INCOGNITOS: A Practical Unikernel Design for Full-System Obfuscation in Confidential Virtual Machines

Kha Dinh Duy, Jaeyoon Kim, Hajeong Lim, and Hojoon Lee*
Department of Computer Science and Engineering,
Sungkyunkwan University

***Abstract*—Recent works have repeatedly proven the practicality of side-channel attacks in undermining the confidentiality guarantees of Trusted Execution Environments such as Intel SGX. Meanwhile, the trusted execution in the cloud is witnessing a trend shift towards confidential virtual machines (CVMs). Unfortunately, several side-channel attacks have survived the shift and are feasible even for CVMs, along with the new attacks discovered on the CVM architectures. Previous works have explored defensive measures for securing userspace enclaves (i.e., Intel SGX) against side-channel attacks. However, the design space for a CVM-based obfuscation execution engine is largely unexplored.**

This paper proposes a unikernel design named INCOGNITOS to provide full-system obfuscation for CVM-based cloud workloads. INCOGNITOS fully embraces unikernel principles such as minimized TCB and direct hardware access to render full-system obfuscation feasible. INCOGNITOS retrofits two key OS components, the scheduler and memory management, to implement a novel adaptive obfuscation scheme. INCOGNITOS's scheduling is designed to be self-sovereign from the timer interrupts from the untrusted hypervisor with its synchronous tick delivery. This allows INCOGNITOS to reliably monitor the frequency of the hypervisor's possession of execution control (i.e., VMExits) and adjust the frequency of memory rerandomization performed by the paging subsystem, which transparently performs memory rerandomization through direct MMU access. The resulting INCOGNITOS design makes a case for a self-obfuscating unikernel as a secure CVM deployment strategy while further advancing the obfuscation technique compared to previous works. Evaluation results demonstrate INCOGNITOS's resilience against CVM attacks and show that its adaptive obfuscation scheme enables practical performance for real-world programs.

1. Introduction

Ensuring the confidentiality of computation in untrusted environments such as the cloud remains a daunting challenge even with the mature hardware-assisted *TEE (Trusted Execution Environment)* architectures [1], [2], [3]. A plethora of attacks have demonstrated their effectiveness in undermining the confidentiality of the userspace enclaves (e.g., Intel SGX) [4], [5], [6], [7], [8], [9], [10], [11], [12].

**Corresponding author*

Shift towards CVMs. Meanwhile, a shift from process-based enclaves [13], [14], [15] such as Intel SGX towards *Confidential VMs (CVMs)* is anticipated with the current and upcoming hardware extensions. Intel's *Trust Domain Extensions (TDX)* [3], AMD's *Secure Encrypted Virtualization (SEV)* [2], [16], and ARM's *Confidential Compute Architecture (CCA)* [17] all adopt the virtual machine construct as TEE boundary. This means the OS kernel now resides within the trust boundary, unlike the previous userspace enclave design of Intel SGX [1].

Persisting side-channel threats in CVMs. Unfortunately, side-channel attacks continue to pose a significant threat to confidential computation even in the era of CVMs. Controlled channel attacks that leverage *Nested Page Faults (NPFs)* on the CVMs that can be observed in the untrusted hypervisor [8] to monitor page-granular memory access patterns have been an essential component of exploits on AMD-SEV [18], [19], [20], [21], [22], [23]. Recent research also showed that certain architectural and microarchitectural side-channels are feasible on CVMs. For instance, last-level cache attacks were shown to be feasible in AMD SEV(-SNP) [24], despite the per-ASID cache line isolation in place for virtual machines [25]. Also, Ciphertext side-channel attacks [18], [21] were discovered, which exploit the weak AES-XEX encryption mode used for CVM memory encryption that uses the *Host Physical Address (hPA)* as a tweak.

Obfuscated execution engines. Many previous works have proposed obfuscated execution engines for SGX to render the enclave execution more resilient against side-channel attacks through compiler transformations [26], [27], [28], [29]. Among them, Klotski [29] sought practicality through periodic mini-page-granular (2KB) memory layout rerandomization. The approach focuses on mitigating the profiling stage of the side-channel secret extraction attack in which the sensitive code and data locations are pinpointed. Klotski seeks to be cost-effective, given that the fine-grained data extraction attacks can only proceed when the profiling stage is successful.

Termination on exit frequency threshold. Detecting potential side-channel attack attempts through *exit detection* has been another pragmatic approach in mitigating side-channel attacks against SGX enclaves [30], [31], [32], [33]. Observing that most known practical side-channel attacks require frequent *Asynchronous Enclave Exit (AEX)* to acquire necessary temporal resolution on the channel, Varys [30] sets a threshold for AEX rate and terminates the enclave if the observed exit rate exceeds the threshold.

Limitations of existing defenses. We point out that the policies mentioned above for pragmatic defense against side channels have limitations. The fixed-rate periodic rerandomization of previous work [29] is inefficient, as the costly rerandomization is performed constantly regardless of ongoing attacks, especially when the exit frequency is a plausible measure for detecting ongoing attacks. Moreover, according to our empirical analysis, the termination policy on high exit frequency is *not robust*, at least when applied to SEV-SNP CVMs. As such, such policies may experience false positives in benign workloads while carrying the risk of missing low exit rate attacks. As we will show through our paper, a design space exists for adaptive and robust obfuscation that can reliably protect trusted execution workloads.

Towards obfuscation of CVMs. Besides the efficiency (fixed-rate) and robustness (fixed threshold) limitations regarding the policies of obfuscation for SGX, the accumulated research faces a shift towards CVMs [34], [35], [36], [37], [38]. To our knowledge, no previous works have discussed full-system obfuscation for a virtual machine. The shift implicates an incomparably larger execution unit to be obfuscated that includes an OS kernel. At the same time, CVMs open up a new design space for obfuscation engines with the highest degree of control (i.e., Ring 0) over the hardware.

Our proposal. This paper introduces **INCOGNITOS**, a novel unikernel design that safeguards unmodified programs with the efficient rate-adaptive obfuscation scheme for SEV-SNP CVMs. Moreover, as an operating system, INCOGNITOS is binary-compatible; it can run unmodified workloads compiled for the Unikraft unikernel. The minimal memory footprint and complexity of unikernels render full-system obfuscation feasible, including the memory regions of the kernel itself.

INCOGNITOS efficiently retrofits the OS kernel subsystems for constant system rerandomization by exploiting the unique characteristics of unikernel, such as the user-kernel homogeneity and direct hardware access [39], [40]. To maintain the system's resilience against side-channel attacks, INCOGNITOS's *scheduling subsystem* constantly monitors the temporal resolution of the untrusted host on the system and instructs the *paging subsystem* to rerandomize system memory reactively.

INCOGNITOS also makes substantial advancements in system memory obfuscation strategy with its novel rate-adaptive rerandomization scheme. INCOGNITOS implements software synchronous timer ticks to break the dependency on the untrustworthy hypervisor timer interrupts. At each synchronous tick, INCOGNITOS scheduler measures the occurrence of VMExit and executed instructions, or *VMExit per executed instructions*, which directly represents the adversary's temporal resolution. Based on the *measured* VMExit frequency that can be used to alert potential side-channel attacks, INCOGNITOS scheduler commands its paging subsystem to regulate its rerandomization rate. INCOGNITOS's design must overcome the challenges in designing a robust VMExit frequency measurement system *inside* the CVM.

INCOGNITOS's paging subsystem is designed to perform full-system rerandomization, including user/kernel pages and the page tables. While applying rerandomization of the currently active regions of CVM memory, INCOGNITOS's paging coherently integrates an ORAM-managed page pool that hides traces of page-ins and page-outs.

We also comprehensively evaluated INCOGNITOS to show its security and performance feasibility. Security evaluations against real attacks highlight the robustness of INCOGNITOS's adaptive obfuscation and its resilience against attacks. The microbenchmark and real-world application benchmarks with NGINX and Redis demonstrate the practicality of INCOGNITOS in terms of performance and also its ability to support unmodified programs.

Lastly, we summarize the contributions of this paper as follows:

- We propose a unikernel-based design for an efficient obfuscation execution engine that achieves transparent and full-system obfuscation.
- We propose a novel rate-adaptive obfuscation scheme to address the inefficiency of fixed-rate rerandomization and robustness issues of threshold-based termination schemes.
- We performed evaluations using microbenchmarks and real-world workloads to show its effectiveness.
- We open source INCOGNITOS and all implementations involved in its evaluation for future research and adoption[1].

2. Background and Related work

2.1. AMD SEV

Secure Encrypted Virtualization (SEV) isolates the execution of *Confidential VMs (CVMs)* from the untrusted hypervisor through transparent memory encryption and access control. *SEV-Secure Nested Paging (SEV-SNP)* is the newest iteration of SEV that encrypts register states in *VM Save Area (VMSA)* and addresses mapping integrity through a hardware-maintained *Reverse Mapping Table (RMP)*.

VMSA update on VMExits. When a VMExit transitions from CVM to host execution, the processor automatically updates the VMSA with exit information. Specifically, the `vmsa.exit_code` field is set as the exit reason for the current VMexit, e.g., 0x400 for *Nested Page Fault (NPF)*. This recording is unavoidable as it takes place at the hardware level without the intervention of the hypervisor [16] and is non-forgeable as the VMSA must be mapped as a private CVM page that is automatically encrypted.

Mapping integrity with RMP. SEV-SNP provides mapping integrity through the RMP, which contains integrity-protected mapping information such as the page's state

1. Available at https://github.com/sslab-skku/incognitos

(encrypted or non-encrypted) and mapping sizes. On any mapping changes, the mapping information in the RMP is invalidated; the CVM must *validate* this update with the `pvalidate` instruction. The RMP is consulted by the CPU on every memory access, which allows a CVM to detect illegal changes to the mappings.

2.2. Side-channel threats in CVMs

Side-channel attacks that undermine the confidentiality of trusted executions have long plagued Intel SGX, and many such threats continue in CVMs.

NPF Controlled-channel attacks. Adversaries who control the untrusted hypervisor can leverage its control over the *Nested Page Table (NPT)* to strategically trigger NPF on CVM pages by marking a page *non-present*. When the victim CVM accesses such pages, the CPU generates a `#NPF`, with the corresponding faulting address recorded in the *VM Control Block (VMCB)*. In the current SEV-SNP [16], the processor notifies the hypervisor of (1) the faulting *Guest Physical Frame Number (GPFN)* (e.g., `0xff...ab000`) through `VMCB.EXITINFO2` and (2) the cause of the fault, along with other information [16] through the `VMCB.EXITCODE` upon the NPF occurrence. These so-called NPF controlled-channel attacks are adaptations of the page fault-based attacks against Intel SGX [8], [9]. Through the attack, the adversary can trace instruction fetches and data accesses at page granularity. In many previous SEV(-SNP) attacks, NPF attacks have been employed as a CVM memory profiling primitive for pinpointing attack targets (§8.1).

