model the power consumption for key extraction.

PLATYPUS [38] is the first attack that demonstrated how to mount power side-channel attacks purely from software targeting desktop and server-class CPUs. The unprivileged Running Average Power Limit (RAPL) [29] software interface, reporting a CPU's power consumption, eliminates the need for physical proximity to the target device. Subsequent removal of unprivileged access to RAPL motivated Hertzbleed-type attacks [40, 58], which show that CPU frequency (and thus execution time) can act as a proxy for power, effectively re-enabling unprivileged power side channels from software.

3 Threat Model

The threat model of this work aligns with the AMD SEV VM trust model [6], i.e., protecting a VM against a compromised hypervisor. We assume a victim runs confidential workloads within a VM that is managed by a malicious hypervisor with complete system control. Specifically, the victim processes sensitive data that the hypervisor is not allowed to infer. Furthermore, we assume that the victim employs constant-time implementations of cryptographic algorithms without secret-dependent memory access patterns, thereby reducing attack surface against cache- or contention-based side-channel attacks. Additionally, we presume that the victim's VM is pro-

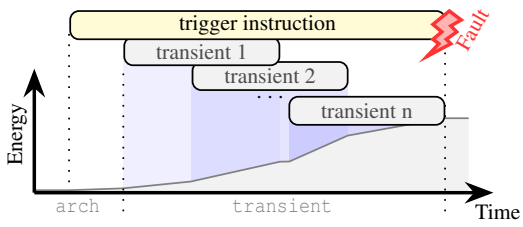

Figure 1: The energy consumption of multiple instructions executed architecturally and transiently. Notably, execution consumes energy, regardless of whether the result is later discarded due to an earlier fault, as given in the energy curve.

tected with AMD SEV with SEV-ES and SEV-SNP enabled, encrypting the victim's memory and register state.

The malicious hypervisor leverages the Running Average Power Limit (RAPL) interface to measure CPU power consumption when available to privileged software, i.e., the hypervisor. If access to RAPL is restricted or disabled [9], the attacker instead relies on commonly available power proxies, e.g., frequency-based leakage as in Hertzbleed [58]. The malicious hypervisor can hence mount a software-based power side-channel attack to leak secret data that is processed by the victim inside the VM.

4 The PowerHooK Attack

This section presents the PowerHooK attack, our novel approach for amplifying power leakage by applying transient execution replay across VM boundaries. First, we outline the foundations of software-based power leakage, combined with transient execution. Second, we detail the PowerHooK page-fault-based replay mechanism that transiently replays the guest's instructions, subsequently denoising power measurements, targeting constant-time algorithms processed in a single guest execution run.

4.1 Power Leakage of Transient Execution

Conventional, software-based power analysis commonly observes the power consumption of executed and architecturally committed instructions [38, 58, 59]. A typical scenario in this simplified model involves reading a value from a power interface, invoking an algorithm, and subsequently reading a second value from the power interface after the algorithm completes execution. The difference between the two measurements corresponds to the energy consumed by the algorithm. At a program level, this simplified view reflects the algorithm's total energy consumption.

However, modern CPUs integrate microarchitectural optimizations to execute instruction streams efficiently. One prominent optimization is out-of-order execution, which reorders the instruction stream (executing independent instructions out of order) to reduce CPU idle times. Notably, out-of-order-executed instructions consume energy during execution, regardless of when their results, that are cached in the ROB are committed to the architectural state.

Figure 1 shows a specific case of out-of-order execution; The energy consumption of multiple instructions executed in the transient domain, e.g., instructions executed out-of-program order but never architecturally committed due to an earlier fault. The example shows a triggering instruction, e.g., a memory interaction, which takes a long time to execute, stalling subsequent instructions that depend on its computational result. To mask memory latency, independent instructions within the instruction stream (denoted as *transient 1* to *transient n*), are executed out-of-order and thus consume energy when executed. When encountering a fault for the triggering instruction, the architectural state is recovered, i.e., the results in the ROB are not committed. Nevertheless, the consumed energy of the transient instructions leaves visible traces in the system's energy readings. Note, our work considers only unauthorized, transient instructions revoked due to memory-faults, previously considered impractical for power attacks due to noise [16].

The synthesis of an exploit leveraging the high-level idea of power leakage from transient execution in the context of secure VMs raises the following open research questions.

RQ1 How can an attacker continuously invoke and replay dedicated code sequences, e.g., of a victim VM, within a transient window to exploit their power consumption?

RQ2 Is the unpredictable measurement noise and the overhead of heavy-weight VM isolation hindering an adversary from mounting practical power analysis attacks?

RQ3 Is the coarse resolution and sample rate ($\sim$1 ms) of the software power interface accessible from the hypervisor (e.g., RAPL or Hertzbleed-style proxies) sufficient to extract AES keys from constant-cycle, hardware-accelerated implementations within a victim VM via power analysis?

RQ4 Can an attacker leak VM secrets by observing a single execution run of real-world constant-cycle cryptographic algorithms inside a victim VM, e.g., from OpenSSL?

4.2 PowerHooK: Transient Execution Replay Power Analysis

This section describes the PowerHooK attack, which demonstrates page-fault-based transient execution replay on AMD SEV technologies, to mount and denoise power analysis attacks targeting confidential VMs. PowerHooK leverages the concept of page-fault-based transient-execution replay [54] targeting AMD confidential VMs. PowerHooK enables power side-channel attacks on constant-time, hardware-accelerated implementations that lack data-dependent control flow or memory accesses, which are often employed as a software de-

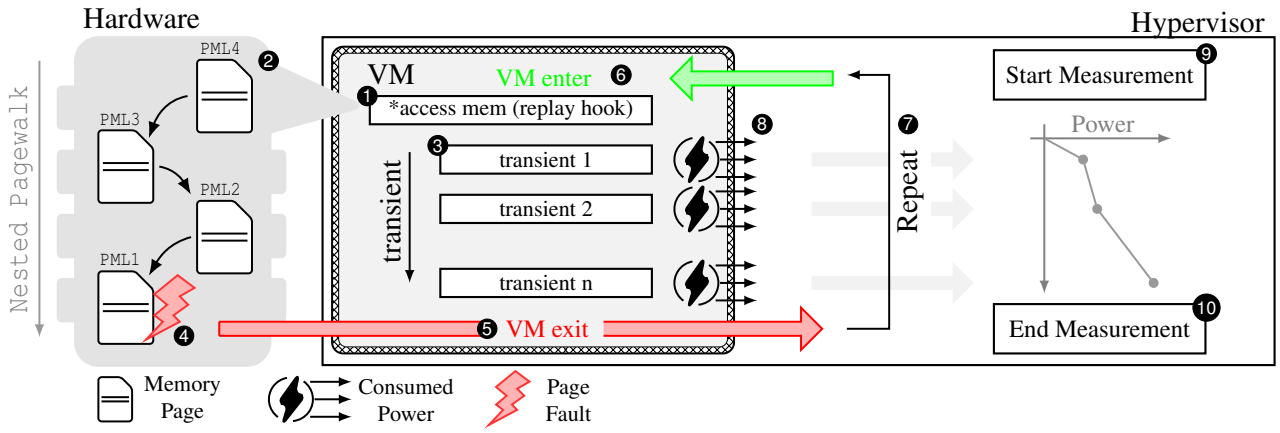

Figure 2: Conceptual attack setup of PowerHooK, the transient power side channel mounted from a malicious hypervisor leaking secrets of a victim VM. The malicious hypervisor repeats victim operations using a transient replay mechanism relying on page faults while taking power measurements using the RAPL software interface (or the Hertzbleed effect).

fense against microarchitectural side-channel attacks [7, 43]. In more detail, we show how a malicious hypervisor can deliberately mishandle a page fault triggered by a guest VM, enabling arbitrary replay of transient instructions following the faulting instruction. These arbitrary replays enable mounting software-based power side-channel attacks, leaking secrets processed by the victim VM in a single run, countering the interface's low resolution and significant system noise.

Figure 2 provides the core idea of a PowerHooK attack scenario in a concrete attack setting, i.e., on-demand paging, mapping a guest's memory pages only when required. The setting includes a guest running inside a VM, executing code and accessing a memory page that is not yet mapped ❶. The hardware, responsible for managing the memory resources, resolves the memory access by traversing the nested page-table walk ❷. Since resolving the invalid memory access takes several CPU cycles, the CPU transiently executes subsequent instructions following that memory access. We denote these transient instructions as *transient 1* to *transient n* ❸. Once the hardware recognizes the invalid memory access, a *page fault* ❹ occurs. This discards any results of the transient window, which are cached in the reorder buffer. Due to the restricted privileges of the VM, actions that require higher privileges (e.g., mapping pages) must be executed by a hypervisor, running outside these trusted boundaries. Thus, the pending fault subsequently triggers a `VMEXIT` ❺, transferring control to the hypervisor [8]. Following the `VMEXIT`, the hypervisor *should* map the missing page before returning the control back to the guest VM via a `VMENTER` call ❻.

