PLATYPUS: Software-based Power Side-Channel Attacks on x86

Moritz Lipp*, Andreas Kogler*, David Oswald†, Michael Schwarz‡,
Catherine Easdon*, Claudio Canella*, and Daniel Gruss*

*Graz University of Technology †University of Birmingham, UK
‡CISPA Helmholtz Center for Information Security

***Abstract*—Power side-channel attacks exploit variations in power consumption to extract secrets from a device, e.g., cryptographic keys. Prior attacks typically required physical access to the target device and specialized equipment such as probes and a high-resolution oscilloscope.**

In this paper, we present PLATYPUS attacks which are novel software-based power side-channel attacks on Intel server, desktop, and laptop CPUs. We exploit unprivileged access to the Intel Running Average Power Limit (RAPL) interface that exposes values directly correlated with power consumption, forming a low-resolution side channel.

We show that with sufficient statistical evaluation, we can observe variations in power consumption, which distinguish different instructions and different Hamming weights of operands and memory loads. This enables us to not only monitor the control flow of applications but also to infer data and extract cryptographic keys. We demonstrate how an unprivileged attacker can leak AES-NI keys from Intel SGX and the Linux kernel, break kernel address-space layout randomization (KASLR), infer secret instruction streams, and establish a timing-independent covert channel. We also present a privileged attack on mbed TLS, utilizing precise execution control to recover RSA keys from an SGX enclave. We discuss countermeasures and show that mitigating these attacks in a privileged context is not trivial.

I. Introduction

The concept of extracting data from a computer system by monitoring side-channel information, such as its power consumption or electromagnetic emissions, is known since World War II [3]. Power analysis attacks were first presented in an academic context by Kocher et al. [50] for attacks on cryptographic implementations in smart cards. Subsequent research applied these attacks to different devices and algorithms, particularly to supposedly side-channel-resistant encryption-scheme implementations [24], [26]. However, until recently, power analysis attacks had two limitations. First, they primarily targeted small embedded microcontrollers rather than more complex high-performance desktop and server CPUs. Second, software-based attacks relying on the available interfaces [56], [68], [87] were so far not successfully applied on x86 to leak fine-grained information, e.g., cryptographic key bits.

Software-based power side-channel attacks have been demonstrated on mobile devices for website [68] and app fingerprinting [87], UI inference [87], password length guessing [87], and geolocation estimation [87]. More recently, O'Flynn [64] recovered secrets processed in the secure world on an ARM TrustZone-M platform using an onboard ADC, and Mantel et al. [56] distinguished different RSA keys by measuring the power consumption on Intel desktop machines. The experimental results of Mantel et al. on RSA demonstrated that certain multiply operations of the square-and-multiply implementation can be detected, but no full key recovery was achieved. Similarly, Fusi [20] tried to recover RSA-16384 keys but concluded that the sampling rate of the interface is too low to mount an attack.

In this work, we present PLATYPUS[1] attacks which are novel software-based power side-channel attacks on Intel servers, desktops, and laptops by abusing unprivileged access to Intel's RAPL interface. By observing changes in power consumption with a resolution of up to $20\,\mathrm{kHz}$, we show that different executed instructions and features of their operands can be distinguished. Furthermore, we observe that when a register is filled with data from a cache line, the Hamming weight, *i.e.*, the number of bits set to one, of the loaded value measurably influences the power consumption. We show how these power differences between different operands and load values enable the inference of inputs and intermediate values used for multiplications or masks in an encryption algorithm. We present the building blocks to enable the creation of power traces at instruction-level granularity and develop novel techniques for RAPL power analysis attacks on enclaved and non-enclaved execution.

To demonstrate the applicability of these attacks, we successfully recover AES-NI keys from an SGX enclave and the Linux kernel in 26 hours. In a privileged attack context, we recover RSA private keys from mbed TLS within 100 minutes by inferring the instructions executed inside SGX from a power trace with instruction-level granularity. We derandomize the kernel address space within 20 seconds by observing that accesses to valid and invalid kernel addresses from user space expose a different power consumption footprint. Furthermore, we demonstrate that RAPL enables victims to be observed at sub-cache-line granularity, and use this to establish a timing-independent covert channel with a transmission rate of $18.7\,\mathrm{bit/s}$. While an unprivileged attack can be prevented by restricting access to the interface, mitigating privileged

[1] Power Leakage Attacks: Targeting Your Protected User Secrets

attacks is not trivial. We discuss different countermeasures and mitigation strategies for the presented attacks.

To summarize, we make the following contributions:

1) We improve software-based power side-channel attacks to distinguish instructions, operands, and data.
2) We show that the RAPL interface provides sufficient resolution for practical attacks on Intel CPUs.
3) We demonstrate an attack on a cryptographic implementation running in Intel SGX, recovering RSA private keys from mbed TLS within 100 minutes.
4) We show that an unprivileged attacker can use Correlation Power Analysis to recover keys from an AES-NI implementation in an SGX enclave and the Linux kernel within 26 hours (when minimal I/O noise is present) to 277 hours (under real-world conditions).
5) We break kernel address space layout randomization (KASLR) from user space within 20 seconds, observe intra-cache-line accesses, and demonstrate a timing-independent covert channel.

Responsible Disclosure: We responsibly disclosed our findings to Intel on November 16th, 2019. Intel acknowledged our findings and verified our experiments. The issues are tracked under CVE-2020-8694 and CVE-2020-8695 and are held under embargo until November 10th, 2020. We responsibly disclosed our findings to AMD on June 6th, 2020.

Outline: Section II provides background. Section III analyzes the information leakage induced by the Intel RAPL interface. Section IV presents the threat model, attack overview, and building blocks. Section V evaluates these building blocks and constructs concrete attacks with them. Countermeasures and related work are discussed in Section VI and Section VII, respectively. We conclude in Section VIII.

II. Background

In this section, we provide background on power analysis, Intel RAPL, and Intel SGX.

A. Power Analysis

Power analysis attacks are built upon the observation that the power consumption of CMOS digital circuits is data-dependent *by design*. Each bit flip requires one or more voltage transitions from 0 to high (or vice versa). Different data values typically entail differing numbers of bit flips and therefore produce distinct power traces. Equation (1) presents the primary sources of power consumption, where α is the probability of a voltage transition, C is the load capacitance, V_{dd} is the supply voltage, F is the clock frequency, I_{sc} is the short-circuit current (when NMOS and PMOS transistors are active simultaneously) and I_{leak} is the leakage current [14].

$$\begin{aligned} P &= (P_{switching}) + (P_{short-circuit} + P_{leakage}) \\ &= \alpha \cdot C \cdot V_{dd}^2 \cdot F + I_{sc} \cdot V_{dd} + I_{leak} \cdot V_{dd} \end{aligned} \tag{1}$$

Crucially, $P_{switching}$ with its data-dependent α value is significantly larger than the other terms. Therefore, any circuit not explicitly designed to be resistant to power attacks has data-dependent power consumption. However, in a complex circuit, the differences can be so slight that they are difficult to distinguish from a single trace, particularly if an attacker's sampling rate is limited. Therefore, it is necessary to use statistical techniques such as Differential Power Analysis and Correlation Power Analysis across multiple power traces.

Simple Power Analysis (SPA): In SPA attacks [50], secret-dependent power consumption differences during an operation, e.g., a cryptographic signature computation, are directly analyzed from power traces to determine the underlying secret. For example, there may be a detectable spike in power consumption when the key bit multiplied is 1 versus when it is 0 because the implementation executes a different instruction sequence in each case. Using SPA, the secret can be extracted with only a small number of traces. However, this is only possible if the secret has a significant impact on the power consumption of the device, and the traces are relatively noise-free. Noise can be averaged out by aligning the traces and computing the mean of the collected traces.

Differential Power Analysis (DPA) and Correlation Power Analysis (CPA): DPA attacks [50] are based on a statistical analysis of a large number of traces with varying input data. Rather than analyzing individual power traces along the time axis as in a typical SPA attack, DPA analyzes how the power consumption at fixed moments in time is a function of the secret data being processed [55]. DPA is significantly more powerful than SPA, as small secret-dependent biases can be detected even in the presence of noise. In our measurement context for power attacks against the CPU, this is relevant for the analysis of operand-dependent power consumption, as these differences are much smaller than the power differences between instructions and can be hidden by measurement error and noise. However, using DPA, these differences can still be identified and used to recover the underlying secret data. CPA [11] is an extension of DPA, which examines the correlation between variations in the set of traces and a leakage model depending on the value of intermediate values [49]. We further explain the inner workings of CPA in Section V-B.

B. Intel RAPL

The Intel Running Average Power Limit (RAPL) mechanism was introduced with the Sandy Bridge microarchitecture to ensure the CPU remains within desired thermal and power constraints [27]. Since Haswell, it has provided three distinct capabilities for controlling average power over timescales of multiple seconds, ~10 ms, and <10 ms (PL1, PL2, and PL3, respectively). These three control loops dynamically adjust the CPU frequency to maximize performance while ensuring the running power average is within each of their (configurable) limits. By design, this modifies the voltage and power consumption. To implement these control loops, it is necessary to provide power-measurement feedback [27]. This can be done with an analog circuit, e.g., voltage regulator current monitoring, or by estimating the energy consumption in the core, as done in Sandy Bridge and Ivy Bridge [27].

Intel defines four different domains for RAPL [40]: package (*PKG*), power planes (*PP0* and *PP1*), and *DRAM*. The package

domain estimates energy consumption for the entire socket. PP0 contains the energy consumption estimates of the cores while, on client systems, the PP1 domain refers to a specific device's power plane in the uncore. In this work, PP0 is subsequently referred to as the core domain. On Skylake, Intel has introduced the `PSys` domain covering the entire SoC.

Intel CPUs also provide other functionality for dynamic frequency and voltage scaling (DFVS). For example, they support configurable processor performance states (P-states), as defined in the Advanced Configuration and Power Interface (ACPI) specification [78]. Each state specifies a frequency and voltage operating point [16]. When enabled, the Intel Turbo Boost feature adjusts each core's P-state automatically.

C. Intel SGX

Intel SGX (Software Guard Extensions) is an instruction set extension that provides a mechanism for confidentially executing code on a system, isolated from other software on the CPU [40]. The SGX threat model assumes that even privileged software such as the operating system, administrative users, and peripheral hardware may be compromised and behave maliciously. An application using SGX is split into two distinct parts, an untrusted part (which launches enclaves as needed to process secrets) and a trusted part (within an enclave). Each enclave operates within an encrypted and isolated memory region to protect application secrets from hardware attackers. As neither the operating system nor any other application is trusted under the SGX threat model, the processor guarantees that the enclave's memory cannot be accessed by anything but the enclave itself. Additionally, encryption ensures that enclave memory cannot be read directly from the DRAM module, as even peripheral hardware may be malicious. Intel generally considers physical side-channel attacks on SGX out of scope. Side channels [9], [73], race conditions [85], [72], and memory-safety violations [51] are not in the threat model, and it is the developer's responsibility to defend against these.

III. Intel RAPL Leakage Analysis

In this section, we analyze the power side-channel information leakage from Intel RAPL data, considering both user-space and SGX-enclave targets. We experimentally evaluate that we can distinguish and fingerprint both individual instructions (Section III-C) and the influence of their operand values (Section III-D). Furthermore, we evaluate the influence of concrete data values on energy consumption (Section III-E) as well as the influence of the cache status of a memory address in a load operation (Section III-F).