Fine-grained side-channels. Fine-grained side-channel attacks represent attack vectors that leak memory access patterns at a resolution finer than the page size. These attacks typically exploit microarchitectural leakages (e.g., cache [4], [5], [9] and TLB [41]) or architectural leakages such the encrypted memory access patterns [42], and ciphertext updates [18], [21]. Fine-grained side-channel attack vectors have been extensively studied in the context of Intel SGX [4], [5], [6], [7], [9], [10], [11], [12], and certain attacks remain feasible against SEV-SNP CVMs. For instance, recent research highlights the exploitation of L2 cache Prime+Probe and ciphertext side-channel attacks to extract sensitive information from SEV CVMs [18], [21], [24]. Ciphertext side-channel attacks [18], [21], [43] are specific to AMD SEV(-SNP). They exploit the weak AES-XEX cipher used for CVM memory encryption, which consistently produces the same ciphertext for identical plaintext written to the same physical address. In combination with other primitives, a previous work [21] demonstrated a case of full RSA key leak.

Untrustworthy interrupts and temporal resolution. A SEV-SNP CVM currently has no option but to rely solely on the untrusted hypervisor to deliver *virtualized* interrupts, which may be delayed or completely skipped. Also, the hypervisor may leverage the precise hardware interrupt timer (e.g., APIC) to seize the victim's CVM execution for itself at any moment and at an arbitrary frequency. This untrustworthy interrupt delivery relation has been exploited as a primitive to boost the temporal resolution of the fine-grained side channels. Since the fine-grained side channels must target very specific addresses and are often noisy due to the ongoing CVM computations, *single-stepping* technique [6], [24], [44] was often used to acquire temporal resolution of a single instruction.

2.3. Previous mitigation approaches

While side-channel mitigations are not explored in the context of CVM, several works have strived to protect Intel SGX workloads from side-channel attacks.

Detection of side-channel attacks. Many works propose the *detection* of side-channel attacks through their observable side-effects, such as significantly long delays in program execution [32] or exceedingly high rates of enclave exits [30], [31]. T-SGX [31] and SGX-LAPD [33] propose terminating enclave execution if a configured number of enclave exits is detected. T-SGX retrofits the Intel-specific TSX feature to detect page faults during sensitive computation. SGX-LAPD proactively accesses memory pages before use and checks for enclave exits. Unfortunately, these approaches overly restrict the OS page management capabilities by preventing traditional demand paging.

Instead of forbidding all enclave exits, Varys [30] advocated periodically sampling the enclave's exit rate and terminating based on a set threshold.

Obfuscated execution engines. A line of work sought after an ideal oblivious execution model in which ORAM manages all code and data, and their execution and accesses are always performed through cache line-size *scratchpads* [26], [27]. Notably, Obfuscuro [26] provides cache-line-level obfuscated execution through ORAM. Obelix [27] extends upon the ideas of Obfuscuro to also grant protection against ciphertext attacks and resistance against microarchitectural instruction profiling attacks. However, such approaches accompany multiple magnitudes lower runtime overhead (up to 100,000× in ECDHE algorithm [27]) for maximized resilience against side-channel attacks. Besides performance overheads, the approaches require recompilation of protected programs.

Klotski [29] argues for a more practical attacker modeling and presents a practical page-level obfuscation engine that balances security and performance by leveraging larger scratchpad sizes and periodic rerandomization.

2.4. Threat model

IncognitOS adopts the conventional CVM threat model, where the hypervisor is untrusted and may commit to powerful side-channel attacks that undermine the confidentiality of the CVM. The *Trusted Computing Base (TCB)* encompasses the CPU hardware, the entire CVM, and IncognitOS components. We assume the attacker has knowledge of the CVM application and can send requests to it, enabling runtime page access profiling and secret extraction attacks through side channels. IncognitOS aims to thwart these attacks through obfuscation while maintaining reasonable performance. We do not consider transient execution attacks, e.g., Spectre [45] and Meltdown [46]. Their variants for Intel SGX enclaves have been patched in SGX's latest iteration [11], [12]; their effectiveness on CVMs has not been extensively studied.

3. INCOGNITOS Overview

INCOGNITOS (Figure 1) is a unikernel design that targets AMD SEV-SNP for practical and transparent system-level defense against CVM side-channel attacks. It proposes a novel *rate-adaptive obfuscation* scheme that allows relaxing the rerandomization rate during normal execution, as INCOGNITOS can sense the symptoms of attacks and reactively increase the rate.

3.1. Kernel subsystems

As shown in Figure 1, INCOGNITOS retrofits the following kernel subsystems to enforce its rate-adaptive obfuscation scheme.

Scheduling subsystem. INCOGNITOS's scheduling subsystem (§4) constructs a sampling system that can accurately determine the current VMExit rate. Using the measured frequency of VMExit, the subsystem establishes a high-level *rerandomization policy* to regulate the system's obfuscation. To this end, INCOGNITOS's scheduler implements a trustworthy self-invocation mechanism independent of the untrustworthy timer interrupts. Our design decision was to implement so-called *synchronous* ticks inserted throughout the user program and kernel to invoke the scheduler.

Paging subsystem. The paging subsystem (§5) takes advantage of the direct access to the hardware MMU now available to CVMs to implement a page rerandomization scheme. It integrates an ORAM scheme into the OS page tables to perform secure page-ins and page-outs. It performs on-demand rerandomization at a frequency regulated by the scheduler. INCOGNITOS paging allows the CVM to enjoy full freedom using the *Guest Virtual Address (gVA)* space, as INCOGNITOS's rerandomization is achieved transparently in the OS-level.

3.2. System requirements

INCOGNITOS's scheduling and paging subsystem must achieve functional design requirements (**R2**-**R4**) while respecting the userspace by maintaining transparency (**R1**).

R1: Transparent Obfuscation. As an operating system, INCOGNITOS's obfuscated execution must be transparent to the user programs that it hosts. It must not require porting or even recompilation of the user programs. This requirement has an impact on many design decisions made in INCOGNITOS. For instance, while INCOGNITOS's synchronous ticks and VMExit sampling system resemble methods used in a previous work [30], it leverages static binary instrumentation to support unmodified binaries.

R2: Self-sovereign scheduling. INCOGNITOS must implement a self-sovereign way of scheduling critical tasks independent of the timer interrupts the hypervisor delivers due to untrustworthy virtualized interrupts (§2.2). At each scheduling *tick*, INCOGNITOS can carry out self-defensive operations such as measuring the current VMExit rate and rerandomizing CVM memory as necessary.

R3: Robust VMExit rate sampling and policy. INCOGNITOS's scheduling must be equipped with a robust VMExit rate sampling system. Accurately and continuously measuring the VMExit rate is not a trivial task. For instance, what is the system's necessary sampling rate for accurately sampling the VMExit frequency range of the attacks? This robust sampling allows us to build a high-level policy that moderates the rerandomization rate.

R4: Full-system memory rerandomization. INCOGNITOS's rerandomization of CVM memory must be transparent to userspace and, at the same time, must also include the kernel components. Notably, we found that rerandomizing the locations of the page tables themselves is indispensable to avoid leaking INCOGNITOS's memory layout.

3.3. Embracing unikernel design

INCOGNITOS makes a case for unikernels as a basis for efficient transparent (**R1**) and full-system (**R4**) obfuscation of CVMs. The unikernel-based approach brings the following unique design leverages.

Minimized memory footprint and complexity. The unikernel adoption takes full advantage of its minimal memory footprint to achieve full-system page access obfuscation with lower overheads. Even compared to Linux MicroVMs [47] running NGINX, a Unikraft-based NGINX deployment consumes 93% less memory during runtime (2MB vs. 29MB) [48]. Moreover, the significantly lower code complexity of the unikernels compared to the Linux kernel facilitates INCOGNITOS's rerandomization scheme that directly modifies the kernel paging.

Single address space and privilege level. INCOGNITOS takes full advantage of direct hardware access in unikernel's single privilege level execution to implement efficient OS-level obfuscation techniques. For instance, INCOGNITOS's scheduler design that must be invoked at a high frequency is feasible due to the lack of user-kernel switching costs. Similarly, the INCOGNITOS's performance overhead of an unusually high rate of *Page Faults (PFs)* from constant rerandomization is significantly alleviated. The unified address space for user and kernel pages makes it harder for adversaries to isolate specific program behaviors. For example, prior attacks [22], [23], [49] typically filter out kernel page accesses to reduce noise when profiling target user programs.

Portability and application support. INCOGNITOS, being a fork of Unikraft, inherits the achieved and ongoing efforts towards practicality and compatibility. As INCOGNITOS's defense is designed to be transparent to the hosted program, it can support various programs already ported to

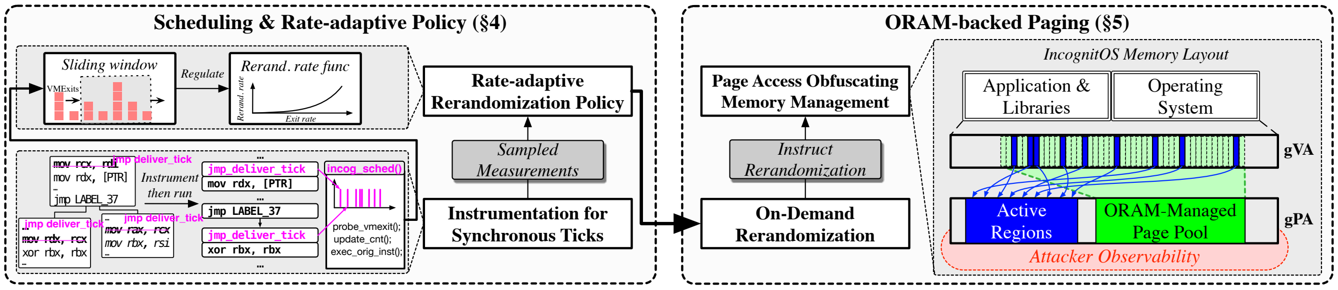

Figure 1: Overview of INCOGNITOS

Unikraft, such as NGINX, Redis, Node.js, and Flask and unmodified binaries through its ELF loader [50].

3.4. Adaptive defense strategy

Before explaining the rationale behind our strategy, we discuss the empirical analysis that shows the VMExit rates of benign workloads and reproduced side-channel attacks on a SEV-SNP CVM.

Empirical analysis: VMExit rates during attacks. We empirically analyzed the VMExit rates in workloads and attacks to support our motivation for the work and acquire concrete data for establishing system design requirements. The results are shown in Table 1. To this end, we implemented an in-hypervisor measurement framework that places probes inside the KVM's VMExit handler (`svm_vcpu_enter_exit`) and also counts executed CVM instructions using the AMD's hardware performance monitoring tool [51]. The framework collects the runtime traces of (1) the number of VMExits and (2) the instruction distance between each VMExit. Hence, the VMExit rate is calculated in the unit of instructions per VMExit. All experiments were conducted with INCOGNITOS instance inside SEV-SNP without its obfuscation capabilities. We implemented the attacks by either developing them from scratch based on previous methodologies (e.g., the SGX-based controlled-channel attack [8], CVM profiling [22]) or by porting the open-sourced version to our environment, e.g., single-stepping frameworks for AMD SEV [24], [44].

L1, L2, and L3 represent the VMExit frequencies of benign execution, while L4, L5, and L6 represent hostile scenarios in which side-channel attacks are employed. We obtained the geometric means (GeoMean), the geometric standard deviation (GSD), and the maximum value of VMExit frequency sampled in these scenarios.

Motivation: Large gaps in averages. The results indicate that the side-channel attacks indeed accompany abnormally high VMExit frequency (f_{VMExit}), distinguishable from the averages of the normal CVM executions. Even with the attack L4 that is known to suppress the number of NPFs by only monitoring data page accesses, the frequency is about 50.68× higher than the average seen in intensive NGINX execution: L2 (NGINX): 0.0016 vs. 0.0811. In other words, even the low-exit NPF attack (L4) induces a VMExit for every 12.33 CVM instruction execution ($T_{\text{VMExit}} = 1/f_{\text{VMExit}}$). This means that the VMExit frequencies of benign workloads and adversary-bound side-channel attacks are generally distinguishable in terms of average VMExit frequencies.