Conversely, if a malicious hypervisor *does not map* the corresponding memory page before giving control back to the guest VM, another page fault is triggered, upon re-execution of the memory access by the VM ❶. This subsequently triggers the re-execution of the transient window, repeating the process ❶ to ❺. This replay mechanism ❼ can be triggered by a malicious hypervisor repeatedly, thus allowing arbitrary re-execution of the transient window.

From an architectural point of view, the guest's program counter does not advance. However, the repeated recomputations of the instructions in the transient window consume power ❽. A malicious hypervisor can capture these power differences via the RAPL power interface accessible outside the VM's boundaries, or by utilizing power-related signals (Hertzbleed effect [58]), i.e., the execution time differences of repeated transient replay. Thereby, one sample of a PowerHooK attack corresponds to the difference in power consumption before ❾ and after ❿ repeatedly invoking transient replay. Notably, without repeated execution of the transient window, the coarse resolution, and low sampling rate (∼1 ms) of the power interface, and high system noise hinder successful exploitation. The replay mechanism additionally enables an attacker to replay targeted subparts of an algorithm that processes secrets, enabling attacks on algorithms that are executed by the victim VM in a single run.

We present PowerHooK, an attack methodology utilizing transient replay gadgets to amplify power measurements. This allows the hypervisor to continuously invoke code sequences of a victim VM, thus enabling denoising of software-based power side-channel attacks on AMD confidential VMs.

5 Systematic Transient Leakage Analysis

This section systematically analyzes data-dependent power leakage by comparing leakage rates of PowerHooK transient replay across VM boundaries against leakage rates of architectural and speculative execution. Specifically, we characterize leakage rates observable via the RAPL power interface, and

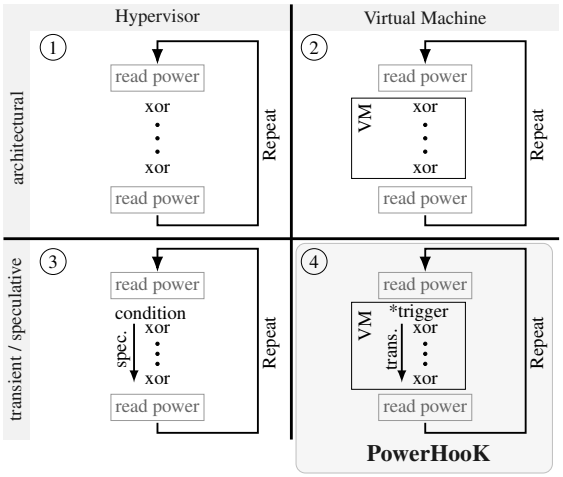

Figure 3: Schematic setup of the systematic leakage analysis of processing varying data operands with ① architecturally executed instructions in the hypervisor, ② architecturally executed instructions in a VM, ③ speculatively executed instructions in the hypervisor, and ④ transient instructions replayed via PowerHooK across the VM's boundaries.

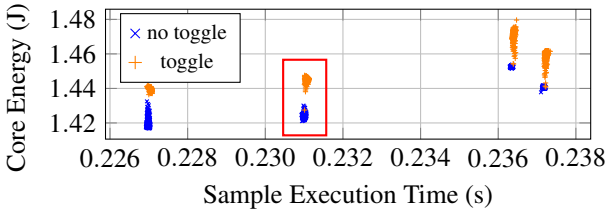

Figure 4: Scatterplot of sample execution time versus core energy consumption of experiment ① running on the AMD Ryzen 5 9600X. We use the distinct clusters containing the most samples in execution times to filter the observed energy consumption (highlighted in red).

the Hertzbleed leakage effect, measured via a timing instruction. We evaluate the leakage effect across different generations of AMD CPUs (i.e., Zen+, Zen 3, Zen 5), characterizing the impact of (i) resolving and recovering from architectural faults and (ii) the overhead of repeated VM enter and exit. Moreover, we analyze different CPU classes, ranging from server-grade to mobile-class CPUs, characterizing power and Hertzbleed-type leakage of our PowerHooK attack.

Experimental Setup. We evaluate power leakage in 4 distinct scenarios shown in Figure 3, namely, ① architectural execution in the hypervisor, ② architectural execution inside the guest VM, ③ speculative execution in the hypervisor, and ④ transient replay inside the guest VM via PowerHooK. Each scenario executes the same constant-cycle workload based on the `xor`-instruction, aligning repetition numbers to allow a direct comparison between the scenarios. We measure the power consumption of each scenario via the RAPL power interface and the execution time with a timing instruction quantifying the Hertzbleed leakage effect.

Scenario Description. Experiment ① executes the workload architecturally in the hypervisor, i.e., instructions execute and retire in program order, and serves as a baseline for our leakage analysis. Experiment ② executes the same workload architecturally inside a VM, gathering measurements from the hypervisor before entering and after exiting the VM, and determines leakage of instructions executed inside the VM across VM boundaries. Experiment ③ executes the workload speculatively in the hypervisor by triggering misspeculation with a SpectreRSB gadget [34] that maliciously fills the return stack buffer. This scenario quantifies speculative leakage, including noise of recovering from misspeculation. Experiment ④ executes the workload in a transient replay window inside the VM, triggered by the hypervisor, corresponding to our PowerHooK replay method (Section 4.2). We trigger the transient window with an invalid memory access. While hardware resolves the fault, the workload executes transiently until the fault is detected. This experiment quantifies leakage overhead from trigger address resolution, VM transitions, and fault recovery. We collect 5000 measurements (samples) from the hypervisor in each scenario.

To reduce system noise across all evaluations, we disable hyperthreading, turn off interrupts on the measurement core, set Periodic Directory Rinse (PDR) tuning to the lowest supported frequency in BIOS (where available), and disable unnecessary host services that could interfere with the measurements. For the Zen 5 EPYC server CPU, which provides several performance-tweaking options in BIOS, we set the performance profile to KVM-optimized.

Executed Workload. The executed workload is constant-cycle, operating exclusively on registers without memory interactions, and has no data-dependent timing or branching. The workload is designed to exhibit different power draw for two operand classes, with one consuming more power than the other. In more detail, we implement the workload with a 128-bit exclusive-or *vpxor* which takes two data operands. The first operand is set to 0 each invocation, and the second operand is all-zero or all-ones, controlling whether the input is flipped before writing the output. The result of each toggling operation is used as the input to the next *vpxor* instruction in a loop, thus repeatedly toggling (or leaving unchanged) the first operand. The workload operates on 7 distinct input registers initialized with identical values to amplify leakage, consuming all available execution engines and ports.

To ensure that the leakage rates of all four scenarios ① - ④ are comparable, each scenario executes an identical number of workload repetitions. We choose the number of repetitions such that the PowerHooK transient replay evaluation (scenario ④) takes approximately 2 s to obtain one sample. Note that the execution times of the other scenarios that do not

Table 1: The systematic leakage analysis of the RAPL *core-energy* domain (Signal-to-Noise Ratio (SNR), Pearson correlation r, and timer per sample t (s)) of instructions architecturally executed in the hypervisor ①, architecturally executed in a VM ②, speculative executed in the hypervisor ③, and transiently replayed using PowerHooK in the VM ④, on different AMD CPUs and categories. More intense cell colors highlight larger absolute values of SNR and execution time per sample.

ID	CPU	uarch	Class	Arch HV ①			Arch VM ②			Spec. ③			Transient ④			
				r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	
$\mathcal{A}$	AMD Ryzen 7 PRO 3700U	Zen+	M	0.333 ±0.063	0.125	0.122 $\pm10^{-3}$	0.333 ±0.063	0.125	0.118 $\pm10^{-3}$	0.216 ±0.050	0.048	0.053 $\pm10^{-3}$	0.175 ±0.068	0.032	1.993 $\pm10^{-2}$	PowerHooK
$\mathcal{B}$	AMD Ryzen 7 PRO 5850U	Zen 3	M	0.264 ±0.066	0.075	0.118 $\pm10^{-4}$	0.214 ±0.067	0.048	0.120 10^{-4}	– ±	0.003	0.028 $\pm10^{-4}$	0.082 ±0.069	0.007	1.852 $\pm10^{-3}$	
$\mathcal{C}$	AMD Ryzen 5 9600X	Zen 5	D	0.990 ±0.001	51.205	0.231 $\pm10^{-3}$	0.973 ±0.003	17.786	0.232 $\pm10^{-3}$	– ±	0.001	0.038 $\pm10^{-5}$	0.128 ±0.069	0.017	2.035 $\pm10^{-4}$	
$\mathcal{D}$	AMD Ryzen 7 PRO 5750G	Zen 3	D	0.559 ±0.050	0.455	0.123 $\pm10^{-4}$	0.181 ±0.066	0.033	0.125 10^{-4}	– ±	0.0001	0.030 $\pm10^{-4}$	0.075 ±0.069	0.006	1.948 $\pm10^{-4}$	
$\mathcal{E}$	AMD EPYC 7313P	Zen 3	S	0.116 ±0.068	0.014	0.116 $\pm10^{-4}$	0.284 ±0.063	0.088	0.118 $\pm10^{-3}$	– ±	0.000	0.027 $\pm10^{-4}$	0.092 ±0.069	0.009	1.925 $\pm10^{-3}$	
$\mathcal{F}$	AMD EPYC 9005	Zen 5	S	0.719 ±0.035	1.076	0.169 $\pm10^{-5}$	0.377 ±0.061	0.166	0.167 $\pm10^{-5}$	0.133 ±0.067	0.018	0.029 $\pm10^{-6}$	– ±	0.0003	2.000 $\pm10^{-4}$	