While energy-consumption interfaces also exist on non-Intel CPUs, we focus on Intel's RAPL implementation and briefly discuss other architectures in Section VII-B.

Note that while we primarily refer to runtime *energy* consumption rather than *power* consumption throughout this work, these are directly related, as $power = energy \div time$.

TABLE I: RAPL register update intervals if accessed directly in the kernel or via the *powercap* driver.

Register	Measurement Unit	Kernel	Driver
MSR_PKG_ENERGY_STATUS	μJ	1000 µs	1000 µs
MSR_DRAM_ENERGY_STATUS	μJ	1000 µs	1000 µs
MSR_PP0_ENERGY_STATUS	μJ	50 µs	50 µs
MSR_PERF_STATUS (core voltage)	V	150 µs	-

A. RAPL Interface

RAPL provides an interface both for controlling the core frequency and voltage and for monitoring the power consumption of the socket and memory domain (see Section II-B). To date, Intel RAPL has typically been used to model energy consumption on a system level [67] or in benchmarks [47].

We can read the RAPL register values to measure energy consumption, *i.e.*, the cumulative power consumption over a sampling period, in two ways:

- **Unprivileged Access**: On Linux, the power capping framework `powercap` provides unprivileged access to Intel RAPL by exposing the MSRs through the `sysfs` interface. This allows an unprivileged attacker to directly read the value of the individual packages from a file located in the `/sys/devices/virtual/powercap` tree.
- **Privileged Access**: A privileged attacker targeting Intel SGX can load a kernel module to read the RAPL MSRs.

While measuring the update intervals of the values provided by both the Linux RAPL user-space driver and by accessing the MSRs directly, we observed that several values update faster than the documented RAPL update rate of 1 ms. We observe that the `MSR_PP0_ENERGY_STATUS` (core energy consumption) and `MSR_PERF_STATUS` (core voltage) values update substantially faster, at 50 µs and 150 µs intervals, respectively. The results of this evaluation are shown in Table I. These rates were consistent across the different tested microarchitectures.

B. Experimental Setup

Throughout this work, we tested on Intel mobile, desktop, and server CPUs. Table II provides details of each Intel CPU used in our experiments. In the mobile setting, we tested on a Lenovo Thinkpad T480s and T495s, both using Core i7-8650U CPUs, on a Lenovo Thinkpad T460s with a Core i7-6600U and an Intel NUC7I3BNH using a Core i3-7100U. For the desktop setting, we evaluated a system using a Core i5-3230M, a Core i7-6700K, and a Core i9-9900K. Finally, for the cloud server setting, we evaluated 3 systems, with Xeon E3-1240 v5, Xeon E3-1275 v5, and Xeon Silver 4214 CPUs. All tested devices run Ubuntu Linux, with versions from Ubuntu 16.04 to Ubuntu 20.04, and kernels 4.15.0 to 5.4.0. Different Ubuntu versions and kernels did not appear to influence the results, and we would only expect this to occur if there were a substantial update to the behavior of the `powercap` driver.

Unless stated otherwise, all systems were used using the default system configuration, and all mobile systems were connected to an AC power source. For example, we did not fix the CPU frequency or disable Intel Turbo Boost.

TABLE II: CPU type, model, and microarchitecture for each device under test, and whether it leaks data of operands, type of instructions, and the target of memory loads.

Type	CPU	Microarchitecture	Leakage		
			Data	Instr.	Target
Mobile	Core i7-6600U	Skylake	✓	✓	✓
Mobile	Core i3-7100U	Kaby Lake	✓	✓	✓
Mobile	Core i7-8650U	Kaby Lake-R	✓	✓	✓
Desktop	Core i5-3230M	Ivy Bridge	✗	✓	✓
Desktop	Core i7-6700K	Skylake-S	✓	✓	✓
Desktop	Core i9-9990K	Coffee Lake-R	✓	✓	✓
Server	Xeon E3-1240 v5	Skylake	✓	✓	✓
Server	Xeon E3-1275 v5	Skylake	✓	✓	✓
Server	Xeon Silver 4214	Cascade Lake	✓	✓	✓

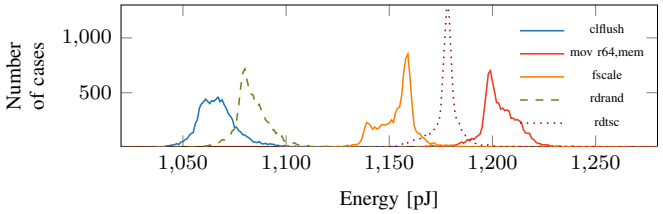

Fig. 1: A histogram of the power consumption of various instructions on the i7-6700K (desktop) system.

TABLE III: Average observed energy consumption (package domain) of different instructions on our i7-6700K (desktop), Xeon Silver 4214 (server), and i7-8650U (mobile) systems.

Instruction	Xeon Silver 4214	i7-6700K	i7-8650U
`nop`	0.1795 nJ	0.1189 nJ	0.0843 nJ
`inc r64`	0.1795 nJ	0.1208 nJ	0.0858 nJ
`xor r64, r64`	0.1795 nJ	0.1209 nJ	0.0849 nJ
`mov r64, mem`	0.1868 nJ	0.1247 nJ	0.0840 nJ
`imul r64, r64`	0.1798 nJ	0.1169 nJ	0.0887 nJ
`fscale`	0.1867 nJ	0.1182 nJ	0.0877 nJ
`rdrand r64`	0.1797 nJ	0.1129 nJ	0.0982 nJ
`rdtsc`	0.1864 nJ	0.1189 nJ	0.0848 nJ
`clflush mem`	0.1865 nJ	0.1129 nJ	0.1018 nJ
`aesenc xmm, xmm`	0.1794 nJ	0.1188 nJ	0.0946 nJ

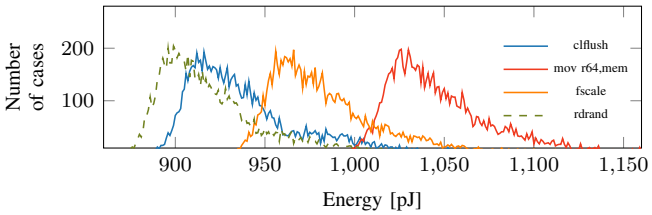

Fig. 2: A histogram of the power consumption of various instructions inside an SGX enclave on our i7-8650U (mobile).

C. Distinguishing Instructions

With our first experiment, we demonstrate that Intel's RAPL interface enables distinguishing different instructions via their energy consumption. To measure the energy consumption of an instruction, we record its energy consumption over 10 000 consecutive executions and take the median value to eliminate system-level noise, e.g., erroneous high values caused by interrupt handling or the process being descheduled. We observe the energy consumption across the entire CPU package to ensure that non-core activity, e.g., DRAM access, is included.

Table III lists the measured energy consumption of different instructions on our i7-6700K (desktop), Xeon Silver 4214 (server), and i7-8650U (mobile) systems. We can clearly observe inter-instruction differences in energy consumption. This enables an attacker to identify which instructions are executed, provided they can profile the energy consumption of the victim microarchitecture. For instance, the `rdtsc` instruction consumes 0.1189 nJ on the i7-6700K, versus 0.1864 nJ on the Xeon Silver 4214 and 0.0848 nJ on the i7-8650U. As illustrated in Figure 1, this clearly distinguishes it from `rdrand` and `clflush`, which have much lower average energy consumption. However, as some instructions have similar energy consumption, this method may identify multiple instruction candidates. For example, on the Xeon Silver 4214, `nop`, `inc`, and `xor` are indistinguishable at this measurement granularity. While the table only shows the values for when the mobile system (i7-8650U) is connected to an AC power source, we also observed these differences when running on battery power. As not every instruction sequence has the same probability, it may be possible to recover individual instructions using heuristics for typical instruction sequences, or by leveraging existing research regarding distinguishing x86 code sequences from data bytes [84].

These results align with those of prior work, in which the different energy consumption of instructions was identified using either Intel RAPL [36], [20], [59], [31] or dedicated hardware [77], [74], [7], [82].

Differing power consumption can also be observed for instructions executed inside SGX enclaves, as shown in Figure 2. The enclave's isolation is no protection here: just like with execution outside the enclave, instructions can be clearly distinguished. Interestingly, energy consumption for the `clflush` instruction is higher inside an SGX enclave, which we attribute to the transparent memory encryption. With other instructions, we do not observe such a difference.

D. Distinguishing Operands

In addition to the energy-consumption differences of instructions, the energy consumption of some instructions further depends on their operand value. Intuitively, e.g., integer multiplication should use more energy if more operand bits are set. We measure the `imul` instruction with different operand values in user space on our Xeon E3-1240 v5 system with a fixed core frequency. For the 64-bit operand, we used Hamming weights of 0, 16 (a quarter of the bits), 32 (half of the bits), 48 (three-quarters of all the bits), and 64 (all of the bits). The second operand remains fixed to the value 8. In Figure 3, it can be seen that the power consumption differs based on the Hamming weight. While we cannot deduce the exact value of the operand, it reduces the range of potential values, and it can be used in CPA attacks (cf. Section V-B).

The distinction is not limited to the `imul` instruction. Figure 4, for example, shows the differences in power consump-

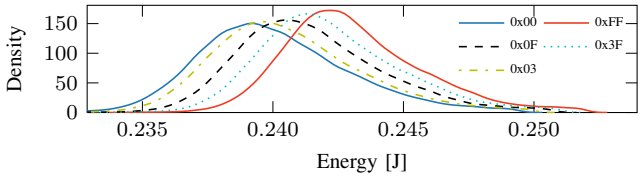

Fig. 3: Measured energy consumption of the `imul` instruction with one operand fixed to 8 and the other varying in its Hamming weight.

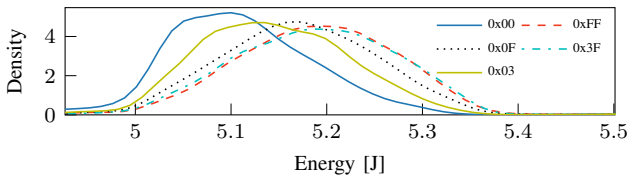

Fig. 4: Measured energy consumption of the `shr` instruction with a register set to different Hamming weights.

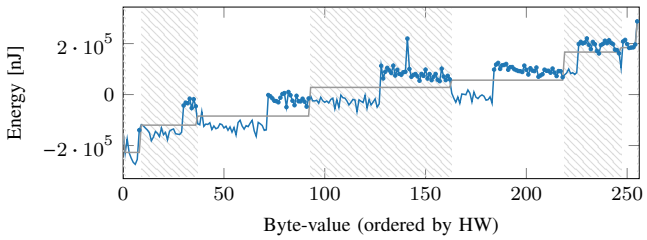

Fig. 5: Energy consumption of the `movb` instruction for all byte values, ordered by Hamming Weight (HW) and value. The circle marks values where the most-significant bit is set.

tion for `shr` on our i7-8650U system with a clear difference in power consumption depending on the Hamming weight of the shifted register. We reproduced these results on an i7-6600U, i7-6700K, and i9-9900K and Xeon 4214 CPU. For the `vpand` instruction, the distributions of the energy consumption differs if one of the operands is zero or not. Ivy Bridge and Sandy Bridge estimate the power consumption [27] and do not rely on hardware probes. Thus, we cannot distinguish operands and data, as we verified on an i5-3230M (cf. Table II).