ID	Workloads / Attacks	VMExit Frequency (exits/instruction)		
		GeoMean	GSD	Max
L1	`nbench` benchmark suite	0.0009	2.37	0.06
L2	NGINX (1000 requests, 8 concur. conn., 4KB file)	0.0016	10.77	0.13
L3	Redis (1000 requests, 50 concur. conn., 4KB payload)	0.0019	7.54	0.13
L4	Low-exit NPF attack [8]	0.0811	2.84	0.5
L5	NPF-based profiling [19], [22], [24]	0.1658	2.80	0.5
L6	Single-stepping [24]	0.8973	1.58	1.0

TABLE 1: SEV-SNP CVM VMExit frequency (exits/isns) in benign workloads and attacks. L4 were performed on LibJPEG (§6.3), L5 and L6 were performed on Redis workload.

Motivation: Spikes in benign workloads. However, we also observe the high-frequency spike appearances in benign workloads as shown in L2, L3's *Maximum* f_{VMExit} intervals. The highest VMExit frequency interval observed in NGINX (L2) and Redis (L3) was measured to be 0.13. The number surpasses the average VMExit frequency of the low-exit NPF attack (L4, 0.0811) and is close to conventional NPF profiling attacks (L5, 0.1658). Our manual in-depth analysis identified that the sources of the spikes are consecutive NPFs from demand paging during the process initialization and from I/O requests from `virtio`. We expect such occurrences to be common in benign workloads that require large working sets or network connections.

Defense strategy. The above analysis provides implications that serve as motivations for INCOGNITOS's adaptive defense strategy. First, the VMExit frequency ranges of the benign workloads and side-channel attacks show clear gaps, further confirming the high VMExit frequency as indicative of ongoing attacks. Given the clear gaps in the average VMExit frequencies, we argue that a fixed rerandomization rate throughout execution [29] is inefficient. Second, our testing proves our concern that benign workloads with exceptionally high VMExit rate spikes can render fixed threshold schemes used in SGX by previous work infeasible for SEV-SNP [30], [31]. This calls for a robust mechanism that tolerates potential spikes in frequencies. Lastly, the numbers from the analysis serve as a concrete requirement in designing INCOGNITOS's scheduler's VMExit sampling.

INCOGNITOS proposes a rate-adaptive defense strategy that overcomes the inefficiency of a fixed-rate rerandomization and robustness problems of a fixed threshold. To implement the strategy, INCOGNITOS adjusts the *rerandomization rate* proportionally to the detected VMExit frequency, representing the adversary's temporal resolution.

4. Scheduling for Rate-Adaptive Policy

In this section, we explain the design of INCOGNITOS's scheduler. INCOGNITOS implements a static binary instrumentation tool (**R1**) to plant synchronous ticks in *each basic block* program executables and achieve self-sovereign scheduling (**R2**). Each synchronous scheduler tick must detect the occurrence of the VMExit and count executed instructions to calculate the VMExit frequency (§4.1). We found that constructing a robust VMExit frequency sampling system (**R3**) is a daunting challenge. We make careful design decisions to make the system as robust as possible (§4.2). With the synchronous ticks implemented (§4.3), INCOGNITOS finally constructs a high-level policy atop (§4.4).

4.1. Data acquisition at ticks

Each synchronous tick invocation checks and reports to the scheduler 1) the VMExit occurrence since the last tick and 2) the number of executed instructions since the previous tick. These two key data for VMExit frequency measurement are collected through the following devices:

VMSA mapping for VMExit detection. INCOGNITOS leverages the unforgeable VMExit occurrence information recorded in the VMSA (§2.1) to establish a primitive for in-CVM sampling of VMExits. While the initial VMSA of the boot vCPU (initialized by the hypervisor) is not accessible by the CVM, we found that the secondary vCPU, i.e., *application processors (APs)*, can access their VMSA since the VMSA is initialized by the CVM itself [16]. Thus, during CVM boot, INCOGNITOS ensures that the VMSA page is securely mapped into the guest address space. RMP checks ensure the integrity of the VMSA by preventing tampering: the host cannot forge *Host Physical Address (hPA)* to *Guest Physical Address (gPA)* mappings or perform unauthorized write accesses to CVM's private memory.

Instructions executed as passage of time. Each tick delivers the instructions count of its parent basic block to the scheduler, thereby allowing the scheduler to accumulate the total number of instructions executed in between ticks. The instruction count of each basic block is precalculated during static instrumentation and stored in the executable (explained in detail in §4.3). INCOGNITOS utilizes the executed instructions count to measure the passage of time for security and robustness reasons (**R2**, **R3**). While SEV-SNP provides SecureTSC [16] that allows secure acquisition of the current time, the usage of *absolute time* to measure the VMExit rate would be insecure. For instance, the hypervisor can intentionally delay the scheduling of a CVM to provide a misleading VMExit count per second.

4.2. Tick placement strategy

INCOGNITOS's tick placement adopts two key rules to maximize the robustness of VMexit frequency sampling (**R3**): the Nyquist sampling frequency and per basic block ticks for maximum loop coverage.

Nyquist Sampling frequency. We define the required *sampling rate* according to the *Nyquist Theorem* [52], based on the observation that our VMExit frequency measurement is essentially a signal sampling problem where the value of `vmsa.exit_code` over time forms a *square wave signal*. Following the Nyquist sampling theorem, the required sampling rate for accurately measuring a signal whose frequency is f_{sampled} must be at least $2 \times f_{\text{sampled}}$, and therefore:

$$f_{\text{Tick}}^{\text{Req}} \geq f_{\text{VMExit}}^{\text{Target}} \times 2.$$

$f_{\text{VMExit}}^{\text{Target}}$ must be a value between the frequency levels of benign workloads and attacks, such that INCOGNITOS can apply an adaptive rerandomization rate with minimized false-positive cases reliably.

From our empirical analysis (Table 1), we know the average f_{VMExit} of the highest benign workload and lowest attack f_{VMExit} to be 0.00164 (L2) and 0.081103 (L4). Hence, INCOGNITOS sets its target frequency to be *two* times the frequency of benign execution, $f_{\text{VMExit}}^{\text{Target}} = 2 \times 0.00164 = 0.00328$. Thus, the required, or Nyquist, $f_{\text{Tick}}^{\text{Req}}$, must be at least twice that amount, 0.00656, or once every 152 instruction execution.

Loop coverage. Apart from achieving the Nyquist frequency, we found incomplete *loop coverage* with ticks to be the most prominent threat to robustness during implementation. Given that accurately identifying loops in binary is a known hard problem [53], we resort to a per-basic-block tick placement scheme as mentioned, which also contributes to achieving the required sampling frequency.

4.3. Tick and trampoline implementation

INCOGNITOS's binary instrumentation tool is based on the `e9patch` binary rewriting framework [53]. The tool instruments an unmodified dynamically-linked Unikraft binary to insert synchronous ticks and trampolines for INCOGNITOS scheduler invocation.

Synchronous ticks. INCOGNITOS synchronous ticks are placed in each basic block of the user program to invoke its matching *trampolines*, and eventually, the scheduler function. An INCOGNITOS synchronous tick is a 5-byte relative jump instruction (`jmp rel32`) to a unique trampoline that is paired with the tick. The instrumentation tool iterates over each basic block, selecting one instruction within a basic block to be replaced with an INCOGNITOS tick. Due to the non-intrusive nature of jumps and the generally longer byte sequences of x86 instructions, finding a 5-byte sequence in a basic block is not challenging. However, basic blocks can exist without such suitable instructions. In such cases, the tool opts for the fallback strategy that revolves around *instructions punning* [53], [54]. This design choice of employing jump-based ticks takes full advantage of the unikernel design. Due to the lack of the user-kernel barrier in INCOGNITOS, the *trampoline* of the jumps is only a few cycles away.

Trampolines. For each tick inserted, the tool also generates a unique trampoline (e.g., `trampoline_UUID` for `tick_UUID`.) Each generated trampoline performs three operations: 1) executes the overwritten instructions that had been relocated to the trampoline body during instrumentation, 2) loads the hardcoded *instruction count* of the source basic block into the argument register then finally 3) calls INCOGNITOS's scheduler function (`incog_sched()`).

Scheduler function. On receiving the control, the scheduler first inspects the `vmsa.exit_code` field of the VMSA to sample the occurrence of a VMExit since the last tick. `incog_sched()` always sets the field to a special value that it can recognize (i.e., `0xFFF`); the change of the value since the last tick indicates a VMExit occurrence. The instruction counts forwarded by the trampoline and the detection of VMExits are consumed by the scheduler's rate-adaptive policy (§4.4).

Kernel tick placement. Unlike user program executables, placing ticks in kernel code is performed through compiler instrumentation and manual effort. The unikernel's small code base allows us to identify and avoid problematic cases manually. For instance, ticks should not be inserted into the trampolines and scheduler functions, and a handful of kernel functions that require atomicity must be excluded.

4.4. Rate-adaptive rerandomization policy

Based on the tick delivery and VMExit measurement explained thus far, INCOGNITOS constructs a higher-level policy (**R3**) that schedules *when* the paging subsystem must perform rerandomization.

Achieved sampling capability. Using the described synchronous tick instrumentation and kernel tick placement strategies, we found that the evaluation targets provide a reliable measurement of f_{VMExit}. In particular, we use the previously described measurement framework to obtain the distance between ticks and calculate the tick rate for the evaluation targets (`nbench`, NGINX, and Redis). The average (geometric mean) tick rate for the `nbench` suite, NGINX, and Redis was measured to be 0.023, 0.028, and 0.037, respectively. Based on the previously defined requirement, at its worst tick rate (the `nbench` suite), the INCOGNITOS is capable of achieving 3.51 times $f^{\text{Req}}_{\text{Tick}}$.

Sliding window measurement scheme. We use a sliding window [55], [56] to reliably calculate the current trend in VMExit frequency (i.e., f_{VMExit}), based on samples of detected VMExits and instruction counts. Given a monotonically increasing tick number t, we define a sliding window as a collection of W latest samples of (VMExit$_t$, #InstExec$_t$) delivered by the scheduler. INCOGNITOS computes $f^{\text{t}}_{\text{VMExit}}$ at the current tick t as follows:

$$f^{\text{t}}_{\text{VMExit}} = \frac{\sum_{i=t-W}^{t} \text{VMExit}_i}{\sum_{i=t-W}^{t} \#\text{InstExec}_i}, \text{ where VMExit}_i \in \{0,1\}$$

	Description
W	Number of samples in single sliding window
$f^{\text{Alarm}}_{\text{VMExit}}$	Frequency threshold to trigger alarmed rerandomization state
$f^{\text{Normal}}_{\text{ReRand}}$	Frequency of rerandomization during *un*alarmed state
$\mathbb{F}_{\text{Alarm}}$	Function that determines rerandomization frequency $f^{\text{Alarm}}_{\text{ReRand}}$ with current measured f_{VMExit} as input during alarmed state
G	Grace period constant where termination happens after G consecutive sliding windows measurements above $f^{\text{Alarm}}_{\text{VMExit}}$

TABLE 2: Configurable parameters to policy

This enables efficient computation of VMExit frequency using two size-W ring buffers that track VMExit and InstExec samples. Running sums are maintained by adding new values and subtracting expired ones, avoiding iteration over tick samples.