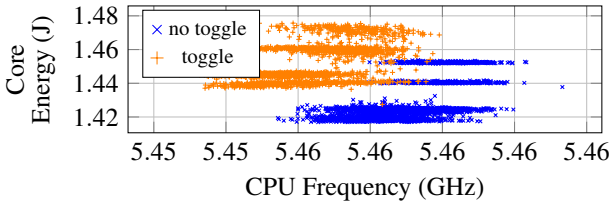

Figure 5: Scatterplot of CPU frequency vs core energy consumption of experiment ① on the AMD Ryzen 5 9600X. The scatter plot shows separable clusters between the two classes.

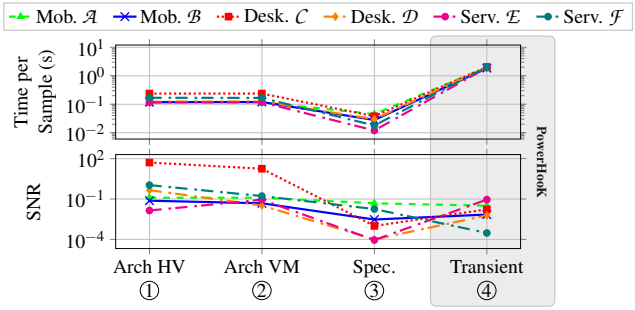

Figure 6: Sample execution times and Signal-to-Noise Ratios (SNR) for scenarios ① - ④ across all evaluated devices.

perform transient replay are lower, as they do not include the overhead of repeated VM transitions. Before and after each invocation of the workload, we collect RAPL power readings from the core and package domains, the core frequency and the execution time in clock cycles, thereby enabling the quantification of the Hertzbleed effect (see Appendix A).

Enhanced Signal Filtering. In our evaluation, we apply a generic filtering step to the collected per-core energy readings to ensure fair comparison between the different leakage scenarios, isolating relevant leakage signals. Figure 4 shows a scatter plot of the collected per-core power readings versus the sample execution time of scenario ① collected on the AMD Ryzen 5 9600X. The figure shows distinct clusters in the sample execution time, originating from Dynamic Voltage and Frequency Scaling (DVFS) effects. This is consistent with the results given in Figure 5, which also show clustering of the CPU frequency over the core energy consumption. Each cluster contains samples of both operand classes, which are clearly separable in the core energy domain. The absolute values of the core energy across the different clusters vary, thereby decreasing the leakage rate when considering the raw signal.

Note that the number of clusters (DVFS operating points, i.e., voltage-frequency pairs) varies across scenarios, workloads, and machines, hindering direct comparison. Thus, we filter the collected signal based on the cluster with the largest number of samples in execution time or core frequency (highlighted in red), selecting 800 samples for evaluation. This ensures consistency across scenarios and machines, without relying on more sophisticated signal post-processing.

Evaluation. We quantify the leakage strengths of the energy readings of the *core-power* domain by comparing the four experiments by computing (i) the Pearson correlation (r) of the measured traces with a binary power model $\mathcal{M} = \{0, 1\}$, (ii) the Signal-to-Noise Ratio (SNR), and (iii) the time per sample for each experiment. We provide the results in Table 1 and Figure 6, evaluating different AMD CPU generations and CPU classes. Additionally, we provide the experiment's leakage rates captured via the unprivileged accessible Hertzbleed leakage effect, providing results in Appendix A. For our statistical evaluations, we use energy readings obtained from

the per-core RAPL interface, filtering DVFS effects with a generic filtering approach.

Evaluation Results. The evaluations show similar leakage trends across AMD CPUs of different generations and classes (mobile, desktop, and server). Experiments ① and ② executing the workload architecturally show the most significant correlations r, which lie between 0.1 and 0.99. In comparison, experiment ③, executed speculatively, shows decreased correlations, with some devices yielding results that do not reach statistical significance. We attribute this to the overhead of triggering speculation and the early resolution of misspeculation before the workload completes, as indicated by the reduced sample execution time. Finally, the transient PowerHooK replay ④ yields correlations of 0.1 to 0.2, with only minor deviations between different CPU classes and generations. The SNR of the experiments follows a similar trend.

Figure 6 depicts the SNR and the execution time per sample across the different experiments. As experiment ④, transient PowerHooK replay, defines the number of repetitions such that each sample executes for roughly 2 s. Due to the reduced overhead, i.e., no repeated VM transitions and no recovering from architectural faults, architectural executions (① and ②) showcase decreased sample execution times by a factor of 10 to 20 compared to ④. Speculative execution ③ shows the fastest sample execution times. We suspect early detection of mis-speculation before the full workload finishes execution as a possible reason. While PowerHooK transient replay shows larger execution times and lower correlations than architectural execution, it allows for controlled instruction replay, enabling an attacker to extract secrets across VM boundaries.

We systematically show that PowerHooK exposes data-dependent power leakage, comparing leakage rates against architectural and speculative execution. While PowerHooK replay yields lower leakage rates than architectural execution due to guest-host transition noise, it enables arbitrary replay of victim code sequences, allowing an attacker to average out noise.

6 PowerHooK Leakage Characterization Across SEV Technologies

This section presents a systematic evaluation of PowerHooK leakage across different levels of AMD's confidential computing technology (AMD-V, SEV, SEV-ES, SEV-SNP) using a synthetic Correlation Power Analysis (CPA) benchmark, recovering a key byte of a first-round AES implementation accelerated with AES-NI. In an initial step, we quantify the execution time overhead that confidential computing adds to PowerHooK replay. Second, we perform an AES CPA, leaking an AES key byte processed by a confidential VM by a single AES-round using the AES-NI hardware-acceleration. In a subsequent frequency-based filtering step, we demonstrate how an attacker can filter the collected traces based on the CPU frequency or execution time to enable the extraction of secrets, i.e., cryptographic key material. Besides characterizing PowerHooK leakage via the RAPL power interface, we showcase in Appendix B, an AES key byte recovery exploiting the Hertzbleed effect with a timing instruction (`rdtscp`).

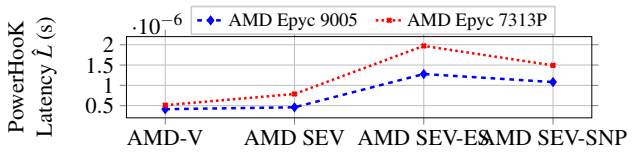

Figure 7: Per-PowerHooK replay latency $\hat{L}$ across confidential computing modes, averaged over $n = 1500$ samples, each sample with a t =1 s replay duration ($\hat{L} = t/N$; N = # of replays per sample) (std. dev below visual resolution; 10^{-8} to 10^{-10}s).

6.1 Replay Latency

As an initial step of our analysis, we quantify how different levels of virtualization (AMD-V, SEV, SEV-ES, and SEV-SNP) under identical system configuration (e.g., Transparent Huge Pages (THP) disabled in hypervisor), affect the PowerHooK replay execution time. Specifically, we measure PowerHooK execution times by replaying the PowerHooK gadget for approximately one second, estimating replay execution time using a timing instruction, i.e., `rdtscp`.

Figure 7 displays the average, per-PowerHooK replay execution time across 1500 samples on two AMD EPYC server-class CPUs (Zen 3 and Zen 5). In relation to AMD-V, enabling AMD SEV increases PowerHooK execution time by factors of 1.54 (Zen 3) and 1.11 (Zen 5). Enabling AMD SEV-ES increases average PowerHooK execution time by a factor of 3.85 (Zen 3) and 3.10 (Zen 5), which is in line with the additional state and register encryption performed upon guest entry and exit. Notably, enabling SEV-SNP introduces lower execution time overhead to the replay gadget than enabling SEV-ES. Compared to SEV-ES, SEV-SNP reduces per-replay execution time by 24% and 16% for Zen 3 and Zen 5, respectively. This reduction in replay execution time is also consistent when pinning the CPU frequency. Despite the additional RMP checks performed by SEV-SNP, the decrease in replay latency may partly be explained by RMP-caching, which optimizes repeated RMP checks [2, 10].