E. Distinguishing Data

We showed that it is possible to fingerprint different instructions and the Hamming weight of their operands. In the third experiment, we evaluate the influence of data values loaded from the cache on the energy consumption. We set up a cache line with alternating 1 and 0 bits to achieve an even Hamming weight. We then set the value of the first byte in the cache line and measure the energy consumption of a memory load of that specific byte, using the `movb` instruction for all 256 value possibilities. To prevent a possible measurement side-effect introduced by the order of the different values measured, we set the value in a pseudo-random order.

We performed the experiment on our Intel Xeon E3-1240 v5 (server) system, collecting measurements for all possible byte values for 627 hours. While the obtained measurements show a trend of increasing energy consumption with increasing value, a power model was not observable. When sorting the values based on their Hamming weight and value, as illustrated in Figure 5, the increasing power consumption is clearly visible. However, one can measure a different power consumption within values of the same Hamming weight (separated in the plot by the white background or gray pattern). These spikes correlate to exactly those values where the most-significant bit is set (data points with circle marks).

To verify the results on other microarchitectures, we performed a reduced experiment with fewer different Hamming Weights (Section III-D). On the i7-6600U (mobile) system set to a fixed frequency, we observed a similar increasing energy consumption with the Hamming Weight of the byte being read after measuring for 5 minutes.

While we cannot deduce the exact data value that is loaded, one can clearly infer information about the Hamming weight and whether the most-significant bit is set by measuring its energy consumption. Similarly to the varying power consumption, we observed with instruction operands, this allows us to constrain the range of potential values.

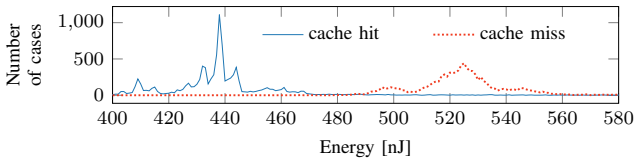

Fig. 6: Using RAPL to distinguish whether the target of a memory load is cached (cache hit) or not (DRAM access).

F. Distinguishing Load Targets

To get an even finer granularity when distinguishing instructions, we demonstrate further that it is possible to distinguish the cache status of a load destination. When a memory load accesses data that is already cached, DRAM consumes significantly less energy than when a data access misses the cache and must be first fetched from the main memory.

We evaluated this experiment on several CPUs, as shown in Table II. Figure 6 shows a histogram of data fetched from the cache and DRAM on our i7-8650U (mobile) system. When recording power consumption using RAPL on the DRAM domain, there is a clear difference in power consumption for cache hits and cache misses, both when connected to a power supply and when running on battery. Hence, code sequences which are vulnerable to cache attacks can also be exploited using power measurements. This allows an attacker to build a timer-free cache attack, similar to the timer-free attacks presented by Diesselkoen et al. [18] and Gruss et al. [28].

IV. Attack Overview & Building Blocks

In this section, we introduce the basic concept of PLATYPUS attacks based on the observations from Section III. We describe the necessary building blocks and their applicability

in various scenarios and attacker models before demonstrating several attacks in Section V.

A. Attack Scenarios & Attacker Model

We consider two different attacker models for our attacks, namely an unprivileged user-space attacker and a privileged kernel-space attacker. For all our attacks, we assume native code execution on an Intel CPU and no software bugs or hardware vulnerabilities.

Unprivileged User-space Attacker: A user-space attacker can run native *unprivileged* code. Hence, the user-space attacker only has access to power interfaces provided by kernel drivers, e.g., the RAPL `sysfs` interface from `powercap`. In addition, the user-space attacker can communicate with other interfaces, e.g., `ioctl`, to the kernel, and interfaces exposed by other applications, e.g., sockets. Furthermore, the user-space attacker could, to some extent, influence other running applications, e.g., by attempting to slow down another process by exhausting its resources [4].

Privileged Kernel-space Attacker: The kernel-space attacker can execute native *privileged* code. Hence, the kernel space has direct access to Intel RAPL's MSRs. The privileged kernel-space attacker has full control over the operating system and, thus, direct access to the memory of running applications. Therefore, we assume an attack setting on SGX enclaves (see Section II-C) where the memory is encrypted and cannot be inspected by the operating system. For the SGX enclave, a malicious operating system is in the threat model [17].

B. Building Blocks

In this section, we describe the necessary building blocks. We describe how a privileged attacker can achieve precise execution control, enabling them to overcome the low sampling rate faced by an unprivileged attacker. We characterize the documented power interfaces we use for our attacks.

1) Power Information: To mount PLATYPUS attacks, it is necessary to obtain a power consumption measurement within the software. While throughout this work, we focus on Intel RAPL, these attacks are, in general, not restricted to the Intel platform. We discuss other microarchitectures and interfaces in Section VII-B.

One inherent challenge of software-based power analysis is the low update rate of power data sources in contrast to the frequency of the execution stream under attack (see Section V). When attempting to reconstruct a signal, it is crucial to sample at a sufficiently high rate. While measuring the `PP0` MSR directly from kernel space, the sample rate is a bit higher; it is still suboptimal. For other attacks, the relevant values are from other domains, e.g., `PKG` and `DRAM`, which do update at the documented slower rate (e.g., Section III-C).

In general, undersampling means that we cannot obtain samples at a sufficient number of points over the time axis, e.g., because the time axis is very short when sampling only for a few nanoseconds. However, if the attacker can conduct repeated attacks, then multiple traces can be combined to recover an averaged but more complete trace.

Moreover, note that Intel RAPL does not provide the energy consumption per core but per processor package. Thus, code executed on other cores have a direct influence on the measurement of a specific piece of code running on one core and, thus, the number of overall measurements increases to average out the noise introduced by the other cores. In the case of a privileged attacker, the noise introduced by other cores can be limited as the attacker can disable them or control what code is executed on which core. In contrast, AMD's implementation of RAPL provides per-core counters (cf. Section VII-B).

Note that while factors such as frequency and P-state do influence the raw energy consumption values measured, it is not necessary to fix them, as the data-dependent differences which our attacks exploit remain observable.

2) Alignment and Execution Control: In the attack scenario where the attacker measures power consumption in parallel to the victim's execution, the attacker needs to align the recorded traces. The trace needs to contain a distinctive feature, e.g., a distinct peak in power consumption, so that traces can be shifted into alignment with each other. While a privileged attacker can precisely control the victim's execution and interrupt it at will, an unprivileged attacker cannot. However, if the attacker can control when the execution of the attacked code begins, or use a trigger signal such as a cache-based side channel [72], then the collected traces can be aligned based on that timing information.

Precise execution control is the capability to control the victim's execution at instruction-level granularity. To achieve precise execution control of SGX enclaves, we repurpose previously published techniques for microarchitectural attacks and apply them in our software-based power analysis attack.

Single-Stepping: With SGX-Step, Van Bulck et al. [81] introduced the concept of single-stepping SGX enclaves. They achieve this by configuring the local APIC timer interrupt interval so that the interrupt arrives during execution of the first instruction after `eresume`. This triggers an Asynchronous Enclave Exit and execution of an attacker-controlled interrupt handler, where attack-specific code can be executed. This process can be repeated, resuming the enclave to execute precisely one instruction each time. The SGX-Step framework enables these APIC timer interrupts to be configured from user space, along with user-space modification of page table entries. Single-stepping has since been used in a range of microarchitectural attacks. For example, it was used in the Foreshadow attack [79] to extract key material from SGX enclaves to bypass enclave launch control and to forge local and remote attestation. It was further used with LVI [80] to mount a transient fault attack on AES-NI.

Zero-Stepping: If the local APIC timer is configured such that the interrupt arrives within `eresume`, the enclave instruction pointer will not advance, and so a single instruction can be repeatedly executed for measurements. Zero-stepping can also be achieved by revoking the execute permissions of an enclave's code pages triggering a page fault on the first instruction after `eresume`. Thus, no enclave instruction is actually executed [81]. MicroScope [75] provides an additional

technique to replay an enclave instruction repeatedly using a memory access instruction triggering the page fault handler as a *replay handle*.

Zero-stepping provides us with a powerful attack primitive to measure the power consumption of a single instruction repeatedly. We can advance to the desired instruction using single-stepping as described above, and then sample the instruction an arbitrary number of times with zero-stepping. Crucially, it enables us to take this arbitrary number of samples *even if we are only able to trigger a single execution of the algorithm under attack in the enclave*. Taking a large number of samples in this way allows to overcome the limited sampling rate and resolution of RAPL.

V. Evaluation

In this section, we combine our attack primitives to build concrete PLATYPUS attacks. We demonstrate that we can recover an RSA key used inside an SGX enclave using mbed TLS (Section V-A). We use CPA attacks to extract AES keys from the Linux kernel and from an SGX enclave, both utilizing the AES-NI instruction extension (Section V-B). Furthermore, we exploit Intel RAPL to observe victims at sub-cache-line granularity (Section V-C), to derandomize the kernel address space (Section V-D), and to establish a timing-independent covert channel (Section V-E).

A. RSA Key Recovery

In this attack scenario, we consider a privileged attacker targeting an Intel SGX enclave performing RSA signatures. As the threat model of SGX considers the operating system to be untrusted, the attacker is allowed to load arbitrary kernel modules. We consider two different target implementations. First, we will show a toy example imitating a square-and-always-multiply RSA implementation that allows to visually illustrate the leakage observable through the RAPL domain and the core voltage using precise execution control. Second, we will demonstrate an attack on mbed TLS [5] to extract RSA private keys from the SGX enclave. Further, we will discuss scenarios where the code executed within the enclave is unknown as well as scenarios where the implementation of the enclave is known by the attacker, thus enabling the attacker to target specific instructions within the enclave's execution.

Setup: In our experiment, the victim provides an API for signing or decrypting user-provided data inside an SGX enclave, making it secure against direct attacks from the operating system, other enclaves, and user space. For simplification and evaluation purposes, we first imitate a square-and-always-multiply RSA implementation that performs the same instructions with different operands based on the value of the currently processed key bit. In our second scenario, we attack the RSA implementation of mbed TLS inside an enclave.

We use SGX-Step [81] and hook the local APIC interrupt `apic_irq` handler to record the values of the timestamp counter `TSC`, the current energy consumption for the desired domain (`MSR_PP0_ENERGY_STATUS`), and the current P-state and core voltage (`MSR_PERF_STATUS`). Further, we hook the Asynchronous Exit Pointer (AEP) to decide if we want to zero-step the current instruction or advance to the next instruction (single-step), as described in Section IV-B2.

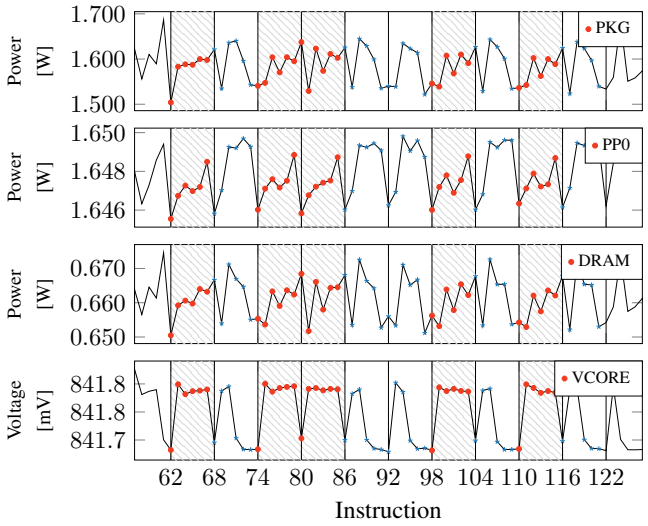

Fig. 7: Energy consumption and core voltage per instruction of a victim enclave. The attacker uses single-, and zero-stepping to precisely measure single instructions of the victim, allowing to distinguish between them to leak the single key bits. Highlighted areas with red markers indicate a 1-bit, blue markers indicate a 0-bit.