Policy formulation. A high-level rerandomization frequency regulation policy can now be constructed based on the robust VMExit frequency measurements from the current sliding window. Table 2 explains the configurable parameters of the policy. Below is the policy at a glance:

$$f^{\text{t}}_{\text{ReRand}} = \begin{cases} f^{\text{Normal}}_{\text{ReRand}} & \textit{if } f^{\text{t}}_{\text{VMExit}} < f^{\text{Alarm}}_{\text{VMExit}} \\ \mathbb{F}_{\text{Alarm}}(f^{\text{t}}_{\text{VMExit}}) & \textit{otherwise} \end{cases}$$

$$\text{where } \mathbb{F}_{\text{Alarm}}\ (f^{\text{t}}_{\text{VMExit}}) = {f^{\text{t}}_{\text{VMExit}}}^2 \times \alpha$$

INCOGNITOS maintains a relaxed rerandomization rate of $f^{\text{Normal}}_{\text{ReRand}}$ when the measured VMExit frequency is below the threshold frequency $f^{\text{Alarm}}_{\text{VMExit}}$. However, when the threshold exceeds the threshold, the policy enters an *alarmed rerandomization state*. The rerandomization frequency during the alarmed state ($f^{\text{Alarm}}_{\text{ReRand}}$) is regulated through the function $\mathbb{F}_{\text{Alarm}}$. Currently, we use a simple quadratic function for $\mathbb{F}_{\text{Alarm}}$ that allows the $f^{\text{Alarm}}_{\text{ReRand}}$ upward curve to maximize when attack frequencies are measured. Also, INCOGNITOS allows an optional termination policy to abruptly terminate the CVM after a *grace period* (G). If VMExit frequencies higher than the threshold $f^{\text{Alarm}}_{\text{VMExit}}$ are observed for G consecutive windows, CVM termination proceeds. A value of 0 for G signifies that the termination policy is disabled.

Rationale for default parameters. As to the concrete parameters we use for evaluation, we set the $f^{\text{Alarm}}_{\text{VMExit}}$ to be 0.003 as mentioned (Nyquist frequency of 2× benign workload frequency). This threshold renders the low-exit NPF attack (L4) infeasible due to constant randomization, and only occasional benign high-VMExit execution suffers as a side effect. Also, we used the values 1/10K, 1/100K, 1/500K, 1/1M, 1/2M for $f^{\text{Normal}}_{\text{ReRand}}$. The rationale behind the general range is that INCOGNITOS's rate can be substantially lower than the previous work's fixed rerandomization rates (e.g., every 1K *memory accesses* [29]). Hence, we assume one memory access per 10 instructions (1/10K) and relax the number in multiple steps. Additionally, we used $W = 100$ and $G = 1000$, which we empirically found optimal for promptly detecting VMExit rate changes during workloads and attacks.

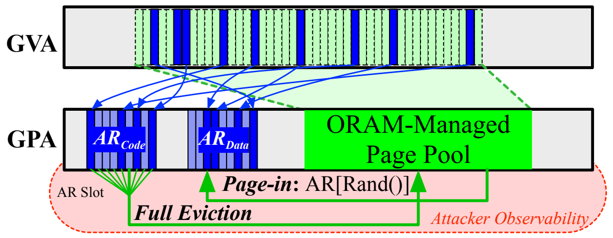

Figure 2: INCOGNITOS's system memory layout and page management

5. OS-level Page Access Obfuscation

INCOGNITOS achieves OS-level page access obfuscation that operates independently from the gVA space. INCOGNITOS's paging subsystem maintains all CVM memory pages, including kernel and userspace pages (**R4**) as well as the *Guest Page Tables (GPTs)*, in an ORAM-backed page pool and only allows memory accesses through a narrow and constantly rerandomized gPA *active regions*. It carries out page-ins and page-outs in a page-access oblivious manner. Also, it responds to the scheduler's rerandomization commands with eviction-based memory rerandomization to reshape the gPA space.

5.1. System memory layout

The paging subsystem manages the following components to maintain the INCOGNITOS system memory layout as shown in Figure 2.

ORAM-managed page pool. The page pool implements the Path ORAM scheme [57], where each node is a page. The path ORAM algorithm maintains a binary ORAM tree with height l containing $N = 2^l - 1$ buckets, each containing Z blocks. It also maintains an ORAM stash of S in size to house fetched ORAM blocks temporarily. An ORAM node may be a dummy page containing no actual data or a real page that stores the content of a memory page.

The Path ORAM algorithm performs data transfers between the ORAM tree and the ORAM stash at the granularity of *paths* on the ORAM tree [57]. It keeps a *Position Map (PosMap)* structure that maps each ORAM block to a randomized path indexed by the leaf nodes (*leaf ID*). In INCOGNITOS, both the ORAM tree and stash are implemented as an array of pages in contiguous physical pages reserved during CVM boot. The ORAM PosMap is integrated with the OS page table, as described in §5.2.

Active gPA regions. INCOGNITOS manages gVA address space and also the OS page table into the constrained and constantly randomized *active gPA regions* ($\mathcal{AR}_{Code}$ and $\mathcal{AR}_{Data}$ in Figure 2). INCOGNITOS manages *separated regions* for CVM code, data (including stack and heap), and page tables ($\mathcal{AR}_{code}$, $\mathcal{AR}_{data}$, and $\mathcal{AR}_{pgt}$). This design decision is based on the fact that SEV SNP's hypervisor-side NPF handler notifies the untrusted hypervisor of the type of PF: code fetch, data fetch, and guest page table faults. Hence, mixing the three types of pages in a smaller active gPA region does not increase security.

Active region size. Each active region can shelter a configurable number of *slots* (S) of 4KB pages. A slot may be *occupied*, containing a page with an active mapping, or *empty*. Unlike previous work that prefers an active region (or scratchpad) that spans a single rerandomization unit or a small number of units [27], [29], INCOGNITOS chooses to acquire entropy through larger active regions (8,192 slots of 4KB per region), due to the known ciphertext weakness [21] of SEV-SNP. We further discuss the issue in §7.1.

Isolated pages for inevitable disclosures. INCOGNITOS isolates certain kernel components whose locations are inherently exposed to the hypervisor into non-randomized address spaces to prevent information leakage. For instance, the SEV-SNP architecture permits the hypervisor to inject faults/interrupts, allowing it to monitor *Interrupt Service Routines (ISRs)* and their subsequent execution. When left on their original pages, these components may expose other code and data residing on the same page. Such pages include pages containing explicit calls to the hypervisor (`vmgexit`), memory mappings for I/O, and ISR pages (including the OS PF handler). We further discuss the security of the PF handler in §7.1.

5.2. Page management and randomization

INCOGNITOS maintains an ORAM-integrated page table that controls the mappings from gVA (guest virtual address) to gPA (guest physical address) to regulate the CVM's memory access within the constantly randomized active regions. As a result, the gVA remains unchanged after its initial setup. Memory rerandomization and page management are performed using the *page-in* and *page-out* procedures, which obliviously transfer pages between the ORAM-managed page pool and the active regions.

Integration of ORAM into OS page tables. INCOGNITOS's approach for coherently integrating ORAM into OS page tables is to essentially transform the *Page Table Entries* themselves to function as the PosMap of the Path ORAM algorithms. This eliminates the need to maintain a separate PosMap data structure. PTEs in an INCOGNITOS-managed page table may reside in one of three states: (1) `active`, where the encoded *Page Frame Number (PFN)* points to a region within the active regions or (2) `paged-out`, the backing page has been evicted to the ORAM page store, or (3) `unallocated`, which indicates that a new physical page frame must be allocated to support OS demand paging. Specifically, for `paged-out` pages, INCOGNITOS reuses PTEs of evicted pages to store their leaf IDs.

Page fault handling. INCOGNITOS's PF handler is invoked when the MMU encounters any PTE with its present bit cleared (`paged-out` and `unallocated`). It first performs a *software* page table walk to identify the non-present PTE. For `unallocated` PTEs, the handler selects a random slot within the active region corresponding to the type of fault for its allocation ($\mathcal{AR}_{Data}$ for read and write, and $\mathcal{AR}_{Code}$ for execute, and $\mathcal{AR}_{gpt}$ for page table faults). If the chosen slot is occupied, the existing page is evicted through the page-out process. Page-ins occur upon PFs on `paged-out` PTEs, and the corresponding page is fetched from the ORAM-managed page pool into the active region.

Page-In. The page-in process begins with a Path ORAM path fetch that retrieves pages based on the ORAM leaf ID in the PTE and moves them to the ORAM stash [57]. A slot within the active region according to the type of fault is then randomly chosen as the gPA for the paged-in page, similar to the handling of `unallocated` PTEs. The page corresponding to PTE is then copied from the stash into the chosen slot. This page is identified using the faulting gVA (e.g., `0xff..cc000`) and the page level where the PTE resides (e.g., PD or PT), which is stored as the page's ORAM metadata. Since the ORAM stash resides in untrusted memory, to obscure which page is being fetched from the stash, IncognitOS utilizes an oblivious memory access wrapper adapted from previous work [27]. The wrapper ensures that all physical pages in the stash are touched, regardless of the target page.

Page-Out. Page-outs occur on two occasions: (1) to serve the rerandomization scheme described next and (2) when the random slot for page allocation is occupied. On a page-out, the target page is first written to the ORAM stash, where it will eventually return to the ORAM tree through ORAM shuffling. ORAM algorithms utilize both real and dummy nodes. Since the stash is stored in untrusted memory, the location of dummy slots must be protected to maintain ORAM's security guarantees [29]. To address this, IncognitOS's stash implementation adopts a modified `WriteStash` operation from previous work [29]. It keeps track of the last stash slot written to, incrementing this index with each write to the stash. When new pages (dummy or real pages) are to be added to the stash, an unused stash slot is selected starting from this index, resulting in an always-increasing write access pattern. If no slots are available, the stash is reshuffled by compacting real blocks to the beginning using the oblivious memory wrapper, after which the empty slot index is reset. Finally, the page's corresponding PTE is then encoded to be `paged-out`.

Rerandomization through full eviction. IncognitOS's paging subsystem performs eviction-based rerandomization upon receiving a request from the scheduler. When rerandomization is to be performed, the paging subsystem evicts *all* currently occupied slots within the active regions back to the ORAM-managed page pool. Thus, memory accesses that follow the rerandomization would automatically construct randomized memory layouts through the oblivious and random page-in process.

Inclusion of page tables in rerandomization. To maintain the obfuscation of the entire system (**R4**), IncognitOS incorporates the GPT into the rerandomization scheme. Although this design choice adds non-trivial complexities, we must address the *Nested Page Fault on GPT (NPF-on-GPT)* information leak we have identified. Specifically, while an untrusted hypervisor has no direct source to leak gVA accesses, exposed GPFNs of the GPTs themselves can serve as an alternative identifier for the 4KB pages they map. For instance, if there is a NPF-on-GPT on one of two PTEs located at PFN `0xaa000` and `0xab000`, it means that a memory access to a specific set of 512 4KB pages is imminent. As a result, the adversary can leak the gVA access pattern of the CVM execution with up to 2MB (4KB×512) granularity by exploiting PTEs location leaks.