Enabling AMD SEV increases replay latency by only a small amount compared to AMD-V (1.5x / 1.1x) (Zen 3/Zen 5). In contrast, SEV-ES raises replay latency sharply (3.8x / 3.1x), likely due to the overhead of state and register encryption. Despite SEV-SNP's added integrity protection, results show a replay latency decrease by (24%/16%) compared to SEV-ES.

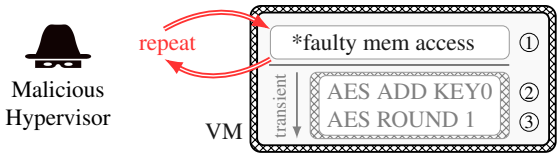

Figure 8: Schematic setup of the synthetic AES CPA attack benchmark: A malicious hypervisor transiently replays the first AES round executed inside a VM while measuring the CPU's energy consumption. Replay is triggered by the guest's repeated access to a not-yet mapped memory address, causing page faults that stall the instruction pointer and repeatedly reopen the subsequent transient window.

6.2 Synthetic AES CPA Benchmark

This section presents a systematic analysis that recovers a single AES-NI key byte from a guest VM using CPA with different levels of confidential computing enabled (AMD-V, SEV, SEV-ES, SEV-SNP). Our analysis aims to serve as a synthetic baseline leakage assessment, including only minimal additional operations in the PowerHooK replay window.

Experimental Setup. Figure 8 shows our schematic attack setup, which includes a malicious hypervisor with full system privileges alongside a guest VM running on the same physical host, e.g., an AMD server-class CPU capable of running confidential VMs. In our attack scenario, the victim VM first performs a memory access (Figure 8, ①), which serves as the replay hook for the PowerHooK attack benchmark. Following this replay hook, the victim executes a single AES-128 encryption round (Figure 8, ② and ③). The first AES encryption round is implemented using the constant-cycle `AESENC` instruction, which performs all sub-operations of a single AES round (Figure 8, ③): *SubBytes*, *ShiftRows*, *MixColumns*, and the subsequent *KeyAddition*. The initial key addition, executed before the AES-NI instruction, is implemented using a vectorized *xor* instruction (Figure 8, ②). This AES round implementation is consistent with state-of-the-art AES implementations in cryptographic libraries, such as OpenSSL [48]. Importantly, since there is no data dependency between the memory load and the subsequent AES round, the attacker is able to exploit the faulted memory access as a PowerHooK replay gadget (also observed in real-world cryptographic implementations; see Section 7). This enables replaying the AES round at will in the transient execution window, subsequently amplifying and denoising power leakage.

In the attack phase, the AES key is fixed. We collect power measurements, CPU frequency, and latency of the transient window. For each sample, we randomize the value of a single byte in a fixed position in the plaintext in a pseudorandom sequence, while keeping all other byte positions fixed to a constant value. To minimize system noise and to ensure consistent data collection, we inject the plaintext bytes directly into the victim's architectural register state before collecting a PowerHooK sample on AMD-V and AMD-SEV. Under AMD SEV-ES and SEV-SNP, direct register injection is not available. Therefore, we pass the bytes through the GHCB guest-hypervisor interface [4] by deliberately triggering an interrupt (e.g., an #VC or NMI) that induces an intercept, which we handle by a guest-side handler. Before each sample, we flush the VM's TLB. A sample then consists of 2^{18} transient replays of an AES round using selected plaintext and fixed key as input. We collect 30000 samples, each corresponding to a plaintext byte transiently replayed with PowerHooK. We conduct the experiment on an EPYC 7313P server CPU under four configurations: (i) an AMD-V VM, and confidential VMs with (ii) SEV, (iii) SEV-ES, and (iv) SEV-SNP enabled.

Evaluation. In the subsequent evaluation, we perform the CPA [11] to recover key bytes using the collected measurements and the known input plaintexts. Specifically, we perform a rank complexity analysis, characterizing attack times required to recover the correct key byte. Our analysis compares leakage rates across different levels of virtualization, characterizing the baseline for leakage speed.

Our AES CPA analysis utilizes the collected power measurements and known plaintext inputs obtained from performing the transient encryptions to retrieve the used key bytes. Specifically, we correlate the gathered power measurements to a power model representing a specific point in the AES algorithm. In traditional power-analysis attacks, a common target point inside the AES algorithm is the output of the non-linear sub-byte operation ($\mathrm{SBOX}(\mathrm{pt} \oplus \mathrm{key})$) [11]. The non-linear properties of the SBOX have the advantage that a one-bit difference at the input typically results in an average of 4 bit-flips at the output, which helps to distinguish key byte guesses during an attack. However, targeting this point in vectorized AES implementations based on `xor` and `AESENC`, fails to recover the correct key byte. The `AESENC` instruction computes the entire AES round internally, hiding implementation details such as the positions of internal register states, thereby eliminating common attack targets. Thus, to recover a key byte from the first AES round, we target the register state after the initial `xor`-based key addition. Although this point exposes less side-channel information due to the missing diffusion of the SBOX, it provides sufficient leakage, which we exploit in our attack.

In more detail, we use the *Hamming weight* (HW) as the power model, which corresponds to the number of bit flips of the initial key addition:

$$h_n^{(i)} = \mathrm{HW}\left(pt_n^{(i)} \oplus k^{(i)}\right) \tag{1}$$

where pt_n^i corresponds to plaintext byte i of the n-th plaintext input, and k^i to byte i of the first round key. This results in a hypothesis for the power consumption h_n^i corresponding to byte i during each encryption invocation n.

To recover a correct key byte k^i, we enumerate all possible key byte values ranging from 0 to 255, computing a hypothe-

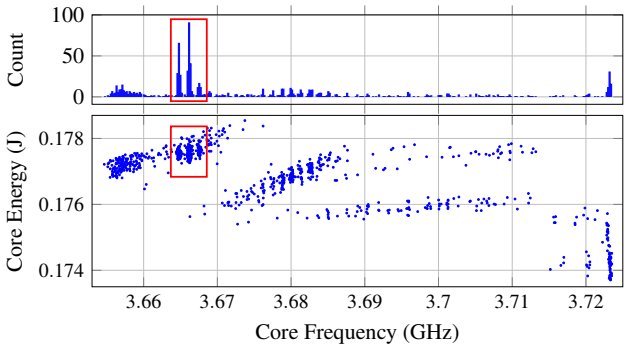

Figure 9: CPU frequency and core energy consumption of samples collected in the synthetic PowerHooK AES key recovery attack running on AMD-V used as a basis for signal filtering to enhance key byte leakage.

sis h_n^i for each value. We subsequently correlate the resulting hypothesis h_n^i for each hypothetical key byte value k^i to the power traces. Specifically, we use the Pearson correlation coefficient r^k to match the hypothesis using key byte k to the actual power measurement. Finally, we rank each key hypothesis's resulting correlation coefficients r^k according to the used, correct key byte. This results in a rank r^k for each key hypothesis between 0 and 255, with, e.g., $r^k = 0$ indicating the correctly guessed key k.

Enhanced Signal Filtering. Before performing the CPA rank analysis, we apply a generic filter rule to the collected power traces based on the observed CPU frequency, discarding samples that are not drawn from consistent univariate distributions. To visualize this process, we plot the first 1000 measurements of our synthetic AES analysis on AMD-V in Figure 9. The figure includes a histogram of the CPU frequency and a scatter plot highlighting the relationship between the CPU frequency and the collected core energy. The scatter plot shows that the core energy does not increase linearly with the CPU frequency, but instead forms distinct clusters at different absolute core energy levels. To account for these absolute value shifts, which are not directly comparable without advanced preprocessing, we restrict our CPA rank analysis to clusters with similar CPU frequency and execution times, that contain the highest number of samples. More precisely, in our generic filtering step, we use the collected CPU frequency and execution time of each sample and discard any sample that does not fall into the Gaussian distribution that includes the largest number of samples. In Figure 9, we frame the region of interest with a red box, from which we select the distribution with the most samples for later evaluations.

Key Byte Recovery. Figure 10 depicts the CPA rank complexity analysis of the filtered traces at different levels of confidential computing, including AMD-V, SEV, SEV-ES, and SEV-SNP. Since different levels of virtualization increase PowerHooK latency (see Section 6), the experiment's total execution time varies across different configurations, i.e., 0.7 h for AMD-V, 1.3 h for SEV, 4.3 h for SEV-ES, and 3.1 h for SEV-SNP. Each curve depicts the key byte ranks over an increasing number of measurements, which converges towards $r^k = 0$, indicating the correct key, and illustrates the number of samples required to recover the correct key byte.