1) Toy Example: Our toy implementation is constant-time, the number of instructions executed is independent of the bit processed. The key insight here is that even an implementation with these defensive properties against side-channel attacks can still be successfully attacked via the RAPL power side-channel. Specifically, for a 1 bit, we execute two `vpmuludq` instructions, one for a square operation and one for multiplication. For a 0 bit, we execute a `vpmuldq` instruction for the square operation and an additional one using a dummy output register with no architectural effect.

Evaluation: We evaluated this attack scenario on our Xeon E3-1275 v5 (server) system and the i9-9000K (desktop) system. For each execution run of the victim, we single-step to each instruction and measure it over 188 zero steps, *i.e.*, the number of zero steps that need to be executed such that the RAPL counter is updated. We measured over 96 000 execution runs, yielding an overall attack time of 8.11 h on the E3-1275 v5. The result is illustrated in Figure 7. One cannot only clearly see the difference in power consumption for every instruction measured, but also distinguish whether the key bit was set to 1 (highlighted areas with red markers) or 0 (areas with blue markers) by examining the instructions depending on the key bit. This allows recovering the secret key successfully. Furthermore, as shown in Figure 7, these differences are not only clearly visible in the different RAPL domains (package, PP0, DRAM) but also in the core voltage.

Under the assumption that the attacker knows which set of instructions needs to be sampled for each key bit, the attacker

does not need to zero-step every single instruction. In our example, it would be sufficient just to sample every seventh instruction to recover every single key bit. Even if different instructions are executed depending on the key-bit value, the attacker can advance directly to the instruction responsible for the next key bit after recovering the current key-bit value. To correctly distinguish between these two instructions, we require at least 350 measurements over 255 zero steps when observing the core voltage to recover 99.4 % of the key bits correctly. For the different RAPL domains, we require more traces, e.g., at least 40 000 traces over 188 zero steps to recover 99.5 % of the key bits. Thus, with a runtime of 1.35 ms per trace for each key bit, a 2048-bit RSA attack can be successfully recovered within 16.5 minutes when observing the core voltage. With RAPL and a runtime of 0.99 ms per trace for each key bit, we can successfully recover the key within 23.3 hours. This number highly depends on how many measurements are required to distinguish both cases with a high probability and, thus, can be different in other scenarios.

2) Attack on mbed TLS: In our second scenario, we extract RSA keys from the mbed TLS [5] (version 2.13.0) implementation with a fixed window length of 1 (`MBEDTLS_MPI_WINDOW_SIZE 1`). In order to distinguish the key bits, we do not directly target the branch instruction of the fixed-window exponentiation. Instead, we aim at an instruction with a more distinct energy consumption inside the branch. In SGX, Intel's `fast_memset` implementation replaces the standard libc `memset` implementation called inside the `mpi_montmul` function with AVX instructions. AVX instructions are located at a given offset from the branch instruction if the key bit is set. If the key bit is 0, a different (non-AVX) instruction is executed with the same instruction offset. Thus, we can directly reconstruct the key bit by measuring the energy consumption at the instruction executed with the instruction offset after the branch.

However, the implementation of mbed TLS skips leading zeroes of the exponent and, therefore, has a setup phase depending on the key. Additionally, depending on whether the key bit is 1 or 0, a different number of instructions is executed for each key bit. In order to recover the full private key, we first need to determine the number of zero bits to find the instruction leaking the first key bit correctly. Second, we need to calculate the offset of the next key bit instruction to zero-step based on previously reconstructed key bits.

Determining the number of zero bits: The mbed TLS implementation skips the leading zeroes of the exponent. Therefore, the offset of the first instruction executed after the key-bit branch depends on the number of leading zeroes. In order to overcome this challenge, we note that the maximum number of leading zeroes relies on the size of the `mbedtls_mpi_uint` data type, which is either 32 or 64 bits. Hence, we assume a possible maximum of 63 leading zeroes. For each possibility, we calculate the offset of the targeted AVX instruction (a 1-bit) under the assumption that we have n leading zero bits. We target each calculated instruction offset and record the core voltage when zero-stepping this instruction. For each measurement, we reset the current energy consumption to a known state by executing multiple `hlt` instructions. Then, we measure each instruction 3 times and use the median of the measured values as a classifier and illustrate the observed measurements in Figure 8. The distinct peak, and, thus, the first set key bit gives away the number of leading zeroes.

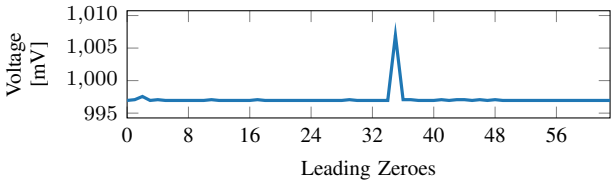

Fig. 8: Measured core voltage of all 63 possible leading zero offsets. The spike at offset 35 marks the first set key bit.

Offset Oracle: In order to find the instruction to zero-step for the next key bit, we create an oracle that predicts the offset of the next key bit instruction based on previously reconstructed key bits. The oracle receives the known-plaintext input, the public modulus, the number of leading zeroes as well as the current key bit. Using this information, the oracle calculates the next instruction offsets that need to be zero-stepped in order to recover the next key bit.

In our attack, we implemented the oracle utilizing the same enclave implementation for demonstration purposes. We inject the current key hypothesis into the enclave to automatically find the next instruction offset using single stepping. While this increases the runtime of the attack, it allows to predict the next offset without having to analyze the enclave on an instruction-level basis.

Evaluation: We evaluated our attack on a Xeon E3-1275 v5 server CPU. In order to profile the instruction at the calculated offset, we measure the observed core voltage 3 times over 256 zero steps. For a 1-bit, the instruction at and after the given offset are AVX instructions and, thus, we measure both to increase the signal. We used an RSA key pair with a 512-bit modulus for evaluation purposes. In 211 minutes (n = 5,$\sigma_{\bar{x}}$ = 7.2), we were able to reconstruct the 509 key bits without any error. Figure 13 in Appendix B illustrates one of the recorded traces. Note that the slow implementation of the oracle compensates for 52 minutes (n = 4, $\sigma_{\bar{x}}$ = 6.73) of the attack. In addition, we successfully recovered the key without any error in 100 minutes, even when we measured each instruction only once.

B. Correlation Power Analysis Attacks

The SPA attack in Section V-A exploits the comparatively strong change in leakage in the energy consumption or core voltage due to the different instructions executed. In contrast, in this section, we focus on differential attacks (see Section II-A) that apply to implementations with secret-independent control flow, e.g., symmetric ciphers like AES, targeting the data-dependent leakage of single instructions. We show that Correlation Power Analysis can be applied to exploit the small, data-dependent leakage of single instructions even

when capturing one aggregate leakage sample for the whole cryptographic algorithm.

To this end, we demonstrate key recovery attacks against AES-NI, an x86 instruction-set extension designed to mitigate timing and cache side-channel leakage [32] in two different settings. First, we will recover the AES key processed inside an SGX enclave and second, from a Linux kernel module.

In contrast to the RSA signature generation from Section V-A, a single run of our target algorithms has a very short runtime (on the order of tens to hundreds of cycles). Hence, the overall energy consumption is below the resolution of the RAPL interface (a single invocation usually reads as a zero energy consumption difference). We therefore generally measure the aggregate energy consumption of R invocations of our target cipher (typically $16\,\mathrm{M}$) to obtain a single leakage sample p. Our attacks, therefore, apply to situations where an adversary can trigger the encryption/decryption of many blocks of data, e.g., disk and file encryption, encrypted network protocols like TLS, or (un)sealing of large enclave state. In the case of a privileged attacker, the attacker model allows the alternative approach of using zero-stepping to only repeat the target instruction in the scenario of Intel SGX. Moreover, differential attacks like CPA make use of many leakage samples p_n (traces) for different inputs (plaintexts) x_n for $n < N$. Depending on the scenario, we used N between $2\,\mathrm{M}$ and $16\,\mathrm{M}$.

1) Key extraction with CPA: To recover a secret value, we compute the correlation $\rho(p,\, h)$ between the observed power consumptions p_n and hypothetical leakage values h_n over all N traces. The choice of h depends on the targeted operation and the leakage characteristics of the target implementation and processor. For example, for recovering byte 0 of the round key in the final round of AES, a common choice (given a key candidate k) is:

$$h_n^k = \mathrm{HW}\left(\mathrm{SBox}^{-1}\left(c_n^0 \oplus k\right)\right) \tag{2}$$

where c_n^0 is byte 0 of the n'th ciphertext, and HW denotes the Hamming weight. Computing $\rho^k\left(p,\, h^k\right)$ for all candidates $k = 0 \ldots 255$, the correct key candidate can be identified as the one with maximum correlation. This process is repeated for each byte. Other choices of h are possible, e.g., when targeting the XOR in the first round of AES:

$$h_n^k = \mathrm{HW}\left(x_n^0 \oplus k\right) \tag{3}$$

For a given number of traces N, the *noise level* is [55]:

$$\rho_{noise} = \frac{4}{\sqrt{N}} \tag{4}$$

Only correlations $\rho \geq \rho_{noise}$ are considered significant. Assuming an ideal correlation ρ_{exp} that captures only the correlation between the target value and a noise-free trace and a Signal-to-Noise Ratio (SNR), the observed correlation ρ can be computed as:

$$\rho = \frac{\rho_{exp}}{\sqrt{1 + {}^1\!/_{SNR}}} \tag{5}$$

TABLE IV: Profiling correlations after $2\,\mathrm{M}$ traces for AES-NI in scenario 1 for the Hamming weight (HW) for each round and Hamming distance (HD) between rounds. Bold entries and a $|\mathrm{SF}| \geq 1$ highlight significant statistical dependencies.

HD	ρ	SF	HW	ρ	SF
00 → 01	0.03675729	**13**	00	0.06885782	**24**
01 → 02	0.02006421	**7.1**	01	0.05032280	**18**
02 → 03	0.03676030	**13**	02	0.00145256	0.51
03 → 04	0.03728021	**13**	03	0.00181104	0.64
04 → 05	0.03754657	**13**	04	0.00188247	0.66
05 → 06	0.03739362	**13**	05	0.00186131	0.66
06 → 07	0.03804800	**13**	06	0.00204561	0.72
07 → 08	0.03790153	**13**	07	0.00151157	0.53
08 → 09	0.03810117	**13**	08	0.00250208	0.88
09 → 10	0.03967649	**14**	09	0.00272294	0.96
10 → 11	0.01820413	**6.4**	10	-0.00045022	-0.16
			11	0.08859152	**31**

2) SGX Enclave: In the first setting, we will demonstrate AES-NI key recovery on an SGX enclave.

Setup: We implement an enclave that exposes an `ecall` to encrypt a buffer using an in-enclave secret key. It deploys a full AES implementation from Intel's Integrated Performance Primitives (Intel IPP) [42] `ippsAESEncryptECB` function that uses the AES-NI instruction set. While the SGX scenario enables a privileged attacker (Section IV-A), we assume an unprivileged attacker.