Since access to higher-level GPTs (PML4, PDPT) would only leak extremely coarse-grained information, we focus on the last two levels (PD and PT). IncognitOS places each layer of the GPT in a dedicated active region (e.g., $\mathcal{AR}^{PT}_{gpt}$,$\mathcal{AR}^{PD}_{gpt}$), such that their rerandomization may be performed independently. To respect the dependencies between pages and their corresponding PTE, IncognitOS's page evictions are performed in the reversed order of the GPT hierarchy. That is, page evictions are performed in the order of $\mathcal{AR}_{Code}$, $\mathcal{AR}_{Data}$ first, then $\mathcal{AR}^{PT}_{gpt}$ and finally $\mathcal{AR}^{PD}_{gpt}$.

6. Evaluation

In this section, we evaluate IncognitOS in terms of its security and performance feasibility through six experiments: 1. robustness of VMExit frequency measurement (§6.1), 2. access pattern profiling resilience in NGINX (§6.2), 3. LibJPEG image extraction attack mitigation (§6.3), 4. `nbench` microbenchmark (§6.4), and real-world application performance benchmarks (§6.5) with 5. NGINX and 6. Redis. We provide a qualitative security discussion in §7.1.

Implementation. IncognitOS is implemented as a fork of the Unikraft unikernel version v0.15.0 [58]. The kernel was modified to support whole-VM memory encryption and mapping integrity protection during boot, along with *Guest-Host Control Block (GHCB)*-based communication with the hypervisor [16]. Additionally, the networking stack was updated for compatibility with the SEV-aware `virtio` driver. $3,401$ lines of code were added or modified for SEV-SNP compatibility, not including IncognitOS's design elements. We implemented IncognitOS's kernel component in $9,427$ lines of code and its binary rewrite in 682 lines as a plugin to the `e9patch` binary instrumentation framework [53].

Experiment environment. The evaluation is performed on a server equipped with AMD EPYC 7513 CPU@2.0GHz (AMD SEV-SNP capable) and 256GB of RAM, running Ubuntu 22.04 with kernel version 6.9.0-rc. Also, the experiments used SEV-supported QEMU (commit hash `bf83c1b9`) and OVMF (commit hash `80318fcd`) versions provided by AMD. We have also taken measures to reduce variances during the benchmarks. For all experiments, the QEMU process and the benchmarking program (e.g., `ab`) are pinned to dedicated cores (e.g., `taskset -c 4`) that are reserved at boot (using `isolcpus` kernel parameter).

Parameter settings. All parameters to IncognitOS's policy are applied as described in §4.4. However, note that we disabled the grace period mechanism by setting $G = 0$ to fully illustrate the effects of rerandomizations for the experiments in §6.1, §6.2, and §6.3. The following table summarizes the parameters that were used, which also includes the ORAM configurations:

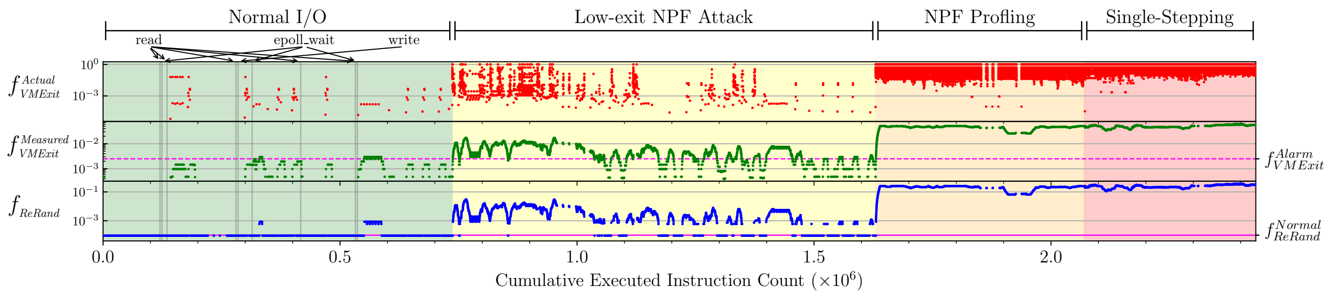

Figure 3: VMExit frequencies measured by INCOGNITOS vs. ground truth (in-KVM), and the resulting adaptive randomization rates for Redis under various workloads and attacks.

INCOGNITOS Parameters					ORAM Params.		
$\mathcal{AR}$ Size	W	$f^{\text{Alarm}}_{\text{VMExit}}$	$f^{\text{Normal}}_{\text{ReRand}}$	G	N	S	Z
$8,192$ (32MB)	100	0.003	$\frac{1}{10K} \sim \frac{1}{2M}$	1000	$32,768$ (128MB)	512	4

6.1. Robustness evaluation

We evaluated the robustness of INCOGNITOS adaptive rerandomization scheme with INCOGNITOS-hosted Redis, as shown in Figure 3. The graph allows us to evaluate how close INCOGNITOS's VMExit frequency measurements ($f^{Measured}_{VMExit}$) and rerandomization rate (f_{ReRand}) approximate the ground truth VMExit frequency measurements (f^{Actual}_{VMExit}).

Measurement method. The data points of the in-hypervisor framework mentioned in §3.4 (ground truth) and INCOGNITOS are synchronized at each of INCOGNITOS's ticks. At each invocation, the scheduler reports its VMExit frequency and rerandomization rate to the framework via a custom hypercall excluded from the VMExit counts.

Experiment setup. We applied 1) normal I/O workload, 2) low NPF controlled-channel attack, 3) conventional NPF controlled-channel attacks, and 4) single-stepping (L3, L4, L5, and L6 from Table 1) to the CVM in a sequence, whose intervals are labeled with *Normal I/O*, *Low-exit NPF Attack*, *NPF Profiling*, and *Single-Stepping* in the graph, respectively. The initiation of the attacks was synchronized with Redis's `accept` system call that signifies the beginning of Redis request handling. The low NPF controlled-channel attack targets specific pages that give away information and, therefore, are program-specific. Thus, we first profiled the working set (i.e., frequently accessed pages), then randomly chose 10% of the pages to be monitored to emulate the effect of the attack.

Result discussion. Overall, the result in the graph confirms the robustness of INCOGNITOS's VMExit frequency measurement. We found the following observations to be noteworthy:

First, the results show that INCOGNITOS suppresses the rerandomization frequency during benign workloads while promptly reacting to the attacks to fight them with a very high rerandomization frequency. When the ongoing attack is absent, $f^{\text{Normal}}_{\text{ReRand}}$ is mostly maintained with minor spikes. However, INCOGNITOS showed drastically higher frequency ($f^{\text{Alarm}}_{\text{ReRand}}$) applied to low exit NPF, conventional NPF, and single-stepping. More specifically, the average randomization frequencies were adjusted to be about 0.003, 0.2007, and 0.245 on average, respectively (i.e., once rerandomization every 333, 4, and 5 instructions). 93.8% of ticks triggered alarmed frequencies of rerandomization, even with the low-exit NPF attack. This number increases to 100% for NPF profiling and single-stepping attacks.

Second, INCOGNITOS reliably handled the VMExit frequency spikes during the benign workloads. If an instantaneous decision was to be made with a fixed threshold, these could have been mistaken for an attack. We manually confirmed that these spikes occurred during I/O activities. We marked the locations of spikes with identified sources on the timeline by labeling invoked system calls, such as `read` and `write`, on the timeline. INCOGNITOS handled the spikes with momentarily alarmed rerandomization frequency enforcement, yet the execution continued.

6.2. Access pattern profiling resilience

The NGINX memory address profiling attack experiment demonstrates INCOGNITOS's resilience to code data access profiling attacks (§8.1). Figure 4 presents the profiling results of two versions of INCOGNITOS-hosted NGINX web servers: one without randomization, whose observable traces are shown with the label *Without Rerand*, and one with INCOGNITOS's rerandomization policy (*With Rerand*.

Experiment setup. We adapted the technique from a previous work that performed a profiling attack on a `OpenSSH` server [22] to a INCOGNITOS-hosted NGINX. The attack traces the CVM's page access patterns through the NPF controlled channel and computes the CVM's *access frequency* for each unique gPA page based on the trace. Both web servers are configured to use the active regions with identical sizes (8,192 slots) for a more accurate comparison.

Result discussion. The results show that performing NPF-based profiling is highly infeasible against INCOGNITOS's adaptive rerandomization. During the entire attack duration, an average $f^{\text{Alarm}}_{\text{ReRand}}$ of 0.2 was observed, triggering

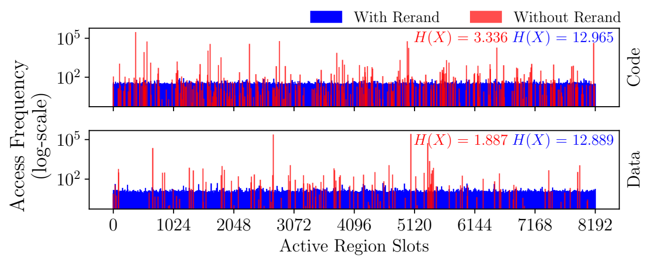

Figure 4: NPF controlled-channel profiling results of INCOGNITOS-hosted NGINX code and data page access frequency during single HTTPS request. $H(X)$ values show Shannon entropy values.

a rerandomization at almost every tick. Each rerandomization would render the attack meaningless since each physical page will take a random position (out of 8,192 slots) after the rerandomization.

In the case of *Without Rerand*, the page access patterns reveal distinct profiles during request handling (max = 289,995/237,372, mean = 82.94/67.48, std. = 3471.95/3684.0 for $\mathcal{AR}_{code}/\mathcal{AR}_{data}$). This quintessential access frequency pattern allows the adversary to narrow down or pinpoint the sensitive pages. On the other hand, the *With Rerand* histogram illustrates the effect of elevated rerandomization rate, which shows the collected profile approximating a uniform distribution, where only slight variations are observed (mean = 24.14/7.78, std. = 5.61/3.24 for $\mathcal{AR}_{code}/\mathcal{AR}_{data}$).

To further validate the results, we computed the *Shannon entropy* for each histogram that quantifies the unpredictability of page accesses to each active region slot. High entropies of $H(X) = 12.965$ for code and 12.889 for data were observed for the constantly rerandomized NGINX, which translates to roughly $2^{13} = 8,192$ all possible page locations, meaning that the page access patterns are indistinguishable.

6.3. Thwarting JPEG image extraction attack

The LibJPEG image extraction attack [8] visually demonstrates INCOGNITOS's effectiveness. The results in Figure 5 illustrate the impact of INCOGNITOS's rate-adaptive randomization policies on the extracted image (*With Adaptive rerand.*) compared to the unprotected version (*Unprotected*).

Experiment setup. We reenacted the attack that originally targeted LibJPEG secured in an SGX enclave in a previous work [8] for the experiment. In the porting process, we adapted the attack to utilize the CVM NPF controlled channel as its attack primitive. The attack extracts the image processed in the CVM through a statistical reconstruction through the controlled channel. As an identical experiment was in Klotski [29], we followed its setting that allows a best-case scenario for the attacker: the attacker knows the address range of the side-channel vulnerable function (Inverse DCT function) and only needs to count the number of page faults during the critical section to reconstruct the processed image.