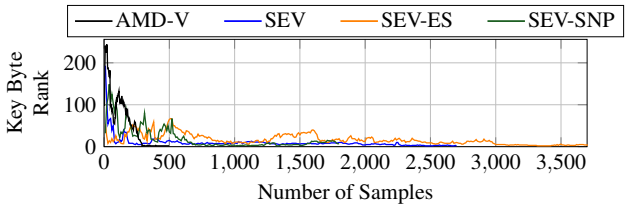

Figure 10: Key rank complexity analysis of a first-round AES attack, transiently replayed in a PowerHooK attack for different levels of AMD's virtualization technology.

Our results demonstrate that the key byte rank of a VM running AMD-V converges towards the correct guess after only considering 300 filtered samples, corresponding to 3 min of trace collection time. Similarly, with SEV enabled, the key byte rank drops below 20 after around 400 samples, fluctuates around that level and converges to the correct key byte after around 2450 samples, which corresponds to 0.44 h of trace collection. On SEV-ES, approximately 3430 samples after filtering are needed until convergence to the correct key byte, which requires roughly 3.23 h of trace collection, including removal of initial signal drift (1500 samples). Finally, SEV-SNP converges after considering 1320 samples, equivalent to 1.36 h of trace collection. The displayed rank complexity shows minor fluctuations in rank around zero. The key byte hypotheses resulting in these ranks differ from the correct key by a one-bit error due to the missing diffusion of attacking a point before the SBOX. Thus, an attacker could easily test for the correct key in the final step.

We demonstrate how a malicious hypervisor can leak AES key bytes from a guest VM using PowerHooK. We demonstrate leakage of a single AES key byte after analyzing 300 samples (∼3 min) under AMD-V, 2450 samples (∼0.44 h) under SEV, 3430 samples (∼3.23 h) with SEV-ES and 1320 samples (∼1.36 h) under SEV-SNP, consistent with our replay-latency estimate.

7 Real-World OpenSSL AES Key Recovery

This section presents a real-world case study of key-byte recovery on OpenSSL's CBC multi-block decryption [48], showing that PowerHooK gadgets are present and exploitable in real-world cryptographic implementations. We demonstrate that transiently replaying the start of each block decryption, within a single CBC decryption execution over multiple ciphertext blocks, enables a malicious hypervisor to extract key

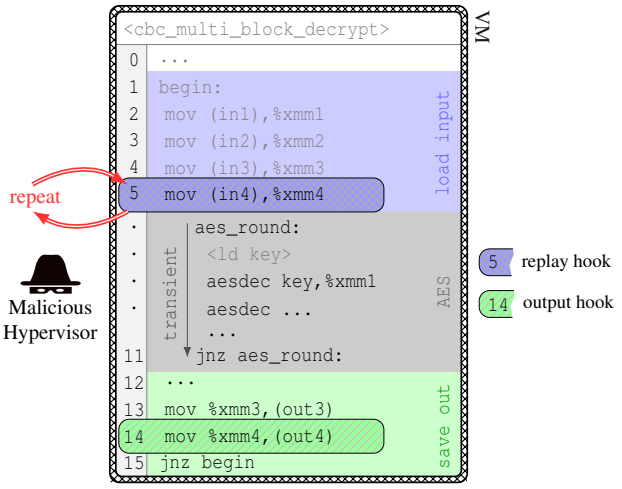

Figure 11: Schematic setup of the real-world AES CPA attack targeting `aesni_multi_cbc_decrypt` of OpenSSL. The victim VM decrypts multiple ciphertexts in parallel while the attacker induces repeated page faults on loads of ciphertext blocks (in Line 5). Faulting one input block opens a transient execution window during which other parallel blocks are decrypted, enabling first-round AES CPA. The attacker advances blockwise through the CBC decryption by alternately unmapping and remapping memory pages of the replay and output hook (the input and output memory transactions).

bytes from a victim VM running OpenSSL. By replaying targeted parts of the algorithm that process the secret key using PowerHooK *transient replay hooks*, an attacker can recover AES key bytes. The transient replay mechanism successfully counteracts the low sampling rate of the RAPL power interface and high background noise, enabling AES CPA attacks. Notably, without replay, the low sampling rate of the power interface ($\sim$1 ms) would cover multiple decryptions with different inputs, hindering successful exploitation.

In more detail, we assume a cloud worker that is running inside an AMD SEV-ES protected VM, which processes data using AES-CBC multi-block decryption (`aesni_multi_cbc_decrypt`), present in OpenSSL 1.0's AES-NI optimized implementation. The adversary (i.e., the malicious hypervisor) silently observes ciphertexts (e.g., from communication with a third party) and transiently replays the victim VM's execution while measuring power in a PowerHooK attack setting. Therefore, in our case study, the attacker knows the processed ciphertexts and transiently replays the code path that decrypts each ciphertext block. The attacker collects 50,000 PowerHooK samples (3.7 h trace collection time) corresponding to different processed ciphertext blocks taking roughly 0.266 s $\pm 10^{-4}$ of collection time per sample, and performs a subsequent CPA key byte recovery, compromising the confidentiality of the captured, encrypted data. Since SEV-SNP's added integrity protection does not primarily affect leakage observable via PowerHooK (see Section 6), and our attack targets the confidentiality assumptions rather than the integrity protection added by SEV-SNP, we conduct our case study on an AMD EPYC 7313P server CPU with AMD SEV-ES enabled. Our evaluation focuses on the leakage assessment of the concrete code snippet that is implemented in OpenSSL's performance-optimized assembly implementation. The decryption function is invoked by the victim inside the VM once (i.e., we call the decryption function directly, omitting execution of the full OpenSSL stack, targeting only relevant code paths to reduce application-level overhead); the hypervisor subsequently replays subparts of its execution using PowerHooK. We additionally assume that the attacker knows the target guest physical addresses used for the *replay hook*. In a full end-to-end scenario, an attacker could observe page fault patterns in a profiling phase, locating the addresses of the *replay-hook* as demonstrated by Liu et al. [36].

Experimental Setup. Figure 11 depicts the schematic setup of our PowerHooK attack targeting the implementation of AES-CBC multi-block decryption in OpenSSL. The example shows a simplified pseudocode version of the algorithm executed by the victim VM. The algorithm takes four encrypted messages as input and decrypts them in parallel using AES-CBC mode, subsequently writing the results into an output buffer. Moreover, the algorithm processes the input messages in blocks of 128 bits in a loop. In each iteration, starting at the label `begin`, a 128-bit block of each message is decrypted in parallel. Each loop iteration can be split into three main parts: reading four ciphertext blocks as input, applying the CBC decryption to each of the four inputs, and storing the decrypted result in the output buffers. Note that additional control logic was removed to simplify the shown pseudocode to its core functionality, highlighting only the important parts of the PowerHooK attack.

At the beginning of each iteration, OpenSSL loads four ciphertext blocks from memory into 128-bit `xmm` registers `xmm1`-`xmm4`. The input pointers `in1`–`in4` initially point to the four ciphertext buffers and are passed to the algorithm along with metadata, such as the block count, initialization vector, and output buffers. OpenSSL increments the pointers on each iteration to fetch the subsequent ciphertext blocks. Notably, the four memory buffers point to different memory regions, e.g., different memory pages. Since the victim's memory pages are mapped on demand, the first access to each buffer triggers a page fault. The attacker exploits these faults and the resulting `VMEXIT`s as a *replay hook* for PowerHooK.

In the conceptual setup shown in Figure 11, we depict this replay hook in Line 5. Out of the four input blocks, we choose the loading of the fourth and last ciphertext block as the PowerHooK replay gadget. This memory load triggers a transient replay window, during which the previously loaded ciphertexts are decrypted. As the previous ciphertexts are already architecturally loaded into the registers prior to the replay hook, this placement of the PowerHooK minimizes

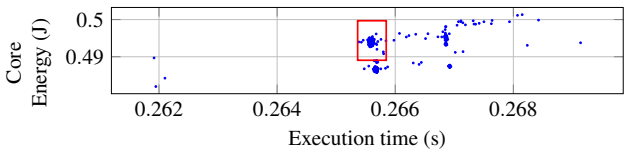

Figure 12: Scatterplot of execution time vs energy for the first 1000 samples of the real-world case study on the AMD EPYC 7313P. The red box marks the cluster used for evaluation.

additional noise from memory loads in the transient window. This enables the attacker to replay the decryption of the loaded ciphertexts in a transient window while collecting software-based power side-channel measurements, forming one sample for the attack. To perform a full CPA attack, the attacker has to collect multiple samples corresponding to the decryption of different ciphertexts. Thus, the victim has to advance to the decryption of the next ciphertext block. Therefore, the attacker maps the memory page used to trigger PowerHooK. This allows the victim to execute the decryption of the four ciphertexts architecturally, advancing the victim's instruction pointer to the point at which the decrypted plaintexts are written out to memory, as depicted in Line 14 of Figure 11. Since `out4` is also mapped on demand, writing the plaintext to this output buffer, referred to as *output hook*, causes a page fault, transferring control to the hypervisor. The hypervisor now marks the memory page used for the *replay hook* as *not present*, preparing the subsequent transient replay. Finally, the attacker maps the *output hook*, allowing the victim to continue the decryption of subsequent ciphertext blocks, enabling the attacker to collect the next PowerHooK sample. Note that the attacker has to flush the victim's TLB entries upon updating the victim's paging structures [8].