We further considered two scenarios:

1) Minimal I/O noise: The unprivileged attacker records the accumulated power consumption of $16\,384$ calls to `ippsAESEncryptECB`, each encrypting $16\,\mathrm{kB}$, within a single `ecall`.
2) Real-world conditions: The unprivileged attacker records the accumulated power consumption of 64 `ecall` invocations, each encrypting $4\,\mathrm{MB}$ with a single call to `ippsAESEncryptECB`.

Profiling: To better understand the leakage behavior of the AES-NI implementation on the processor under attack, we compute the AES state after every round. Further, we compute the correlation between different power models and our observed traces.

We recorded $2\,\mathrm{M}$ traces (thus $\rho_{noise} = 0.0028284$) for scenario 1 in $26\,\mathrm{h}$ and $16\,\mathrm{M}$ traces (thus $\rho_{noise} = 0.001$) for scenario 2 in $277\,\mathrm{h}$. Table IV shows the correlations of the Hamming weight for each round and the Hamming distance between rounds on our Xeon E3-1240 for scenario 1.

As discussed in Section V-B1, bold entries highlight significant entries with an exploitable statistical dependency ($\rho \geq \rho_{noise}$). In addition, the Significance Factor (SF) is computed as ${}^{\rho}\!/_{\rho_{noise}}$, *i.e.*, $|\mathrm{SF}| \geq 1$ indicates a significant correlation. For instance, the Hamming weight of the input and output leak, as well as the Hamming weight of the 128-bit state after the initial XOR of round key 0 to the plaintext (correlation $\rho = 0.05032280$). In addition, the Hamming distance between the input and output of each AES round leaks, which is crucial for subsequent key recovery attacks.

For scenario 2, we similarly observed Hamming weight and Hamming distance leakages for the AES rounds, albeit with a lower magnitude of the correlations. For example, for the final round, the correlation is $\rho = 0.00532594$ in scenario 2, compared to $\rho = 0.01820413$ in scenario 1. Therefore, for key recovery in scenario 2, a larger number of traces is required. The respective profiling results are given in Table VI in Appendix C.

Key Recovery: To recover the key, we build a CPA attack using the Hamming distance between the input and the output of the final round of AES. As observed in the profiling phase, the correlation of the Hamming distance of the final round $10 \rightarrow 11$ yields $\rho = 0.01820413$ in scenario 1. In this case, we successfully recovered all 16 bytes of the final round key using $2\,\mathrm{M}$ traces, and hence also the actual AES key due to the reversible key schedule of AES.

In scenario 2, the respective correlation for the Hamming distance of the final round $10 \rightarrow 11$ is $\rho = 0.00532594$. We performed a CPA key recovery using $16\,\mathrm{M}$ traces and successfully recovered 12 of the 16 bytes of the full key. The remaining four bytes of the key can then be found in negligible time through exhaustive search with 2^{32} AES invocations. Incidentally, we note that the key recovery specifically fails for key bytes 0, 4, 8, and 12, *i.e.*, the first byte of each 4-byte word. This implies that these bytes might exhibit a different leakage behavior than the other (successfully recovered) bytes. Hence, with an appropriate leakage model, it might be possible to also directly recover those four bytes without exhaustive search. We leave this aspect for future work.

3) Kernel Module: Likewise, to our attack on the SGX enclave (Section V-B2), we evaluate the CPA attack on a Linux kernel module, processing an AES-NI key.

Setup: We implemented a kernel module encrypting data using AES-NI accelerated encryption. Therefore, we made use of the Intel AES-NI Sample Library [23] that claims to be some of the most efficient AES assembler code implementations [39]. The kernel module provides an `ioctl` interface to user space where data to be encrypted can be passed to.

Profiling: For the attack on the kernel module, we recorded $4\,\mathrm{M}$ traces ($\rho_{noise} = 0.0002$) in $50\,\mathrm{h}$ on the Xeon E3-1240 v5 (server) system. Each leakage sample corresponds to $16\,384$ encryptions of $16\,\mathrm{kB}$ in a loop inside the `ioctl` handler, similar to scenario 1 for SGX above. As for SGX, we observe statistical significant leakage for the AES rounds using both the Hamming weight and Hamming distance models. The profiling results are given in Table VIII in Appendix C.

Key Recovery: For the attack on the kernel module, we performed a CPA key recovery using the $4\,\mathrm{M}$ traces, again targeting the final round of AES. We successfully recovered 15 of the 16 bytes of the full key. Note that the correct candidate for the remaining byte was the second-best candidate.

4) Limitations: We showed that it is possible to recover secrets from AES-NI, both from implementations in the kernel and from an Intel IPP function using AES-NI in Intel SGX. These attacks are feasible, and the number of traces is also well in the threat model. For example, previous side-channel attacks on AES-NI with physical access required recording a large number of traces for 17 days using an EM probe [70]—longer than the time required for our method. Furthermore, as input to the NISTIR 8214A draft [8], Rijmen and Svetla [69] recommend considering an adversary that can collect up to $100\,\mathrm{M}$ traces.

While we note that our attacks might succeed with fewer traces using algorithms designed to perform a CPA-guided exhaustive search [83], we did not evaluate that in our attacks. Still, whether our CPA attacks are practical depends on the target, as we require large amounts of data to be processed with a fixed or known plaintext. In the case of Intel SGX, as a privileged attacker, it might be possible to alleviate this issue using zero-stepping (see Section IV-B2). Instead of repeating the whole algorithm, it is possible to repeat only the target instruction (which should also result in a better SNR). However, in our experiments so far, we could not successfully apply CPA in this case. This might be due to the noise introduced by the zero-stepping logic, combined with long measurement times, which prevent the acquisition of a sufficient number of traces. Finally, determining the appropriate leakage model depends on the specific implementation of the algorithm under attack and also the targeted CPU—e.g., we observed substantial differences for AES-NI between our i3-7100U and Xeon E3-1240 v5 systems, with the i3-7100U exposing less leakage (see Appendix C). We leave a more in-depth study of the behavior for future work.

C. Observing Intra-Cacheline Activity

A common assumption for side-channel-secure software was that an attacker can only observe victim operations at a cache line granularity [10]. For instance, to protect against cache attacks that observe access patterns at a cache line granularity, such as Flush+Reload and Prime+Probe, *scatter-gather* [25] is a constant-time programming technique for RSA. However, recent work [88], [60] showed that this assumption does not hold when an attacker shares a hyperthread with a victim. Consequently, the attacker can infer the cryptographic key used by an implementation that has sub-cache-line variations in the control flow or data accesses.

However, for our attack, we assume a scenario where the victim and attacker do not share a hyperthread. Consequently, previous attacks [88], [60] cannot obtain this information.

In our experiment, the victim performs a secret-dependent branch within a cache line, executing instructions with different power consumption. If the bit at a given offset of a secret byte is set, a `fscale` instruction is executed. Otherwise, `rdrand` is executed. We assume an unprivileged attacker that can trigger the code executed by the victim through an API passing the offset to it. We evaluated the experiment on our i7-8650U and i7-6600U (mobile) system, both running on battery and connected to an AC power supply, both desktop machines (i7-6700K and i9-9000K) as well as on the 3 servers (E3-1240 v5, E3-1275 v5, Silver 4214). The attacker records the power consumption when triggering the victim. As illustrated in Figure 9, one can clearly distinguish jump-target

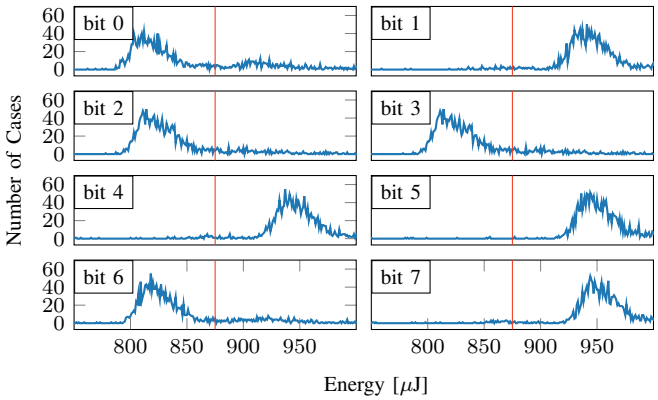

Fig. 9: Our attack clearly distinguishes different jumps within the same cache line. In this figure, leaking the byte `0x4d` (ASCII 'M') (`01001101` in binary) bit by bit by inspecting the power consumption. Values below that threshold are interpreted as '1's, values above as '0's.

locations within a cache line due to the difference in power consumption. Hence, constraining control-flow variations in cryptographic operations to a cache line cannot be considered secure anymore, even in scenarios where victim and attacker do not share a hyperthread. This allows breaking cryptographic implementations, which are currently considered secure in the scenario we investigate [32].

In addition, an extreme approach suggested to impede cache timing attacks is to disable caching for the `PRM` range in SGX [17]. In a second experiment, we mark pages of our victim as uncacheable. Thus, the code cannot leak through cache timings anymore. Still, with our power side-channel, we can observe the leakage.

D. Kernel Address Space Derandomization

In this section, we show that an unprivileged attacker can derandomize the kernel address space using RAPL. As there is no distinction between committed and non-committed instructions at the voltage regulator level, the power consumption also changes for transient instructions. Transient instructions are instructions that have been executed by an out-of-order processor but are never committed to the architectural state, e.g., instructions causing a fault [53] or instructions following a misspeculated branch [48]. The general concept of derandomizing the kernel address space is to distinguish between the transient access of mapped and unmapped kernel addresses via differences in power consumption. The current KASLR implementation randomly chooses one out of 512 2 MB-aligned virtual addresses as the base address for the entire kernel [71]. Hence, as the kernel binary itself does not support fine-grained randomization, knowing the base offset of the kernel allows to calculate the location of kernel code and data [37], [46], [30], [71], [12]. The same approach can also be applied to dynamically loadable kernel modules [71].

Transiently accessing mapped and unmapped kernel addresses show differences in timing [37], [46], [30] and store-forwarding behavior [12], [71]. Hence, the assumption is that there is also a measurable difference in power consumption.

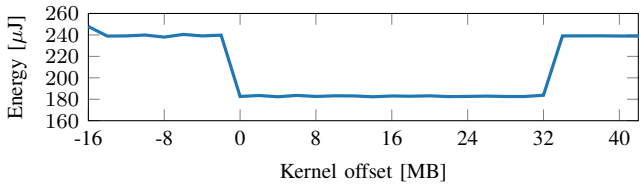

Fig. 10: Power consumption when transiently accessing kernel addresses. If a kernel page is not mapped, the access triggers an entire page-table walk, consuming more power.

Figure 10 shows the power consumption when transiently loading a kernel address while suppressing faults using Intel TSX. The power consumption differs for mapped and unmapped kernel pages. The differences in power consumption correlate with the differences in access times reported by Jang et al. [46]. As unmapped kernel pages cannot be cached in the TLB, accessing these pages triggers a page-table walk which consumes more power than accessing mapped kernel pages, which are cached in the TLB.