Result discussion. Without INCOGNITOS's protection, the attack can reconstruct the patterns of the compressed image with minimal noise (the *Unprotected* image). On the other hand, the constant page rerandomizations (about eight times per extracted pixel) render most of the extracted features unrecognizable (*With Adaptive rerand.* image). Also, the distribution of extracted pixel values (shown as a histogram under the image) becomes smooth because of this.

6.4. nbench microbenchmark

We illustrate the performance characteristics of INCOGNITOS using the `nbench` [59] benchmark. The results are shown in Table 3.

Experiment setup. We deployed different configurations of INCOGNITOS-hosted `nbench`. The *ReRand. Disabled* column shows the benchmark results with the synchronous tick instrumentation but *without* rerandomization, thereby illustrating the isolated overhead from the ticks. We also illustrate the performance overhead of INCOGNITOS's rerandomization in varied *static* frequency of $f^{\text{Static}}_{\text{ReRand}}$ =1/{10K, 100K, 500K, 1M, 2M}. While INCOGNITOS uses rate-adaptive rerandomization in practice, the static frequencies allow us to understand the security and performance tradeoffs when configuring the policy parameters (e.g., $f^{\text{Normal}}_{\text{ReRand}}$). For each configuration, we ran the benchmark for fixed $1,000$ iterations and took the average for the statistics of each run.

Result discussion: isolated tick overhead. The results show an average performance degradation of 3.87× for the `nbench` suite. Since INCOGNITOS performs per basic block synchronous tick placement, the total number of tick triggered varies across different programs. The number of synchronous ticks executed (*#Tick*) in each program in the suit illustrates how the overheads generally scale with the number of ticks. The most significant slowdown occurs in `LU Decomposition` (7.44×, 354M tick executions), while the least slowdown is in `Assign` (1.75×, 179M).

Result discussion: performance impact of rerandomization frequency. Understandably, the rerandomization was a major source of overhead; the geometric mean

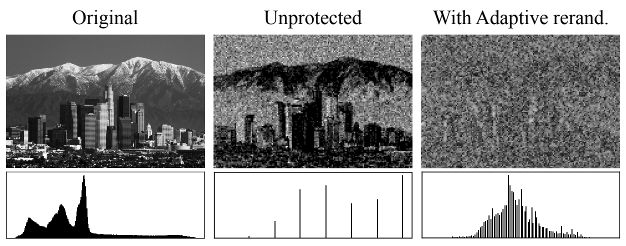

Figure 5: Image extraction results on LibJPEG using NPF controlled-channel attack [8] against unprotected Unikraft CVM vs. INCOGNITOS.

Benchmark	ReRand. Disabled		Static Rerandomization														
			$f^{\text{Static}}_{\text{ReRand}}$ =1/10K			$f^{\text{Static}}_{\text{ReRand}}$ =1/100K			$f^{\text{Static}}_{\text{ReRand}}$ =1/500K			$f^{\text{Static}}_{\text{ReRand}}$ =1/1M			$f^{\text{Static}}_{\text{ReRand}}$ =1/2M		
	Ovh. (×)	#Tick	Ovh. (×)	#RR	#PF	Ovh. (×)	#RR	#PF	Ovh. (×)	#RR	#PF	Ovh. (×)	#RR	#PF	Ovh. (×)	#RR	#PF
NumSort	3.29	326M	381.40	55K	5036.2	41.05	58K	4537.6	14.60	44K	2545.6	8.85	44K	2099.6	7.92	33K	1172.6
StrSort	3.24	313M	327.93	55K	4591.8	39.12	49K	3635	11.57	37K	2450.6	8.09	29K	1752.2	6.31	22K	1122.6
Bitfield	6.74	411M	186.96	53K	17216.6	30.30	38K	10092.8	16.56	24K	3643.4	13.96	20K	2161.4	12.33	15K	1218.6
EmFloat	3.43	291M	477.88	113K	11553.4	71.28	54K	3989.8	19.41	40K	2578.2	12.86	34K	1914	9.46	28K	1286.4
Fourier	5.40	289M	1726.60	76K	3189.4	217.53	52K	1758.4	53.16	44K	1375	30.26	39K	1199.2	18.64	32K	973.6
Assign	1.75	179M	182.40	55K	2598.2	20.44	53K	2236.4	6.24	36K	1404.2	4.25	27K	1035	3.07	19K	704.8
IDEA	5.28	327M	434.82	90K	14955.8	46.40	54K	8378.2	17.78	33K	4189	12.58	24K	2962	9.59	16K	1944.8
Huffman	3.63	289M	130.09	61K	10246.2	18.94	43K	6149.8	8.15	25K	2705.8	6.30	16K	1742.4	5.33	10K	1024.8
NNET	2.25	186M	1106.01	111K	4830.6	110.57	56K	2431	24.84	45K	1945.8	13.94	41K	1741.6	8.24	35K	1464.4
LU Dec.	7.44	354M	1541.90	52K	2558.2	203.94	50K	1885.8	53.91	44K	1418.4	34.12	42K	1118.6	24.02	40K	794.4
GeoMean	3.87		449.87			56.04			17.96			12.01			8.97		

TABLE 3: Performance overheads ***Ovh.*** of different INCOGNITOS configurations, compared against unmodified `nbench` execution. Average number of executed synchronous ticks (***#Tick***), page faults (***#PF***) and rate-adaptive rerandomizations (***#RR***) for each benchmark iteration are also reported. Number of synchronous ticks remains mostly constant across settings.

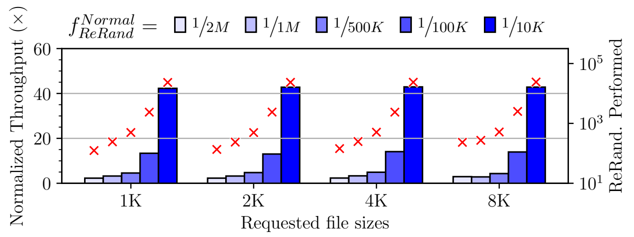

Figure 6: Throughput of INCOGNITOS-hosted NGINX web server with varied requested file sizes and rerandomization frequency, normalized to the throughput of baseline NGINX Unikraft. × marks number of rerandomizations performed.

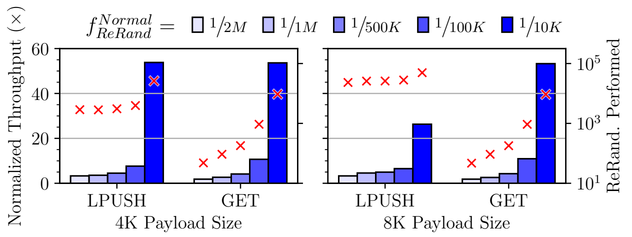

Figure 7: Throughput of INCOGNITOS-hosted Redis in back-to-back `LPUSH` and `GET` workloads with 4K and 8K payloads, normalized to throughput of baseline Redis on Unikraft. × marks number of rerandomizations performed.

of overhead in the benchmarks was lowest (8.97×) when the rerandomization frequency was at 1/2M and highest when at 1/10K (449.87×). Different benchmarks responded differently to the varied rerandomization rate due to two likely sources. The first is the size of the working set pages for each benchmark, which affects the number of page faults between each rerandomization. The second is the instruction accumulation rate (proportional to the basic block sizes and number of executed instructions), which impacts the number of rerandomizations.

6.5. Real-world application performance

We evaluate the performance of INCOGNITOS with two unmodified real-world applications built for Unikraft [50], the NGINX web server, and Redis in-memory key-value store. The results are presented in Figure 6 and Figure 7.

Experiment setup. For both NGINX and Redis, we applied intensive workloads using each program's benchmark tools. However, no side-channel attack was applied during the benchmark. Also, we enabled INCOGNITOS's adaptive rerandomization with varied $f^{\text{Normal}}_{\text{ReRand}}$. This means that the short-lived rerandomization rate spikes are expected during execution. All throughput measurements are normalized to those of vanilla Unikraft running inside a CVM.

NGINX webserver. We measured the throughput (i.e., HTTPS requests processed per second) of the INCOGNITOS-hosted NGINX with the *Apache Benchmark tool* (`ab`) [60]. The keepalive option (`-k`) was not applied, which makes each request perform a full connection establishment to evaluate the worst-case scenario. We use `ab` to perform 1000 concurrent requests (`-c 8`) on the webserver to retrieve files with varied file sizes: 1KB, 2KB, 4KB, and 8 KB.

Redis in-memory data store. Similarly, we use `redis-benchmark` [61] to benchmark the performance of INCOGNITOS-enabled Redis server. We launched the Redis server with a storage size of 64MB. We filled the server memory usage to its maximum using `LPUSH` for data storage, followed by a `GET` workload. Each workload consisted of 10,000 requests, with payload sizes of 4KB and 8KB. The benchmarking was set to use 50 parallel connections (`-c 50`) with TCP keepalive enabled (`-k`) and randomized access keys (`-r 10000`). We obtained the throughput (commands served per second) and compared it against the throughput of an unmodified Redis workload deployed to Unikraft.

Result discussion:. $f^{\text{Normal}}_{\text{ReRand}}$ and performance trade-off We found that the configurable rerandomization frequency during benign execution $f^{\text{Normal}}_{\text{ReRand}}$ also closely correlated to the performance overheads of real-world applications, similar to the microbenchmark's performance characteristics. Redis and NGINX both saw significant overheads with the smallest $f^{\text{Normal}}_{\text{ReRand}}$ of 1/10K, 42.67× and 44.84× on average for NGINX and Redis. However, with a relaxed $f^{\text{Normal}}_{\text{ReRand}}$ of 1/2M, the performance overheads dropped to $2.45\times$ and $2.46\times$. As shown throughout the security evaluation, this relaxed $f^{\text{Normal}}_{\text{ReRand}}$ would maintain resilience under attacks thanks to IncognitOS's adaptive rerandomization policy. More aggressive values for $f^{\text{Normal}}_{\text{ReRand}}$ may be chosen in anticipation of attacks with very low exit rates.

Result discussion: performance loss from spikes. The momentary benign spikes in VMExit frequencies were sparse throughout the performance evaluation and contributed minimally to the total overheads. In particular, for NGINX and Redis, the *maximum* number of ticks being in an alarmed state observed was 1798 (0.006% of all ticks) during serving 8K files and $1,347,282$ (5.11%) during `LPUSH` with 8K payloads. Note, we used the same $f^{\text{Alarm}}_{\text{VMExit}}$ threshold across the evaluations (0.003), and lower thresholds may be fine-tuned for a specific application to reduce the impacts of spikes further. Based on this result, we conclude that IncognitOS's rerandomization policy effectively suppresses obfuscation overheads.

7. Discussion

This section provides additional discussion of IncognitOS's security and also outlines worthwhile future work.

7.1. Security discussion

We further analyze IncognitOS's security in the following aspects.

Security of IncognitOS operations. Aside from the obfuscated user program and kernel, IncognitOS's operations that achieve the obfuscation must leak no useful information to the adversary. Specifically, here we assess the security of the PF handler and ORAM page pool.

IncognitOS's ORAM page pool includes careful design considerations such that it remains secure even under a powerful adversary's ongoing monitoring. As we explained, the ORAM's sensitive components, the stash and the position map, are implemented to leak no information. IncognitOS's secure stash implementation (§5.2) protects the ORAM stash that the adversary can potentially identify and monitor. Also, IncognitOS's integration of ORAM position map into the GPT ensures position map pages are continuously randomized through GPT rerandomization, rendering side-channel attacks extremely difficult.