Evaluation. In the subsequent evaluation, we perform a CPA rank analysis, similar to the synthetic AES evaluation described in Section 6, using the known ciphertexts and collected measurements. Consistent with our initial leakage analysis, we consider only samples from the core-energy and execution time cluster containing the most samples, as shown in Figure 12, and remove approximately the top and bottom ten percent of execution times and core frequencies within this cluster. In Figure 13, we present the rank complexity analysis, which considers around 23,000 samples for the analysis. The plot depicts the key byte ranks of key bytes (KB) 11 to 15, which converge towards a low key byte rank with increasing samples, computed using Spearman correlation. Varying key bytes take different amounts of samples until reaching a low rank, which can be seen by key bytes 14 and 15, which reach the correct key after considering only 5,000 samples. Key byte 12 reaches a low key byte rank after around 10^4 samples, fluctuating near a rank of around 4 for the remaining measurement; however, similar to byte 13, it does not fully converge to the correct key. Due to the used Hamming Weight model in the attack, top key candidates differ from the correct

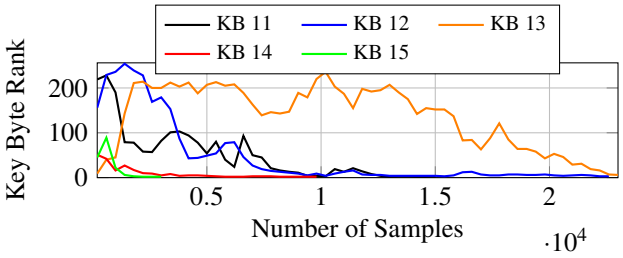

Figure 13: Key byte (KB) rank complexity analysis of different secret key bytes in the real-world OpenSSL AES-CBC case study, demonstrating leakage of multiple bytes.

key by only one bit. An attacker could use this rank-based information in a final step to test for the correct key bytes.

We successfully demonstrate the PowerHooK attack using OpenSSL's constant-time hardware-accelerated AES-CBC implementation. Thereby, we demonstrate the applicability of PowerHooK on real-world cloud workloads and demonstrate the availability of exploitable replay hook gadgets.

8 Discussion

In this section, we detail related work and discuss potential countermeasures against our PowerHooK attack.

8.1 Related Work

Lipp et al. [38] presented the PLATYPUS attack, demonstrating how an adversary can utilize the RAPL interface to mount software-based power side-channel attacks. PLATYPUS shows a variety of attack scenarios, including a KASLR break, RSA, and AES key recovery from Intel SGX. Here, the AES attack requires an interface allowing to trigger multiple AES-NI encryptions processing the same fixed input (i.e., plaintext and encryption key). The authors report that in their experiments, repeated measurement of the power consumption of a single AES-NI instruction via zero-stepping, introduces significant noise to the AES attack, which hindered them from successful exploitation. In contrast, PowerHooK targeting AMD SEV technologies utilizes transient replay gadgets instead of zero-stepping to denoise power measurements, which allows leaking constant-time hardware-accelerated AES-NI-based secrets.

PwrLeak [57] presented initial results targeting baseline AMD SEV VMs, where the guest's register state is not protected. To amplify the guest's power leakage, they present two `interpolators`: an emulation-based interpolator and an interrupt-based interpolator (i.e., zero-stepping). Due to the encrypted guest register state, their emulation-based interpolator does not apply to higher versions of AMD SEV (SEV-ES,

SEV-SNP). They evaluate their *zero-stepping* approach targeting a *non-constant-time* RSA implementation on AMD SEV. In contrast, PowerHooK targets constant-time hardware-accelerated AES-NI running inside SEV-ES and SEV-SNP utilizing transient replay gadgets. PwrLeak further notes that SEV-ES and SEV-SNP introduce substantial unstable noise due to additional protections (i.e., VMSA and RMP checks), which may require precise noise-canceling or a more powerful amplifier. Our work demonstrates that page-fault-based transient replay provides sufficient amplification to denoise power traces on SEV-ES and SEV-SNP.

Wang et al. [58, 59] and Liu et al. [40] presented Hertzbleed-type attacks, demonstrating how attackers can exploit CPU frequency variations (and execution time variations) as a proxy for power measurements. Prior work showcased attacks targeting AES on Intel, AMD [40], and Apple [15], remote attacks breaking SIKE, ECDSA and Classic McEliece [58, 59]. These attacks require the repeated invocation of the encryptions, while PowerHooK achieves instruction replay through transient execution. Taneja et al. [55] showcase the Hertzbleed effect on a broad range of devices, Dipta et al. demonstrated website fingerprinting and keystroke recovery [19] and container sandboxed fingerprinting [20], Yu et al. [63] attacks on lattice-based KEMs. Oberhuber et al. [47] demonstrated attacks using Android sensors, leaking AES key bytes by triggering an interface repeatedly. Draschbacher et al. [21] observed Android user activity via malicious chargers. Chun et al. [17] demonstrated a novel way of converting power- to timing variations using scheduling hints on modern Intel processors. Kogler et al. [33] introduced a generic approach to exploit Hamming distance leakage in shared buffers, which is also demonstrated by misusing prefetchers [28]. SpectremEM [42] targeted embedded devices, combining physical EM measurements with transient execution. Chowdhuryy et al. [16] demonstrate an instruction replay mechanism utilizing Intel TSX [30], leaking RSA keys from Intel SGX enclaves. Utilizing the Intel-specific TSX feature (deprecated and disabled by default on modern Intel CPUs [16]) for attacks targeting AMD VMs is not possible.

8.2 Possible Mitigations

Mitigating PowerHooK is not trivial, as it involves multiple microarchitectural mechanisms. Primarily, the attack requires access to a power interface, such as RAPL [29]. The patch introduced after the PLATYPUS attack [38], restricting RAPL to privileged users, is ineffective against malicious hypervisors with full system privileges. A possible mitigation is to restrict access to performance counters and power interfaces, fully or partially, while a confidential VM is running, which SEV-SNP guest policies can enforce [9, 22]. While this supposedly removes the attack surface for software-based power side-channel attacks, an attacker can still exploit unprivileged, power-related signals, e.g., the Hertzbleed effect [40, 58](as shown in Appendix A), which are hard to mitigate. Reducing timer precision, as commonly implemented in browsers [49], can increase the difficulty for Hertzbleed-style attacks. However, this does not mitigate the attack surface of PowerHooK, as an attacker can arbitrarily replay code sequences.

Constant-time programming is regarded as best practice against side channels targeting application-class processors [3]. However, software-based power side channels, e.g., replayed with PowerHooK, can break these algorithms. Collide+Power-type attacks [28, 33] exploit power consumption of data collisions in shared microarchitectural buffers, e.g., data caches. Since these attacks are difficult to mitigate, PowerHooK may amplify them by denoising their leakage.

Besides targeting the root cause of power leakage or corresponding interfaces exposing leakage, another possible mitigation of PowerHooK is to prevent transient replay. Since PowerHooK triggers repeated guest faults and identical `VMEXIT`s, hardware could bound consecutive same-reason exits, terminating the VM or notifying the guest as in AEX-Notify [18]. Another mitigation could be to insert a serializing `lfence` after potentially faulting memory loads, neutralizing transient replay gadgets as in LVI [12]. We verified this mitigation with our synthetic attack code from Section 6. However, fencing every memory load can incur significant overhead: Van Bulck et al. [12] report 2–19× slowdowns for `lfence`-based LVI mitigations targeting Intel SGX.

9 Conclusion

This paper presented PowerHooK, a novel attack that extracts VM secrets from AMD SEV technologies through software-based power side channels. We demonstrate how an attacker can enable repeated execution of victim operations to denoise power leakage of secrets processed in a single run in a confidential virtual machine. We systematically analyze the architectural and transient power leakage across different AMD CPUs and demonstrate our attack methodology by leaking AES-NI encryption key bytes through power analysis. Finally, we present a real-world AES key-byte recovery attack for individual key bytes on a VM-isolated cloud worker that executes OpenSSL's constant-time AES CBC implementation. Our evaluation showcases that PowerHooK effectively extracts secret information from confidential VMs with transient replay gadgets present in real-world applications.