In our experiments, we used our i7-8650U (connected to a power supply and running on battery), i7-6600U, i9-9900K, and Xeon Silver 4214 systems with PTI (Page Table Isolation) disabled. Note that both, the i9-9900K and the Silver 4214, contain hardware mitigations against Meltdown; thus, PTI can be disabled. To evaluate the success rate, we execute the KASLR break 500 times for known KASLR offsets. On average, we successfully derandomize the KASLR offset in 100 % (n = 500, $\sigma_{\bar{x}}$ = 0.00) of the runs. The average time to find the KASLR offset is 20 s. Hence, while not being the fastest KASLR break, it is still practical. Moreover, in contrast to previous microarchitectural KASLR breaks [46], [37], [30], [71], [12], [13], our KASLR break using power consumption is the first microarchitectural KASLR break, which does not require any timing primitive. Even with the microcode patch on the i9-9900K, there is no significant change in the success rate of the KASLR break. This is in line with Intel's statement that attacks on KASLR are not mitigated by this update [41].

In addition, we evaluated the influence of system activity using `stress-ng` on the success rate of the KASLR break on the i9-9900K running Ubuntu 18.04. These tests are designed to stress the CPU and do not represent a realistic workload, e.g., compilation task, rendering process, or office workload. However, the tool allows us to vary the load on each core. By default, it will cycle through all stress tests unless a specific one is specified. With a load below 10 % on the entire system, there is no change in the success rate. With a moderately high load of 50 %, it decreases to 22 % (n = 100, $\sigma_{\bar{x}}$ = 4.34). However, as system noise is statistically independent from the measured signal, increasing the number of measurements (and thus the runtime) increases the success rate. Especially as system activity only *increases* the power consumption, and mapped pages have a *lower* power consumption than

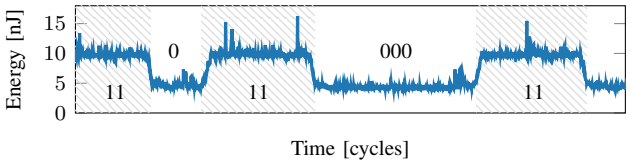

Fig. 11: Transmission of bits `1101100011` using the timeless covert channel.

unmapped pages, noise does not lead to false positives, but only to not being able to detect the kernel (false negative). A simply increase of the measurements by a factor of 10 already results in a success rate of $46\,\%$ (n = 100, $\sigma_{\bar{x}}$ = 4.75).

E. Timing-Independent Covert Channel

In this section, we describe how unprivileged access to power consumption can be utilized to establish a timing-independent covert channel. The basic idea of the covert channel is to encode the information by varying the power consumption of the device. To send a 1-bit, the sender increases the power consumption by executing more energy-consuming instructions. To transmit a 0-bit, the sender idles. The receiver monitors the power consumption of the device through the RAPL interface and decodes the transmitted information by observing the changes in power consumption.

Figure 11 illustrates the transmission of the bits `1101100011` over the power-based covert channel. We transmitted $1\,\mathrm{kB}$ of random data between two unprivileged processes running on different cores of the i7-8650U, either battery-powered or connected to a power supply. We achieved a transmission rate of $18.7\,\mathrm{bit/s}$ with a bit error rate of $0.89\,\%$.

While the transmission rate of our covert channel is significantly lower in contrast to other state-of-the-art covert channels [54], [29], [58], our covert channel has the benefit that it does not rely on high-resolution timers. Furthermore, our proof-of-concept covert channel is not optimized and strictly working only with binary decisions. However, we can transmit not just one bit per symbol but rather several bits by using modulation techniques, such as amplitude modulation, phase-shift keying, or frequency modulation. While Maurice et al. [58] found that these methods are infeasible for cache covert channels due to the unreliable clock, they are applicable to a power-based covert channel. Thus, we believe that the performance of our covert channel could be drastically improved using these techniques.

VI. Countermeasures

In this section, we discuss different countermeasures and mitigation strategies for the presented attacks.

Restricting Access: To obtain the Intel RAPL counters, kernel privileges are required to read the corresponding MSRs. However, the power capping framework `powercap` on Linux provides unprivileged access to these MSRs through the `sysfs` interface. While the purpose of the driver is to expose RAPL for user-space consumption [65], unprivileged access could be directly prevented by respecting the access level similar to `kernel.perf_event_paranoia` for the `perf` interface. While these interfaces may be required for existing functionality, limiting user-space access is necessary to mitigate at least unprivileged attacks. However, as a privileged attacker has direct access to these MSRs, attacks on Intel SGX are not prevented. Thus, access to these MSRs needs to be blocked via a microcode update. Furthermore, trusted computing base recovery is required to allow remote verifiers to re-establish the trust that these MSRs have been deactivated.

Limiting Resolution: The RAPL interface has a µJ resolution. While reducing the counter's granularity does not completely mitigate our attacks, the number of traces for some scenarios might become impractical. However, even without the RAPL interface, it may still be possible to use other limited-resolution sources of energy data, e.g., battery monitoring, to conduct a software-based power side-channel attack, e.g., identifying running applications [87].

Limiting Precise Execution Control: Restricting the user-space access to the RAPL counters only impedes unprivileged attackers, as a privileged attacker has direct access to these MSRs. In addition, the attacker can make use of precise execution control (cf. Section IV-B2) to zero step an enclave. This primitive gives an attacker the possibility to execute a single instruction within an SGX enclave arbitrarily often, enabling sampling of the instruction's energy consumption (cf. Section V-A). Introducing a counter inside SGX that increments every time an enclave is executed from the same instruction pointer could limit the number of zero steps.

Application Hardening: Software computing on particularly sensitive values, e.g., cryptographic algorithms, could deploy state-of-the-art countermeasures against power analysis, e.g., masking, to make these attacks more difficult. However, using zero stepping (Section IV-B2) and the possibility to observe the Hamming weight of bytes (Section III-E), masking is insufficient against our attacks on SGX enclaves.

Intel's Mitigation: To address the presented issues, Intel plans to release microcode updates that help ensure that the reported energy consumption by the RAPL interface hinders the ability to distinguish same instructions with different data or operands if SGX is enabled. In addition, an update to the Linux `powercap` driver restricts the unprivileged access to the RAPL MSRs.

VII. Related Work & Discussion

In this section, we present related and future work and discuss other microarchitectures.

A. Related Work

Hardware-based Power Analysis: Eisenbarth et al. [19] reconstructed control-flow and program code from power consumption on a small microcontroller. Strobel et al. [76] distinguish instructions on a microcontroller using an oscilloscope sampling at $2.5\,\mathrm{GHz}$. Park et al. [66] use an oscilloscope with $2.5\,\mathrm{GHz}$ in combination with machine learning to extract the instruction stream (opcodes and operands) from a microcontroller. Msgna et al. [62] measured differences in power

consumption during the execution of single instructions on a microcontroller using an oscilloscope with a sampling rate of 5 GHz. Saab et al. [70] extracted an AES-NI key from an Intel i7 after collecting traces for 17 days with an EM probe.

Guri et al. [33], as well as Islam and Ren [45] demonstrated that current and voltage respectively, can be monitored and influenced to build covert channels, e.g., in cloud environments. However, both works assume an attacker with hardware equipment connected to the device.

Undersampling: Molka et al. [61] used a physically-connected power meter to record a victim system's power consumption at a rate of 10 Hz, distinguishing loops of `nop`s, and other instructions. Attacks with similar sampling rates to ours were shown by Genkin et al. [22], who recovered 4096-bit GnuPG RSA keys and program code via acoustic cryptanalysis, and Lifshits et al. [52], who inferred sensitive data, including keystrokes, via a malicious battery storing power traces. These works sampled at ≈24 kHz (mobile phone attack) and 1 kHz, respectively.

Our work shows that this can similarly be done from software at even higher sampling rates, and our attacks demonstrate the security ramifications of this. While prior attacks require either physical proximity or physical access to the device, they support this work's finding that a low sampling rate can still achieve fine-grained information leakage.

Software-based Power Analysis: Fusi [20] used RAPL to attack RSA-16384 but concluded that the sampling rate of RAPL is too low to mount an attack, showing that it is only observable whether branches are taken and accessed data is cached. Mantel et al. [56] distinguish RSA keys with different Hamming weights using RAPL but do not try to extract keys or perform other concrete attacks. Gao et al. [21] use RAPL in containers to infer information about the host environment, e.g., co-location of multiple containers.

Power Analysis on Mobile Devices: Yan et al. [87] monitor system power information on mobile devices to acquire voltage and current, observing a correlation with keystrokes, enabling them to infer password lengths and also distinguish different applications. Qin et al. [68] use the same interfaces to fingerprint websites on mobile devices. We instead use RAPL on regular laptops, desktops, and servers that have more subtle variation in power consumption and voltage.

On-die Power Analysis: O'Flynn [64] recorded power measurements using an on-board ADC from the non-secure world to recover secrets processed in the secure world on TrustZone-M. Zhao and Suh [89] use an FPGA to observe a CPU's power consumption on the same SoC to break RSA.

B. Other Microarchitectures

While we focus on Intel's RAPL implementation throughout this work, other microarchitectures offer different interfaces to obtain the energy consumption of the core.

For instance, since the Zen microarchitecture, AMD CPUs also provide a RAPL interface [2]. In contrast to Intel, their counters even allow to measure the energy consumption even per individual core. However, as the `powercap` driver does not support AMD's implementation, an attacker requires kernel privileges to read the corresponding MSRs. In Appendix A, we show that AMD's RAPL interface allows to distinguish different instructions executed on an AMD Ryzen CPU. This could allow similar attacks on AMD CPUs, e.g., against AMD's SEV-SNP, where a privileged kernel-space attacker is conceivable.

Other CPU manufacturers, e.g., ARM, NVIDIA, IBM POWER, Ampere, Hygon, or Marvell, provide different power interfaces as well. We briefly discuss them in Appendix A and leave the investigation of them to future work.

C. Enclave Inspection

While Intel SGX provides integrity and confidentiality of data and integrity of code at runtime, it does not provide confidentiality of code in the binary file stored offline. However, with the Intel Software Guard Extensions Protected Code Loader (Intel SGX PCL) [44], the enclave shared object is encrypted at build time and decrypted during the load phase. This enables intellectual property within SGX enclave code to be protected from inspection by untrusted parties, as reverse-engineering of the encrypted enclave is not possible [6]. Furthermore, encrypting the memory used by the enclave [17] prevents runtime inspection, provided the enclave is built in release mode [43].

Using zero-stepping, we can now measure the energy consumption of every single instruction executed within an SGX enclave. This allows to classify different instructions by evaluating their power consumption, as shown in Section III. Further, differences depending on the values of their operands and loaded data from the cache can be observed. This enables us to not only recover the control flow of the executed program but also to directly disclose sensitive information, as we demonstrate in Section V-A.

For enclave inspection, the idea is to retrofit the power-side-channel-based disassembler by Eisenbarth et al. [19] with PLATYPUS to infer the control flow of the enclave. While our results are promising for a certain set of instructions (see Table III), the general case is very complex due to the complex instruction-set architecture. In total, there are more than 3684 x86-64 instruction variants (combining mnemonics and operand types) [35] that need to be profiled on the microarchitecture under attack first. Thus, the set of instructions with similar power consumption, especially with the influence of different operand values, is currently too large. We leave further exploration to future work.

VIII. Conclusion

In this work, we show that software-based power side-channel attacks are particularly powerful against Intel SGX due to the zero-stepping capabilities of a privileged attacker. We showed how instructions and operand-level differences can be observed, enabling recovery of an RSA key from mbed TLS inside an SGX enclave. We demonstrated that with sufficient statistical evaluation, even user space attackers can exploit unprivileged access to the Intel RAPL interface to extract

AES-NI keys from SGX enclaves or kernel space. Moreover, we demonstrated that this side channel enables an attacker to break KASLR, observe sub-cache-line-granularity activity, and establish timing-independent covert channels.