In addition, the PF handler itself does not yield any useful information. An adversary monitoring the PF handler's access patterns can only observe three types of information: (1) obfuscated ORAM accesses during page-ins, (2) the GPT levels where page-ins occur during the software GPT walk, and (3) the type of page being paged-in inferred from the destination active region. While (2) could reveal proximity between gVA accesses (e.g., consecutive faults at `0xaa000` and `0xab000` could generate distinguishable page-in patterns), IncognitOS's GPT rerandomization scheme prevents its exploitation. Regarding (3), it is already accessible to the hypervisor through the NPF-controlled channel, as established in §5.1, and thus does not compromise the IncognitOS's security model.

Potential even lower VMExit-rate attacks. Another possible concern is the existence of attacks with even lower VMExit rates that may evade IncognitOS's threshold.

Stealthy attacks are primarily discussed in the context of secret extraction, assuming that the program memory layout is known or profiled [41], [62]. IncognitOS and works on continuous rerandomization [26], [29], [63] can thwart the adversary's attempt on memory layout profiling that must precede the extraction phase. The difficulty of such profiling, not to mention constructing a precise stealthy attack, can be approximated through recent works on practical CVM attack studies [22], [23]. Heckler [22] reported that the authors had to collect more than $100,000$ traces of page accesses with non-deterministic success rates for profiling the target memory pages. Furthermore, even when pessimistically assuming that the adversary somehow achieved profiling within a single rerandomization window, IncognitOS is evaluated to be effective in thwarting the reasonably modeled low-NPF extraction attack (§6.3),

Generalization of such stealthy attacks or predicting future occurrences is difficult. This is because they are inherently specific to the target information in a specific program. The stealthiness of the attack, defined by the number of access traces required for data inference and the noise generated from the relevant computation, cannot be defined deterministically; the worst case may be a single page's activation that leaks the secret. However, we believe that the current VMExit threshold can counter a large class of stealthy attacks based on previous work [41], [62]. Many of the previous stealthy attacks targeted SGX, and the equivalent VMExit rate of similar attacks (if possible) on SEV-SNP is not known. However, these so-called stealthy attacks still require fairly high enclave exit periods (e.g., every 184–482 cycle [41]). To address future stealthy attacks, IncognitOS's configurable policy can be adjusted to strike a balance between security and performance across various scenarios.

Active region size and SEV ciphertext security. Besides the tested attacks, IncognitOS's rerandomization scheme renders exploitation of SEV's deterministic ciphertext issue [21] rather difficult. As explained in §5.1, IncognitOS consciously maintains large (32MB = 8,192 slots) active regions to obtain entropy. Even though IncognitOS always chooses a random slot for new pages during page-ins or rerandomization, the entropy is limited. However, monitoring constantly rerandomized 8,192 pages for appearances of identical ciphertexts would be a daunting task.

We also considered alternative designs, but concluded that our current scheme is sufficient and most practical. For instance, compiler and binary instrumentation for applying *xor* encryption to each sensitive memory load and store has been proposed [27], [64], but show significant overheads and are unlikely to be viable for full-system protection. Another deterministically secure way would be never to reuse a physical page for all pages during rerandomization, but it would be too resource-intensive for the host system.

7.2. Future work

We regard the following extensions to be worthwhile future work that can further improve INCOGNITOS.

Multi-core support. The current prototype only supports single-core computation, which is sufficient to support existing unikernel workloads. Supporting a full *Symmetric Multiprocessing (SMP)* kernel remains a challenge, as Unikraft's SMP implementation is still under active development [65]. A consideration for INCOGNITOS that must accompany the incorporation of multi-core support is the multi-core INCOGNITOS's VMExit sampling issue. Since the VMSA is maintained per core, INCOGNITOS's scheduler must aggregate VMExit occurrences from *all* cores to accurately gauge the CVM's VMExit rate.

Improving unikernel's internal security. The single address space design and missing software attack mitigations are often discussed as the potential security weakness of unikernels [66]. While the issue is orthogonal to our threat model, we expect that INCOGNITOS will also benefit from future security improvements to Unikraft. In fact, Unikraft's community is actively improving its security with defenses such as page table isolation, shadow stack, and CFI [67]. Besides, researchers also investigated lightweight isolation mechanisms like *Memory Protection Keys (MPKs)* in unikernels [66], [68], [69].

8. Additional Related work

We now examine related work on side-channel attacks concerning SEV-SNP CVMs, along with side-channel mitigation approaches that were not discussed in §2.3.

8.1. Side-channel attacks on SEV-SNP

The use of side-channel in previous attacks can be categorized into two main types: (1) identifying sensitive pages or function executions as a preliminary step before initiating more targeted attacks [18], [19], [20], [21], [22], [23], [24], [44], [70], and (2) extracting secrets from the CVM's memory accesses that depend on sensitive information [18], [21], [24], [43]. While most of the SEV-specific vulnerabilities are already addressed in the latest version of SEV-SNP, e.g., [19], [20], [21], [44], [70], their side-channel components remain difficult to mitigate.

Profiling using controlled channel. All existing attacks on SEV and its extensions require a *profiling* phase with the coarse-grained NPF controlled channel (§2.2) to infer locations of sensitive code/data pages within the guest's gPA space. Profiling attacks often undermine KASLR through statistical analysis [18], [20], [22], [23] or by matching page access patterns with known ones [49], [70].

CVM secret extraction. Secret extraction attacks require prior identification of sensitive code and data, allowing the extraction of secret-dependent memory accesses. Previous work discussed secret extraction through fine-grained side channels [18], [21], [24], [43] whose temporal resolution is maximized through the single-stepping technique (§2.2). SEV-Step [24] presents an L2 cache Prime+Probe attack on the AES T-table for key extraction. Li and Wilke et al. [18] enhance the now-patched `CipherLeak` [21] ciphertext attack to perform secret key extraction without relying on the encrypted VMSA. `CipherSteal` [43] infers input data in protected DNN computation using the ciphertext side channel.

8.2. Approaches for side-channel mitigation

Aside from the approaches we discussed in §2.3, there are other proposals for side-channel mitigation, but with certain limitations. Raccoon [71] is the first to discuss obfuscation for side-channel mitigation in the context of SGX by incorporating decoy execution and ORAM, which incurs up to 1000× overheads on simple programs. Shinde et al. [28] propose a deterministic multiplexing that forces sensitive code and data accesses onto a single page but incurs up to 4000× overheads on program execution. SGX-Shield [72] introduces load-time address space randomization for enclaves but would be defeated through online access profiling. OBLIVIATE [73] introduces an obfuscated file system for SGX enclaves. DR.SGX [63] proposes periodic runtime enclave memory rerandomization at the cacheline level. ZeroTrace [74] implements a memory interface that uses ORAM to obfuscate memory accesses. Aside from introducing significant overheads, DR.SGX and ZeroTrace only support the obfuscation of data accesses. Finally, it is unclear how previous systems would be adapted to support CVM full-system memory access obfuscation.

9. Conclusion

This paper introduced INCOGNITOS, a unikernel design for full-system CVM obfuscation that proposes a side-channel defense strategy adapted to the attacker's attack attempts. We explained the design of INCOGNITOS that retrofits the kernel scheduling and paging subsystems. The scheduling subsystem enables robust VMExit rate measurement and adaptive policy enforcement. The paging subsystem enforces memory rerandomization through hardware MMU access and remains transparent to unmodified applications. We performed thorough evaluations of INCOGNITOS security and performance, demonstrating its robustness, resilience against real-world attacks, and efficiency through adaptive rerandomization rate adjustment.

10. Acknowledgements

This work was supported by grants funded by the Korean government: the National Research Foundation of Korea (NRF) grant (NRF-2022R1C1C1010494), The Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2022-II221199, RS-2022-II220688, RS-2024-00437306, RS-2024-00439819, RS-2024-00437849) and The Korea Internet & Security Agency (KISA) grant (1781000009).

References

[1] I. Corporation, “Intel Software Guard Extensions (Intel SGX),” 2024, last accessed March 08, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html

[2] D. Kaplan, “AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More.”

[3] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, “Intel TDX Demystified: A Top-Down Approach,” Mar. 2023. [Online]. Available: http://arxiv.org/abs/2303.15540

[4] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, “Software grand exposure: Sgx cache attacks are practical,” in *Proceedings of the 11th USENIX Conference on Offensive Technologies*, ser. WOOT’17. USA: USENIX Association, 2017, p. 11.

[5] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, “Cache attacks on intel sgx,” in *Proceedings of the 10th European Workshop on Systems Security*, ser. EuroSec’17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3065913.3065915

[6] J. Van Bulck, F. Piessens, and R. Strackx, “Sgx-step: A practical attack framework for precise enclave execution control,” in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, ser. SysTEX’17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3152701.3152706

[7] J. V. Bulck, F. Piessens, and R. Strackx, “Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 178–195. [Online]. Available: https://doi.org/10.1145/3243734.3243822

[8] Y. Xu, W. Cui, and M. Peinado, “Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems,” in *2015 IEEE Symp. Secur. Priv.* San Jose, CA: IEEE, May 2015, pp. 640–656.

[9] M. Hähnel, W. Cui, and M. Peinado, “High-Resolution Side Channels for Untrusted Operating Systems,” in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 299–312. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel

[10] A. Moghimi, G. Irazoqui, and T. Eisenbarth, “Cachezoom: How SGX amplifies the power of cache attacks,” *CoRR*, vol. abs/1703.06986, 2017. [Online]. Available: http://arxiv.org/abs/1703.06986

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

[12] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution,” in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[13] V. Costan and S. Devadas, “Intel SGX Explained,” 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[14] V. Costan, I. Lebedev, and S. Devadas, “Sanctum: Minimal Hardware Extensions for Strong Software Isolation,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[15] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, “Keystone: an open framework for architecting trusted execution environments,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387532

[16] “AMD64 Architecture Programmer’s Manual, Volumes 1-5, 40332, 24592, 24593, 24594, 26568, 26569,” 2024.

[17] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, “Design and Verification of the Arm Confidential Compute Architecture,” in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 465–484. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/li

[18] M. Li, L. Wilke, J. Wichelmann, T. Eisenbarth, R. Teodorescu, and Y. Zhang, “A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP,” in *2022 IEEE Symp. Secur. Priv. SP*, 2022, pp. 337–351.

[19] J. Werner, J. Mason, M. Antonakakis, M. Polychronakis, and F. Monrose, “The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves,” in *Proc. 2019 ACM Asia Conf. Comput. Commun. Secur.*, ser. Asia CCS ’19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 73–85.

[20] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, “SEVered: Subverting AMD’s Virtual Machine Encryption,” in *Proc. 11th Eur. Workshop Syst. Secur.*, ser. EuroSec’18. New York, NY, USA: Association for Computing Machinery, Apr. 2018, pp. 1–6.

[21] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, “CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel,” in *30th USENIX Secur. Symp. USENIX Secur. 21*, 2021, pp. 717–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan

[22] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi, and S. Shinde, “HECKLER: Breaking confidential VMs with malicious interrupts,” in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3459–3476. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/schl{ü}ter

[23] B. Schlüter, S. Sridhara, A. Bertschi, and S. Shinde, “WeSee: Using malicious #vc interrupts to break AMD SEV-SNP,” in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4220–4238.

[24] L. Wilke, J. Wichelmann, A. Rabich, and T. Eisenbarth, “Sev-step: A single-stepping framework for amd-sev,” 2023.