Acknowledgements

We thank Daniel Gruss, Hannes Weissteiner, Sudheendra Raghav Neela and the anonymous reviewers for their help and valuable feedback. This project has received funding from the Austrian Research Promotion Agency (FFG) via the RESIST project (FFG grant number 915106) and the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85).

Ethical Considerations

This section discusses ethical considerations regarding our demonstrated attack targeting AMD CPUs.

Since our work proposes an attack targeting state-of-the-art AMD CPUs, we initiated a responsible disclosure process with the AMD Product Security team in late June 2025. AMD reviewed our report and prepared a security brief in response, scheduled for publication on August 10, 2026. Moreover, we performed all evaluations exclusively on CPUs under our control. Therefore, our evaluations *did not* target external production systems and only involved secret data under our control.

Open Science

We release an artifact package accompanying this paper at https://zenodo.org/records/20080425. The package contains the Python framework used for power analysis throughout our evaluation. It also includes proof-of-concept code for the `systematic analysis`, PowerHooK `latency analysis`, and synthetic `AES CPA analysis`, together with setup and execution instructions.

References

[1] AMD. AMD SEV-ES: Protecting VM Register State with SEV-ES. https://www.amd.com/de/developer/sev.html, 2017. Accessed: 2024-05-27.

[2] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More, 2020.

[3] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More". https://www.amd.com/de/developer/sev.html, 2020. Accessed: 2024-05-27.

[4] AMD. SEV-ES Guest-Hypervisor Communication Block Standardization, 2020.

[5] AMD. AMD Memory Encryption. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf, 2021. Accessed: 2024-05-27.

[6] AMD. AMD Secure Encrypted Virtualization (SEV). https://www.amd.com/de/developer/sev.html, 2021. Accessed: 2024-05-27.

[7] AMD. Execution Unit Scheduler Contention Side-Channel Vulnerability on AMD Processors, 2023.

[8] AMD. AMD64 Architecture Programmer's Manual, 2024.

[9] AMD. Advanced Micro Devices SEV Secure Nested Paging Firmware ABI Specification, 2025.

[10] Alexis Bagia, Vincent Quentin Ulitzsch, Daniël Trujillo, Mengyuan Li, Mengjia Yan, and Jean-Pierre Seifert. A close look at rmp entry caching and its security implications in sev-snp. In *Proceedings of the 14th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2025.

[11] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 16–29, 2004.

[12] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 54–72, 2020.

[13] Jo Van Bulck, Frank Piessens, and Raoul Strackx. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*, pages 4:1–4:6, 2017.

[14] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 249–266, 2019.

[15] Nikhil Chawla, Chen Liu, Abhishek Chakraborty, Igor Chervatyuk, Thais Moreira Hamasaki, Ke Sun, and Henrique Kawakami. Uncovering Software-Based Power Side-Channel Attacks on Apple M1/M2 Systems. In *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC 2024, San Francisco, CA, USA, June 23-27, 2024*, pages 305:1–305:6, 2024.

[16] Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. PowSpectre: Powering Up Speculation Attacks with TSX-based Replay. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*, 2024.

[17] Inwhan Chun, Isabella Siu, and Riccardo Paccagnella. Scheduled Disclosure: Turning Power Into Timing Without Frequency Scaling. In *2025 IEEE Symposium on Se-*

curity and Privacy (SP), pages 3340–3358. IEEE Computer Society, 2025.

[18] Scott Constable, Jo Van Bulck, Xiang Cheng, Yuan Xiao, Cedric Xing, Ilya Alexandrovich, Taesoo Kim, Frank Piessens, Mona Vij, and Mark Silberstein. AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4051–4068, 2023.

[19] Debopriya Roy Dipta and Berk Gülmezoglu. DF-SCA: Dynamic Frequency Side Channel Attacks are Practical. In *Annual Computer Security Applications Conference, ACSAC 2022, Austin, TX, USA, December 5-9, 2022*, pages 841–853, 2022.

[20] Debopriya Roy Dipta, Thore Tiemann, Berk Gülmezoglu, Eduard Marin, and Thomas Eisenbarth. Dynamic Frequency-Based Fingerprinting Attacks against Modern Sandbox Environments. In *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024*, pages 327–344, 2024.

[21] Florian Draschbacher, Lukas Maar, Mathias Oberhuber, and Stefan Mangard. ChoiceJacking: Compromising Mobile Devices through Malicious Chargers like a Decade ago. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 4363–4379, 2025.

[22] Stefan Gast, Hannes Weissteiner, Robin Leander Schröder, and Daniel Gruss. CounterSEVeillance: Performance-Counter Attacks on AMD SEV-SNP. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*, 2025.

[23] Philipp Giersfeld, Benedict Schlüter, and Shweta Shinde. BREAKFAST: Confused Deputy Attack on Infinity Fabric to Break AMD SEV-SNP. 2026.

[24] Google. Entry Sign: Zen and the Art of Microcode Hacking. https://bughunters.google.com/blog/5424842357473280/zen-and-the-art-of-microcode-hacking, 2025. Accessed: 2025-05-30.

[25] Jean-Claude Graf, Sandro Rüegge, Ali Hajiabadi, and Kaveh Razavi. VMSCAPE: Exposing and Exploiting Incomplete Branch Predictor Isolation in Cloud Environments. In *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

[26] Roberto Guanciale, Nicolae Paladi, and Arash Vahidi. SoK: Confidential Quartet - Comparison of Platforms for Virtualization-Based Confidential Computing. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED), Storrs, CT, USA, September 26-27, 2022*, pages 109–120, 2022.

[27] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60, 2008.

[28] Lorenz Hetterich, Fabian Thomas, Lukas Gerlach, Ruiyi Zhang, Nils Bernsdorf, Eduard Ebert, and Michael Schwarz. ShadowLoad: Injecting State into Hardware Prefetchers. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2025, Rotterdam, Netherlands, 30 March 2025 - 3 April 2025*, pages 1060–1075, 2025.

[29] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide, 2024.

[30] Intel. Intel® Transactional Synchronization Extensions (Intel® TSX) Memory and Performance Monitoring Update for Intel® Processors, 2024.

[31] 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 *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19, 2019.

[32] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[33] Andreas Kogler, Jonas Juffinger, Lukas Giner, Lukas Gerlach, Martin Schwarzl, Michael Schwarz, Daniel Gruss, and Stefan Mangard. Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 7285–7302, 2023.

[34] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael B. Abu-Ghazaleh. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *12th USENIX Workshop on Offensive Technologies, WOOT 2018, Baltimore, MD, USA, August 13-14, 2018*, 2018.

[35] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 337–351, 2022.

[36] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting Unprotected I/O Operations in AMD's Secure Encrypted Virtualization. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1257–1272, 2019.

[37] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 717–732, 2021.

[38] Moritz Lipp, Andreas Kogler, David F. Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 355–371, 2021.

[39] 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 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990, 2018.

[40] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. Frequency Throttling Side-Channel Attack. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1977–1991, 2022.

[41] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. 2007.

[42] Jesse De Meulemeester, Antoon Purnal, Lennert Wouters, Arthur Beckers, and Ingrid Verbauwhede. SpectrEM: Exploiting Electromagnetic Emanations During Transient Execution. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6293–6310, 2023.

[43] Yuanqing Miao, Mahmut Taylan Kandemir, Danfeng Zhang, Yingtian Zhang, Gang Tan, and Dinghao Wu. Hardware Support for Constant-Time Programming. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023, Toronto, ON, Canada, 28 October 2023 - 1 November 2023*, pages 856–870, 2023.

[44] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX. *Proc. ACM Meas. Anal. Comput. Syst.*, 8:36:1–36:42, 2024.

[45] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. SEVered: Subverting AMD's Virtual Machine Encryption. In *Proceedings of the 11th European Workshop on Systems Security, EuroSec@EuroSys 2018, Porto, Portugal, April 23, 2018*, pages 1:1–1:6, 2018.

[46] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. SEVerity: Code Injection Attacks against Encrypted Virtual Machines. In *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*, pages 444–455, 2021.

[47] Mathias Oberhuber, Martin Unterguggenberger, Lukas Maar, Andreas Kogler, and Stefan Mangard. Power-Related Side-Channel Attacks using the Android Sensor Framework. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*, 2025.

[48] OpenSSL. OpenSSL: The Open Source toolkit for SSL/TLS, 2025.

[49] Thomas Rokicki, Clémentine Maurice, and Pierre Laperdrix. SoK: In Search of Lost Time: A Review of JavaScript Timers in Browsers. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 472–486, 2021.

[50] Benedict Schlüter and Shweta Shinde. RMPocalypse: How a Catch-22 Breaks AMD SEV-SNP. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 3840–3854, 2025.