While unprivileged attacks can be impeded by restricting access to the `sysfs` interface, mitigating privileged attacks in order to protect Intel SGX enclaves is not trivial. We, therefore, propose limiting precise execution control and, while it, unfortunately, breaks backward compatibility and support for software-based thermal management, removing access to these interfaces in general.

Acknowledgments

We want to thank Peter Pessl (Infineon Technologies) and Stefan Mangard (Graz University of Technology).

The research presented in this paper was supported by the Austrian Research Promotion Agency (FFG) via the K-project DeSSnet, which is funded in the context of COMET - Competence Centers for Excellent Technologies by BMVIT, BMWFW, Styria, and Carinthia. It was also supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402). It has also been supported by the Austrian Research Promotion Agency (FFG) via the project ESPRESSO, which is funded by the province of Styria and the Business Promotion Agencies of Styria and Carinthia. It is partially funded by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/R012598/1, EP/S030867/1 and by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

Additional funding was provided by generous gifts from Intel, ARM, Amazon and Red Hat. Further, we would like to thank Equinix Metal for providing us access to bare metal instances to run our experiments.

Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] *Open-Source Register Reference For AMD Family 17h Processors Models 00h-2Fh*, 3rd ed., Advanced Micro Devices Inc., 7 2018.

[2] *AMD uProf User Guide*, 3rd ed., Advanced Micro Devices Inc., 2019.

[3] N. S. Agency, "TEMPEST: A Signal Problem," 1972.

[4] T. Allan, B. B. Brumley, K. Falkner, J. Van de Pol, and Y. Yarom, "Amplifying Side Channels Through Performance Degradation," in *ACSAC*, 2016.

[5] ARM, "mbed TLS," 2020. [Online]. Available: https:///tls.mbed.org

[6] J.-P. Aumasson and L. Merino, "SGX Secure Enclaves in Practice: Security and Crypto Review," in *Black Hat Briefings*, 2016.

[7] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in *HPCA*, 2013.

[8] L. T. Brandão, M. Davidson, and A. Vassilev, "Towards NIST Standards for Threshold Schemes for Cryptographic Primitives: A Preliminary Roadmap," National Institute of Standards and Technology, Tech. Rep., 2019.

[9] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software Grand Exposure: SGX Cache Attacks Are Practical," in *WOOT*, 2017.

[10] E. Brickell, "Technologies to Improve Platform Security," CHES, 2011.

[11] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES*, 2004.

[12] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout: Leaking Data on Meltdown-resistant CPUs," in *CCS*, 2019.

[13] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "KASLR: Break It, Fix It, Repeat," in *AsiaCCS*, 2020.

[14] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, 1995.

[15] A. Computing, "Ampere AltraTM Linux Kernel Porting Guide," 2020. [Online]. Available: https://github.com/AmpereComputing/ampere-centos-kernel/wiki/Ampere-AltraTM-Linux-Kernel-Porting-Guide

[16] I. Corporation, "What exactly is a P-state? (Pt. 1)," 2015.

[17] V. Costan and S. Devadas, "Intel SGX Explained," *Cryptology ePrint Archive, Report 2016/086*, 2016.

[18] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX," in *USENIX Security Symposium*, 2017.

[19] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a Side Channel Based Disassembler," in *Transactions on computational science X*. Springer, 2010.

[20] M. Fusi, "Information-Leakage Analysis Based on Hardware Performance Counters," 2017.

[21] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds," in *DSN*, 2017.

[22] D. Genkin, A. Shamir, and E. Tromer, "Acoustic Cryptanalysis," *Journal of Cryptology*, 2017.

[23] Gladman, Brian, "Intel AESNI Sample Library," 2013. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/download-intel-aesni-sample-library.html

[24] J. D. Golić and C. Tymen, "Multiplicative Masking and Power Analysis of AES," in *CHES*, 2002.

[25] V. Gopal, J. Guilford, E. Ozturk, W. Feghali, G. Wolrich, and M. Dixon, "Fast and Constant-Time Implementation of Modular Exponentiation," *Embedded Systems and Communications Security*, 2009.

[26] L. Goubin and J. Patarin, "DES and Differential Power Analysis: The "Duplication" Method," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 1999.

[27] C. Gough, I. Steiner, and W. Saunders, *Energy Efficient Servers*. Apress, 2015.

[28] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory," in *USENIX Security Symposium*, 2017.

[29] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack," in *DIMVA*, 2016.

[30] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR," in *CCS*, 2016.

[31] A. Guermouche and A.-C. Orgerie, "Experimental analysis of vectorized instructions impact on energy and power consumption under thermal design power constraints," 2019.

[32] S. Gueron, "Intel Advanced Encryption Standard (Intel AES) Instructions Set – Rev 3.01," 2012.

[33] M. Guri, B. Zadov, D. Bykhovsky, and Y. Elovici, "PowerHammer: Exfiltrating Data from Air-Gapped Computers Through Power Lines," *arXiv:1804.04014*, 2018.

[34] A. Herrmann, "Kernel driver fam15h_power: The Linux Kernel documentation," 2019. [Online]. Available: https://www.kernel.org/doc/html/v5.4-preprc-cpu/hwmon/fam15h_power.html

[35] S. Heule, E. Schkufza, R. Sharma, and A. Aiken, "Stratified Synthesis: Automatically Learning the x86-64 Instruction Set," in *PLDI*. ACM, 2016.

[36] M. Hirki, Z. Ou, K. N. Khan, J. K. Nurminen, and T. Niemi, "Empirical study of the power consumption of the x86-64 instruction decoder," in *USENIX CoolDC*, 2016.

[37] R. Hund, C. Willems, and T. Holz, "Practical Timing Side Channel Attacks against Kernel Space ASLR," in *S&P*, 2013.

[38] IBM, *POWER9 Processor User's Manual*, 2nd ed., 2018.

[39] Intel, "Advanced Encryption Standard (AES) Crypto Performance Analysis Project," 2013.

[40] ——, "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide," 2019.

[41] Intel, "Intel-SA-00389," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00389.html
[42] ——, "Intel® Integrated Performance Primitives," 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/integrated-performance-primitives.html
[43] Intel Corporation, "Intel SGX: Debug, Production, Pre-release – What's the Difference?" January 2016.
[44] ——, "Intel Software Guard Extensions (Intel SGX) Protected Code Loader (PCL) for Linux," May 2018.
[45] M. A. Islam and S. Ren, "Ohm's Law in Data Centers: A Voltage Side Channel for Timing Power Attacks," in *CCS*, 2018.
[46] Y. Jang, S. Lee, and T. Kim, "Breaking Kernel Address Space Layout Randomization with Intel TSX," in *CCS*, 2016.
[47] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in Action: Experiences in Using RAPL for Power Measurements," *ToMPECS*, 2018.
[48] 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 *S&P*, 2019.
[49] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to Differential Power Analysis," *Journal of Cryptographic Engineering*, 2011.
[50] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO'99*, 1999.
[51] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in Darkness: Return-oriented Programming against Secure Enclaves," in *USENIX Security Symposium*, 2017.
[52] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference Attacks by Malicious Batteries on Mobile Devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, 2018.
[53] 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 *USENIX Security Symposium*, 2018.
[54] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," in *S&P*, 2015.
[55] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer Science & Business Media, 2008.
[56] H. Mantel, J. Schickel, A. Weber, and F. Weber, "How Secure is Green IT? The Case of Software-Based Energy Side Channels," in *European Symposium on Research in Computer Security*, 2018.
[57] Marvell, "tx2mon," 2020. [Online]. Available: https://github.com/Marvell-SPBU/tx2mon
[58] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. Alberto Boano, S. Mangard, and K. Römer, "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud," in *NDSS*, 2017.
[59] A. Mazouz, D. C. Wong, D. Kuck, and W. Jalby, "An Incremental Methodology for Energy measurement and Modeling," in *ACM ICPE*, 2017.
[60] A. Moghimi, T. Eisenbarth, and B. Sunar, "MemJam: A False Dependency Attack against Constant-Time Crypto Implementations in SGX," in *CT-RSA*, 2018.
[61] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, "Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86-64 Processors," in *International Conference on Green Computing*. IEEE, 2010.
[62] M. Msgna, K. Markantonakis, and K. Mayes, "Precise Instruction-Level Side Channel Profiling of Embedded Processors," in *International Conference on Information Security Practice and Experience*, 2014.
[63] NVIDIA, "Jetson TX2: Thermal Design Guide," 2017.
[64] C. O'Flynn and A. Dewar, "On-Device Power Analysis Across Hardware Security Domains," *CHES*, 2019.
[65] J. Pan, "RAPL (Running Average Power Limit) driver," 2013. [Online]. Available: https://lwn.net/Articles/545745/
[66] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based Side-Channel Instruction-level Disassembler," in *DAC*, 2018.
[67] J. Phung, Y. C. Lee, and A. Y. Zomaya, "Modeling System-Level Power Consumption Profiles Using RAPL," in *NCA*. IEEE, 2018.
[68] Y. Qin and C. Yue, "Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7," in *TrustCom/BigDataSE*, 2018.
[69] V. Rijmen and S. Nikova, "Threshold Cryptography Against Physical Attacks," 2020. [Online]. Available: https://www.esat.kuleuven.be/cosic/events/tis-online-workshop/wp-content/uploads/sites/6/2020/07/Vincent_Rijmen.pdf
[70] S. Saab, P. Rohatgi, and C. Hampel, "Side-Channel Protections for Cryptographic Instruction Set Extensions," *IACR Cryptology ePrint Archive*, 2016.
[71] M. Schwarz, C. Canella, L. Giner, and D. Gruss, "Store-to-Leak Forwarding: Leaking Data on Meltdown-resistant CPUs," *arXiv:1905.05725*, 2019.
[72] M. Schwarz, D. Gruss, M. Lipp, M. Clémentine, T. Schuster, A. Fogh, and S. Mangard, "Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features," *AsiaCCS*, 2018.
[73] M. Schwarz, D. Gruss, S. Weiser, C. Maurice, and S. Mangard, "Malware Guard Extension: Using SGX to Conceal Cache Attacks ," in *DIMVA*, 2017.
[74] Y. S. Shao and D. Brooks, "Energy Characterization and Instruction-Level Energy Model of Intel's Xeon Phi Processor," in *ISLPED*, 2013.
[75] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "MicroScope: Enabling Microarchitectural Replay Attacks," in *ISCA*, 2019.
[76] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A Side-ChANnel-based DisAssembLer using Local Electromagnetic Emanations," in *DATE*, 2015.
[77] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step towards Software Power Minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1994.
[78] Unified Extensible Firmware Interface (UEFI) Forum, "Advanced Configuration and Power Interface (ACPI) Specification, Version 6.3," 2019.
[79] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," in *USENIX Security Symposium*, 2018.
[80] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss, and F. Piessens, "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection," in *S&P*, 2020.
[81] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control," in *Workshop on System Software for Trusted Execution*, 2017.
[82] E. Vasilakis, "An Instruction Level Energy Characterization of ARM Processors," *FORTH-ICS/TR-450*, 2015.
[83] N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F.-X. Standaert, "An optimal key enumeration algorithm and its application to side-channel attacks," in *SAC*, 2012.
[84] R. Wartell, Y. Zhou, K. W. Hamlen, M. Kantarcioglu, and B. Thuraisingham, "Differentiating Code from Data in x86 Binaries," in *ECML PKDD*, 2011.
[85] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves," in *ESORICS*, 2016.
[86] P. Wen, "Add support for Hygon Fam 18h (Dhyana) RAPL," 2019. [Online]. Available: https://patchwork.kernel.org/patch/11123607/
[87] L. Yan, Y. Guo, X. Chen, and H. Mei, "A Study on Power Side Channels on Mobile Devices," in *Symposium on Internetware*, 2015.
[88] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A Timing Attack on OpenSSL Constant Time RSA," *JCEN*, 2017.
[89] M. Zhao and G. E. Suh, "FPGA-based Remote Power Side-Channel Attacks," in *S&P*, 2018.