[25] M. Li, Y. Zhang, and Z. Lin, “CrossLine: Breaking "Security-by-Crash" Based Memory Isolation in AMD SEV,” in *Proc. 2021 ACM SIGSAC Conf. Comput. Commun. Secur .*, ser. CCS ’21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2937–2950.

[26] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee, “OBFUSCURO: A Commodity Obfuscation Engine on Intel SGX,” in *Proc. 2019 Netw. Distrib. Syst. Secur. Symp.* San Diego, CA: Internet Society, 2019.

[27] J. Wichelmann, A. Rabich, A. Pätschke, and T. Eisenbarth, “Obelix: Mitigating side-channels through dynamic obfuscation,” in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 189–189. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00182

[28] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, “Preventing Page Faults from Telling Your Secrets,” in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, ser. ASIA CCS ’16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 317–328.

[29] P. Zhang, C. Song, H. Yin, D. Zou, E. Shi, and H. Jin, “Klotski: Efficient Obfuscated Execution against Controlled-Channel Attacks,” in *Proc. Twenty-Fifth Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, ser. ASPLOS ’20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 1263–1276.

[30] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting {SGX} Enclaves from Practical {Side-Channel} Attacks," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 227–240. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/oleksenko

[31] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs," in *Proc. 2017 Netw. Distrib. Syst. Secur. Symp.* San Diego, CA: Internet Society, 2017.

[32] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 7–18. [Online]. Available: https://dl.acm.org/doi/10.1145/3052973.3053007

[33] Y. Fu, E. Bauman, R. Quinonez, and Z. Lin, "Sgx-Lapd: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds., vol. 10453. Cham: Springer International Publishing, 2017, pp. 357–380.

[34] A. Ahmad, B. Ou, C. Liu, X. Zhang, and P. Fonseca, "Veil: A protected services framework for confidential virtual machines," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ser. ASPLOS '23. New York, NY, USA: Association for Computing Machinery, 2024, p. 378–393. [Online]. Available: https://doi.org/10.1145/3623278.3624763

[35] F. Schwarz and C. Rossow, "00SEVen – re-enabling virtual machine forensics: Introspecting confidential VMs using privileged in-VM agents," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1651–1668. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/schwarz

[36] V. Narayanan, C. Carvalho, A. Ruocco, G. Almasi, J. Bottomley, M. Ye, T. Feldman-Fitzthum, D. Buono, H. Franke, and A. Burtsev, "Remote attestation of confidential vms using ephemeral vtpms," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 732–743. [Online]. Available: https://doi.org/10.1145/3627106.3627112

[37] A. Galanou, K. Bindlish, L. Preibsch, Y.-A. Pignolet, C. Fetzer, and R. Kapitza, "Trustworthy confidential virtual machines for the masses," in *Proceedings of the 24th International Middleware Conference*, ser. Middleware '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 316–328. [Online]. Available: https://doi.org/10.1145/3590140.3629124

[38] D. Li, Z. Mi, C. Ji, Y. Tan, B. Zang, H. Guan, and H. Chen, "Bifrost: Analysis and optimization of network I/O tax in confidential virtual machines," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 1–15. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/li-dingji

[39] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazières, and C. Kozyrakis, "Dune: Safe User-level Access to Privileged {CPU } Features," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 335–348. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/belay

[40] S. Kuenzer, V.-A. Bădoiu, H. Lefeuvre, S. Santhanam, A. Jung, G. Gain, C. Soldani, C. Lupu, c. Teodorescu, C. Răducanu, C. Banu, L. Mathy, R. z. Deaconescu, C. Raiciu, and F. Huici, "Unikraft: Fast, specialized unikernels the easy way," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 376–394. [Online]. Available: https://doi.org/10.1145/3447786.3456248

[41] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2421–2434. [Online]. Available: https://doi.org/10.1145/3133956.3134038

[42] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[43] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, " CipherSteal: Stealing Input Data from TEE-Shielded Neural Networks with Ciphertext Side Channels ," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 79–79. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00079

[44] R. Zhang, L. Gerlach, D. Weber, L. Hetterich, Y. Lü, A. Kogler, and M. Schwarz, "CacheWarp: Software-based fault injection using selective state reset," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[45] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *2019 IEEE Symp. Secur. Priv. SP*, May 2019, pp. 1–19.

[46] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[47] The QEMU Project Developers, "'microvm' virtual platform (microvm)," https://www.qemu.org/docs/master/system/i386/microvm.html, (accessed 2024-01-23). [Online]. Available: https://qemu.org

[48] The Unikraft Authors, "Performance - unikraft," https://unikraft.org/docs/concepts/performance, (accessed 2024-01-23). [Online]. Available: https://unikraft.org

[49] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "SEVurity: No Security Without Integrity : Breaking Integrity-Free Memory Encryption with Minimal Assumptions," in *2020 IEEE Symp. Secur. Priv. SP*, 2020, pp. 1483–1496.

[50] The Unikraft Authors, "Pre-built dynamic elfs," (accessed 2024-01-23). [Online]. Available: https://github.com/unikraft/dynamic-apps

[51] "Performance monitor counters for amd family 1ah model 00h- 0fh processors," 2024. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/programmer-references/58550-0.01.pdf

[52] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, jan 1949. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969

[53] G. J. Duck, X. Gao, and A. Roychoudhury, "Binary rewriting without control flow recovery," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 151–163. [Online]. Available: https://doi.org/10.1145/3385412.3385972

[54] B. Chamith, B. J. Svensson, L. Dalessandro, and R. R. Newton, "Instruction punning: lightweight instrumentation for x86-64," *SIGPLAN Not.*, vol. 52, no. 6, p. 320–332, jun 2017. [Online]. Available: https://doi.org/10.1145/3140587.3062344

[55] K.-H. Chow, U. Deshpande, S. Seshadri, and L. Liu, "Deeprest: deep resource estimation for interactive microservices," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 181–198. [Online]. Available: https://doi.org/10.1145/3492321.3519564

[56] S. ur Rehman Baig, W. Iqbal, J. L. Berral, and D. Carrera, “Adaptive sliding windows for improved estimation of data center resource utilization,” *Future Generation Computer Systems*, vol. 104, pp. 212–224, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19309203

[57] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path ORAM: An Extremely Simple Oblivious RAM Protocol,” *J. ACM*, vol. 65, no. 4, pp. 18:1–18:26, Apr. 2018.

[58] U. Maintainers, “Unikraft releases v0.15.0 (Pandora) ,” https://unikraft.org/blog/2023-10-22-unikraft-releases-pandora, 2023, last accessed Jan 14 , 2024,.

[59] Uwe F. Mayer, “Linux/Unix nbench,” https://www.math.utah.edu/~mayer/linux/bmark.html, 2017, last accessed March 08, 2022.

[60] T. A. S. Foundation, “ab - Apache HTTP server benchmarking tool,” https://httpd.apache.org/docs/2.4/programs/ab.html, 2022, last accessed Jan 14 , 2024,.

[61] R. Inc., “Redis benchmark,” https://redis.io/docs/latest/operate/oss_and_stack/management/optimization/benchmarkshttps://redis.io/docs/latest/operate/oss_and_stack/management/optimization/benchmarks/, 2022, last accessed Jan 14 , 2024,.

[62] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, “Telling your secrets without page faults: Stealthy page Table-Based attacks on enclaved execution,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1041–1056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck

[63] F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, and A.-R. Sadeghi, “Dr.sgx: automated and adjustable side-channel protection for sgx using data location randomization,” in *Proceedings of the 35th Annual Computer Security Applications Conference*, ser. ACSAC ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 788–800. [Online]. Available: https://doi.org/10.1145/3359789.3359809

[64] J. Wichelmann, A. Pätschke, L. Wilke, and T. Eisenbarth, “Cipherfix: Mitigating Ciphertext Side-Channel Attacks in Software,” in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 6789–6806. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/wichelmann

[65] The Unikraft Authors, “Smp-safe unikraft core,” (accessed 2024-01-23). [Online]. Available: https://github.com/unikraft/unikraft/issues/587

[66] M. Sung, P. Olivier, S. Lankes, and B. Ravindran, “Intra-unikernel isolation with intel memory protection keys,” in *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 143–156. [Online]. Available: https://doi.org/10.1145/3381052.3381326

[67] U. Maintainers, “Security,” https://unikraft.org/docs/concepts/security, 2023, last accessed Jan 14 , 2024,.

[68] H. Lefeuvre, V.-A. Bădoiu, A. Jung, S. L. Teodorescu, S. Rauch, F. Huici, C. Raiciu, and P. Olivier, “FlexOS: Towards flexible OS isolation,” in *Proc. 27th ACM Int. Conf. Archit. Support Program . Lang. Oper. Syst.*, ser. ASPLOS ’22. New York, NY, USA: Association for Computing Machinery, Feb. 2022, pp. 467–482.

[69] V. A. Sartakov, L. Vilanova, and P. Pietzuch, “CubicleOS: A library OS with software componentisation for practical isolation,” in *Proc. 26th ACM Int. Conf. Archit. Support Program . Lang. Oper. Syst.*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 546–558.

[70] M. Li, Y. Zhang, Z. Lin, and Y. Solihin, “Exploiting Unprotected I/O Operations in AMD’s Secure Encrypted Virtualization,” in *28th USENIX Secur. Symp. USENIX Secur. 19*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1257–1272. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/li-mengyuan

[71] A. Rane, C. Lin, and M. Tiwari, “Raccoon: Closing Digital Side-Channels through Obfuscated Execution,” in *24th USENIX Secur. Symp. USENIX Secur. 15 Wash. DC USA August 12-14 2015*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 431–446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/rane

[72] J. Seo, B. Lee, S. M. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim, “SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs.” in *NDSS*, 2017.

[73] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, “OBLIVIATE: A Data Oblivious Filesystem for Intel SGX,” in *Proc. 2018 Netw. Distrib. Syst. Secur. Symp.* San Diego, CA: Internet Society, 2018.

[74] S. Sasy, S. Gorbunov, and C. W. Fletcher, “ZeroTrace : Oblivious Memory Primitives from Intel SGX,” in *Proc. 2018 Netw. Distrib. Syst. Secur. Symp.* San Diego, CA: Internet Society, 2018.

Appendix A. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper presents INCOGNITOS, a system that obfuscates (at specific learned intervals) the memory of a *Confidential VM (CVM)*. The system leverages *Oblivious RAM (ORAM)*. The challenge is leveraging ORAM in a practical fashion to entire VM executions. To solve these, the paper leverages a unikernel design (based on Unikraft) and an adaptive exit-rate-based algorithm. The system incurs 2-4 times overhead during execution of real-world programs.

A.2. Scientific Contributions

- Provides a valuable step forward in an established field
- Creates a new tool to enable future science
- Establishes a new research direction

A.3. Reasons for Acceptance

1) Side-channels remain a problem for CVMs. The reviewers appreciated how the paper positioned itself in the domain of side-channel mitigations for existing TEEs like SGX, especially ones that have leveraged ORAM. The system designed by INCOGNITOS helps advance our understanding of ORAM's application into whole VM TEEs.
2) The system designed in this paper can help other researchers develop ORAM-specific mitigations for CVMs.

A.4. Noteworthy Concerns

The paper leaves core features (including multi-threading support and defense against other side-channels like cache) as part of future work or out-of-scope.