[51] Benedict Schlüter, Supraja Sridhara, Andrin Bertschi, and Shweta Shinde. WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 4220–4238, 2024.

[52] Benedict Schlüter, Supraja Sridhara, Mark Kuhne, Andrin Bertschi, and Shweta Shinde. HECKLER: Breaking Confidential VMs with Malicious Interrupts. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, 2024.

[53] Benedict Schlüter, Christoph Wech, and Shweta Shinde. Fabricked: Misconfiguring Infinity Fabric to Break AMD SEV-SNP. In *USENIX Security*, 2026.

[54] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. MicroScope: enabling microarchitectural replay attacks. In *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 318–331, 2019.

[55] Hritvik Taneja, Jason Kim, Jie Jeff Xu, Stephan van Schaik, Daniel Genkin, and Yuval Yarom. Hot Pixels: Frequency, Power, and Temperature Attacks on GPUs and Arm SoCs. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6275–6292, 2023.

[56] Robert M Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of research and Development*, 11(1):25–33, 1967.

[57] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 20th International Conference, DIMVA 2023, Hamburg, Germany, July 12-14, 2023, Proceedings*, pages 46–66, 2023.

[58] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 679–697, 2022.

[59] Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher W. Fletcher, David Kohlbrenner, and Hovav Shacham. DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 2306–2320, 2023.

[60] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 73–85, 2019.

[61] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. SEVurity: No Security Without Integrity : Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1483–1496, 2020.

[62] Luca Wilke, Jan Wichelmann, Anja Rabich, and Thomas Eisenbarth. SEV-Step A Single-Stepping Framework for AMD-SEV. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024:180–206, 2024.

[63] Tianrun Yu, Chi Cheng, Zilong Yang, Yingchen Wang, Yanbin Pan, and Jian Weng. Hints from Hertz: Dynamic Frequency Scaling Side-Channel Analysis of Number Theoretic Transform in Lattice-Based KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024:200–223, 2024.

[64] Ruiyi Zhang, Lukas Gerlach, Daniel Weber, Lorenz Hetterich, Youheng Lü, Andreas Kogler, and Michael Schwarz. CacheWarp: Software-based Fault Injection using Selective State Reset. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, 2024.

[65] Ruiyi Zhang, Tristan Hornetz, Daniel Weber, Fabian Thomas, and Michael Schwarz. StackWarp: Breaking AMD SEV-SNP Integrity via Deterministic Stack-Pointer Manipulation through the CPU's Stack Engine. In *USENIX Security*, 2026.

A Hertzbleed: Systematic Leakage Analysis

This section describes additional evaluations regarding the systematic analysis characterizing transient power leakage. Specifically, we demonstrate that PowerHooK-induced leakage is also observable via unprivileged power-related signals, such as the Hertzbleed effect, serving as a proxy for a power interface. Our evaluation reuses the traces collected in Section 5 and compares Hertzbleed-type data-dependent leakage of the four evaluated settings: architecturally executed instructions in the hypervisor and inside the virtual machine, speculative-executed instructions, and instructions replayed using PowerHooK. To capture the Hertzbleed frequency-throttling effect, we use the coarse-grained unprivileged timing instruction `rdtscp` to capture the execution time of each sample. The execution time variations of each sample serve as a proxy for a power interface due to the data-dependent frequency variations caused by the Hertzbleed effect.

Evaluation. Table 2 gives the results of the systematic Hertzbleed analysis. The given correlations and SNRs demonstrate, on average, lower absolute values than the measurements of the RAPL power interface. Similar to the evaluation

Table 2: The systematic leakage analysis (Signal-to-Noise Ratio (SNR), Pearson correlation r, and timer per sample t (s)) of the Hertzbleed effect measured via the unprivileged `rdtscp` instruction, of instructions architecturally executed in the hypervisor ①, architecturally executed in a VM ②, speculative executed in the hypervisor ③, and transiently replayed in the VM ④, on different AMD CPUs and categories. More intense cell colors highlight larger absolute values of SNR and execution time per sample.

ID	CPU	uarch	Class	Arch HV ①			Arch VM ②			Spec. ③			Transient ④			
				r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	r ·1	SNR ·1	t s	
$\mathcal{A}$	AMD Ryzen 7 PRO 3700U	Zen+	M	0.3776 ±0.060	0.166	0.122 $\pm10^{-3}$	– ±	0.001	0.118 $\pm10^{-3}$	0.236 ±0.066	0.059	0.053 $\pm10^{-3}$	0.082 ±0.069	0.007	1.993 $\pm10^{-2}$	PowerHooK
$\mathcal{B}$	AMD Ryzen 7 PRO 5850U	Zen 3	M	0.462 ±0.056	0.272	0.118 $\pm10^{-4}$	0.446 ±0.057	0.000	0.120 10^{-4}	– ±	0.000	0.028 $\pm10^{-4}$	– ±	0.002	1.852 $\pm10^{-3}$	
$\mathcal{C}$	AMD Ryzen 5 9600X	Zen 5	D	0.887 ±0.015	3.690	0.238 $\pm10^{-3}$	0.597 ±0.044	0.554	0.232 $\pm10^{-3}$	– ±	0.001	0.038 $\pm10^{-5}$	0.112 ±0.069	0.013	2.035 $\pm10^{-4}$	
$\mathcal{D}$	AMD Ryzen 7 PRO 5750G	Zen 3	D	0.163 ±0.068	0.027	0.123 $\pm10^{-4}$	– ±	0.000	0.125 10^{-4}	– ±	0.001	0.030 $\pm10^{-4}$	0.086 ±0.069	0.007	1.948 $\pm10^{-4}$	
$\mathcal{E}$	AMD EPYC 7313P	Zen 3	S	0.397 ±0.060	0.187	0.116 $\pm10^{-4}$	0.120 ±0.068	0.015	0.118 $\pm10^{-3}$	– ±	0.001	0.027 $\pm10^{-4}$	– ±	0.001	1.925 $\pm10^{-3}$	
$\mathcal{F}$	AMD EPYC 9005	Zen 5	S	– ±	0.0005	0.169 $\pm10^{-5}$	– ±	0.000	0.167 $\pm10^{-5}$	– ±	0.0002	0.029 $\pm10^{-6}$	– ±	0.0002	2.000 $\pm10^{-4}$	

using the RAPL interface, speculative execution, i.e., scenario ③ does not show significant correlation on all, except the AMD Ryzen 7 PRO 5850U mobile CPU. Upon considering PowerHooK replay (scenario ④), one mobile and server CPU do not show statistically significant correlation concerning the Hertzbleed effect.

Our evaluation demonstrates that PowerHooK is explainable via the Hertzbleed effect by reading an unprivileged, low-resolution timing instruction such as `rdtscp`.

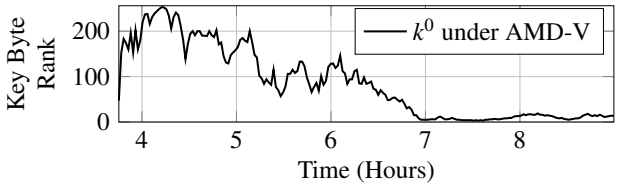

Figure 14: Key rank complexity analysis of a first-round AES attack, transiently replayed in a PowerHooK attack on an AMD Ryzen 7 PRO 5750G running AMD-V. The attacker exploits the Hertzbleed effect by measuring the execution time variations of a replay hook that repeatedly triggers transient encryptions. Execution times are gathered via the low-resolution unprivileged timing instruction `rdtscp`.

B Hertzbleed: AES CPA Rank Complexity Analysis

This section describes additional evaluations of the synthetic AES key byte recovery of Section 6. Specifically, we demonstrate how unprivileged timing instructions, i.e., the `rdtscp` instruction, can be utilized by an attacker as a proxy for a power interface to leak AES key bytes processed by the victim VM.

Experimental Setup. The experimental setup is similar to Section 6. We execute the synthetic AES key byte recovery on the AMD Ryzen 7 PRO 5750G running AMD-V and replay one AES round in the transient domain, changing one input plaintext byte at a fixed byte position. We collect 5000 samples in an PowerHooK setting, each replayed for approximately 9 s, processing different plaintext bytes.

Evaluation. In the subsequent evaluation step, we utilize the measured execution time, gathered with the `rdtscp` timing instruction of each sample and the known processed plaintext bytes performing an AES CPA key byte recovery. After applying our generic execution-time-based filtering, we remove slow signal drift (e.g., from core temperature variations) by subtracting a running average with a 72 s window from the collected trace and discarding the initial 1500 samples, which show substantial drift. Figure 14 depicts the rank complexity plot, which converges toward a low key byte rank after roughly 7 hours, fluctuating around low key byte ranks.

Our evaluation demonstrates that the unprivileged `rdtscp` instruction can be utilized as a power-related proxy for a power interface in a PowerHooK attack setting.