APPENDIX

A. Other Microarchitectures

While we focus on Intel's RAPL implementation throughout this work, other microarchitectures offer different interfaces to obtain the energy consumption of the core.

AMD: Since the Zen (family 17H) microarchitecture, AMD CPUs have provided a RAPL interface [2]. However, little documentation is available regarding its implementation. Power consumption values for the core and package domains are provided respectively in the `CORE_ENERGY_STAT` and `PKG_ENERGY_STAT` MSRs [1]. A notable difference from Intel RAPL is that the core domain is accessible per-core rather

Instruction	Ryzen 7 Pro 3700U	Ryzen 7 3700X	EPYC 7401P
`nop`	0.0886 nJ	0.1052 nJ	0.1571 nJ
`inc r64`	0.1241 nJ	0.1144 nJ	0.1800 nJ
`xor r64, r64`	0.1246 nJ	0.1144 nJ	0.1785 nJ
`mov r64, mem`	0.0978 nJ	0.1266 nJ	0.1571 nJ
`imul r64, r64`	0.0930 nJ	0.0930 nJ	0.1586 nJ
`fscale`	0.0892 nJ	0.0991 nJ	0.1571 nJ
`rdrand r64`	0.0669 nJ	0.0564 nJ	0.0991 nJ
`rdtsc`	0.0885 nJ	0.0896 nJ	0.1296 nJ
`clflush mem`	0.0671 nJ	0.0503 nJ	0.0991 nJ
`aesenc xmm, xmm`	0.0890 nJ	0.0854 nJ	0.1571 nJ

TABLE V: Average observed energy consumption (package domain) of different instructions on an AMD Ryzen 7 Pro 3700U mobile CPU, an AMD Ryzen 7 3700X desktop CPU, and an AMD EPYC 7401P server CPU.

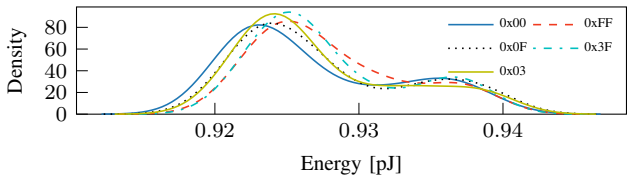

Fig. 12: Measured energy consumption of the `shr` instruction with different Hamming weights (AMD Ryzen 5 3600).

than across all cores of the CPU, which substantially reduces measurement noise.

An attacker targeting AMD currently requires root privileges to read the MSRs, as the Linux `powercap` driver only establishes a user-accessible `sysfs` RAPL interface on Intel CPUs. While they lack the RAPL interface, earlier AMD family 15h and 16h CPUs also have power MSRs which provide consumption estimates based on platform activity levels [2]. Unfortunately, none were available to us for evaluation purposes. However, we believe systems with these CPUs may be vulnerable to user-space attacks because if the `fam15h_power` driver is loaded, power values can be read from user space via `sysfs` [34].

Table V lists the measured energy consumption of different instructions on AMD Ryzen 7 Pro 3700U, AMD Ryzen 7 3700X, and AMD EPYC 7401P. On the mobile and desktop CPU, we disabled processor boost and set the cores to a fixed frequency. To measure the energy consumption of an instruction, we record its energy consumption over 10 000 consecutive executions and take the median value to eliminate system-level noise (e.g., erroneous high values caused by interrupt handling or the process being descheduled). On both AMD and Intel (see Section III), we observe the energy consumption across the entire CPU package to ensure that non-core activity (for example, interactions with DRAM) is included. We can observe inter-instruction differences in energy consumption. This enables identification of executed instructions, provided the attacker can profile the energy consumption of the victim microarchitecture. Furthermore, as shown in Figure 12, the Hamming weight of the register influences the energy consumption of the `shr` instruction.

ARM: The ARM Energy Probe, a 3-channel USB voltmeter which can be attached to a targeted platform, requires physical access to the device. However, different development boards using ARM CPUs contain on-board energy meters like the ARM CoreTile Express A15x2. The odroid XU+E used by Vasilakis [82] to characterize the energy consumption of instructions on ARM contains 4 `ina23` power sensors. The SAML11 running a Cortex-M23 processor used by O'Flynn and Dewar [64], grants access to an onboard ADC.

NVIDIA: NVIDIA's JetsonTX2 module has 3-channel INA3221 monitors [63] exposing current (mA), voltage (mV), and power (mW) used of different power rails. These include the CPU and GPU and are exposed to unprivileged access in the `sysfs`.

IBM POWER: The POWER9 processor contains a dedicated on-chip microcontroller that allows to analog sample various voltage rails. Note, however, that the POWER9 does not include per-core power estimation circuitry [38].

Marvell: For the ThunderX2, Marvell provides a kernel driver [57] exposing readings from hardware sensors, among other things, voltages and power measurements. Similar to Intel RAPL, measurements can be observed for all cores on the System on Chip, the SRAM, memory, and miscellaneous peripherals.

Ampere: For the Ampere Altra SoC, the APM X-Gene SoC hardware monitoring driver gives unprivileged access to the temperature and power sensors and, thus, allows to read the current power consumption of the CPU or the IO [15].

Hygon: Recently, RAPL support for the Hygon Dhyana CPU family has been added to the Linux `perf` interface and, likewise to AMD, allows to read the per-core energy consumption [86].

B. mbed TLS Attack

Figure 13 illustrates the minimum core voltage measurements for each key bit instruction of the mbed TLS attack described in Section V-A.

C. Additional Profiling Results for AES-NI

Table VI, Table VII and Table VIII present additional profiling correlations for AES-NI of the attacks described in Section V-B.

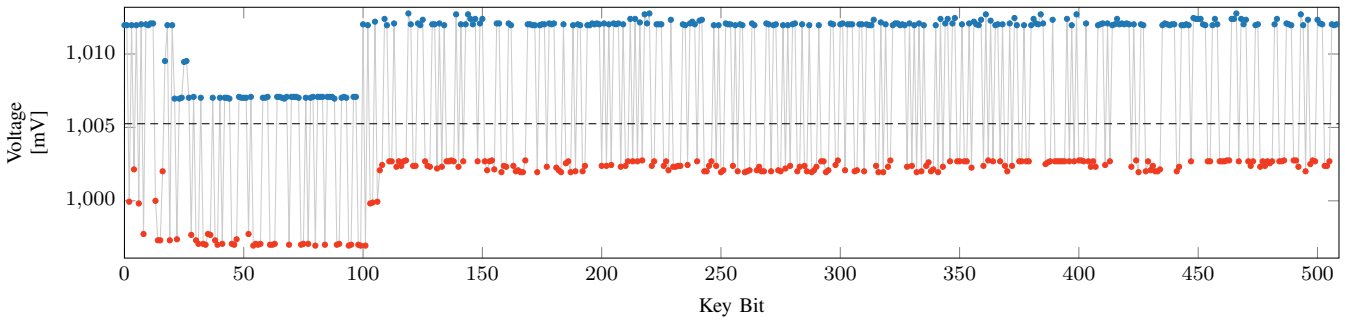

Fig. 13: Core voltage per measured instruction for each key bit offset in the fixed window length implementation of mbed TLS inside an SGX enclave on the Xeon E3-1275 v5. The blue marks represent 1 bits, while the red marks represent 0 bits. Using a threshold (dashed line), they can easily be distinguished.

TABLE VI: Profiling correlations (for Xeon E3-1240 v5) after 16 M traces for AES-NI in scenario 2 for the Hamming weight (HW) for each round and Hamming distance (HD) between rounds. Bold entries and a $|\mathrm{SF}| \geq 1$ highlight significant statistical dependencies.

HD	ρ	SF	HW	ρ	SF
00 → 01	0.01412518	**14**	00	0.06653038	**66**
01 → 02	0.00674140	**6.7**	01	0.01389394	**14**
02 → 03	0.01182713	**12**	02	0.00045177	0.45
03 → 04	0.01159959	**12**	03	0.00106697	**1.1**
04 → 05	0.01144089	**11**	04	0.00073025	0.73
05 → 06	0.01069259	**11**	05	0.00058525	0.58
06 → 07	0.01142695	**11**	06	0.00114676	**1.1**
07 → 08	0.01158716	**12**	07	0.00068475	0.68
08 → 09	0.01102899	**11**	08	0.00077455	0.77
09 → 10	0.01114280	**11**	09	0.00094852	0.95
10 → 11	0.00532594	**5.3**	10	-0.00041563	-0.41
			11	0.05861710	**58**

TABLE VII: Profiling correlations (for i3-7100U) after 4 M traces for AES-NI in scenario 1 for the Hamming weight (HW) for each round and Hamming distance (HD) between rounds. Bold entries and a $|\mathrm{SF}| \geq 1$ highlight significant statistical dependencies.

HD	ρ	SF	HW	ρ	SF
00 → 01	0.00429156	**2.1**	00	0.00957385	**4.8**
01 → 02	0.00256447	**1.3**	01	0.00550198	**2.7**
02 → 03	0.00441708	**2.2**	02	0.00056316	0.28
03 → 04	0.00404454	**2**	03	0.00003843	0.01
04 → 05	0.00388573	**1.9**	04	0.00048580	0.24
05 → 06	0.00512078	**2.6**	05	0.00081453	0.41
06 → 07	0.00418470	**2.1**	06	-0.00057528	-0.29
07 → 08	0.00454403	**2.3**	07	-0.00040692	-0.2
08 → 09	0.00477473	**2.4**	08	-0.00005976	-0.03
09 → 10	0.00488921	**2.4**	09	0.00085888	0.43
10 → 11	0.00269663	**1.3**	10	0.00021935	0.11
			11	0.01133641	**5.7**

TABLE VIII: Profiling correlations (for Xeon E3-1240 v5) after 4 M traces for AES-NI in the Linux kernel for the Hamming weight (HW) for each round and Hamming distance (HD) between rounds. Bold entries and a $|\mathrm{SF}| \geq 1$ highlight significant statistical dependencies.

HD	ρ	SF	HW	ρ	SF
00 → 01	0.063436878	**32**	00	0.092565061	**46**
01 → 02	0.029847718	**15**	01	0.075098846	**38**
02 → 03	0.056173544	**28**	02	0.0022803663	**1.1**
03 → 04	0.057817586	**29**	03	0.0033372879	**1.7**
04 → 05	0.057572691	**29**	04	0.0030430309	**1.5**
05 → 06	0.057020521	**28**	05	0.0034340331	**1.7**
06 → 07	0.058405015	**29**	06	0.0038034749	**1.9**
07 → 08	0.05697378	**28**	07	0.0022000058	**1.1**
08 → 09	0.057203062	**29**	08	0.0033568495	**1.7**
09 → 10	0.05837099	**29**	09	0.0031144225	**1.6**
10 → 11	0.027001464	**13**	10	-0.0008108201	-0.16
			11	0.12527739	**63**