"OOPS!": Out-Of-Band Remote Power Side-Channel Attacks on Intel SGX and TDX

Nimish Mishra
Indian Institute of Technology Kharagpur
nimish.mishra@kgpian.iitkgp.ac.in

Kislay Arya
Indian Institute of Technology Kharagpur
kislayarya536@kgpian.iitkgp.ac.in

Sarani Bhattacharya
Indian Institute of Technology Kharagpur
sarani@cse.iitkgp.ac.in

Paritosh Saxena
Intel Corporation, USA
paritosh.saxena@intel.com

Debdeep Mukhopadhyay
Indian Institute of Technology Kharagpur
debdeep@cse.iitkgp.ac.in

***Abstract*—Prior work shows that remote power attacks on Intel processors are possible through two Model Specific Registers (MSRs): `MSR_PKG_Energy_Status` and `MSR_PP0_Energy_Status`. In response, Intel introduced a defense: a bit in MSR `IA32_MISC_PACKAGE_CTLS` allows users to enable/disable "filtering" mechanism that adds additional noise to energy measurements to harden against power side-channel attacks.**

In this work, we demonstrate that "filtering" does not cover *all* possible avenues of measuring power. On Intel server-grade platforms, components like out-of-band management interface (OOB) exist which also expose telemetric information like in-band energy consumption. For this, we first reverse engineer the protocol structure over which OOB communicates with in-band components. We then show how OOB allows read-only access to the Package Configuration Space (PCS) and note that energy readings through PCS are outside the scope of filtering.

Using this, we establish remote power side-channels on Intel SGX and TDX operational on Intel Sapphire Rapids. We first construct a synchronization mechanism to align in-band execution with out-of-band measurements by leveraging deliberately disabled MSRs. We then use energy readings through OOB PCS to recover 2048-bit RSA keys from MbedTLS operational within in-band Intel SGX (with generic single-stepping assumption). Finally, we also leak AESNI keys from within in-band Intel TDX (without any single-step assumption).

Prior to our work, the literature on side-channels has been focused on attacks leveraging in-band interfaces. Our work establishes the importance of evaluating confidential computing architectures against attack vectors that *combine* abilities of both in-band and out-of-band interfaces to achieve adversarial objectives (that both in-band and out-of-band interfaces cannot independently achieve).

***Index Terms*—Power Side-Channel, Intel SGX, Intel TDX**

I. Introduction

Processor vendors like Intel ship Model Specific Registers (MSRs) which relay telemetric information (like energy consumption) to software for monitoring platform health. For instance, Running Average Power Limit (RAPL) MSRs `MSR_PKG_Energy_Status` and `MSR_PP0_Energy_Status` allow software to readout energy consumption by Package and Power Plane 0 domain respectively. Prior work [1] demonstrates how these MSRs can be used to successfully mount *remote* power side-channels on Intel platforms. The attack exploits the fact that, when executing workloads like AESNI, CMOS circuitry exhibits a power consumption pattern dependent on secret data, even across boundaries of Trusted Execution Environment (TEE) like Intel SGX. This implies these MSRs capture power consumption from cores that execute Intel SGX threads, and expose this consumption information to non-SGX processes. Likewise, subsequent works [2], [3] exploit the correlations between energy consumption and dynamic scaling of processor frequency to mount similar exploits as [1].

In response, Intel released a mitigation: *filtering* of RAPL MSRs [4]. The RAPL filtering mechanism scales noise in the RAPL measurements as the power consumption of the package increases, to harden against power side-channel attacks in [1]–[3]. This defense is exposed to software through the MSR `IA32_MISC_PACKAGE_CTLS`, which software can use to enable/disable filtering. Once enabled, filtering can no longer be disabled until the next reboot. It is general consensus that remote power attacks on patched Intel processors are infeasible. As such, in this work, we ask the following question:

With filtering enabled on RAPL MSRs, do there exist other interfaces which still report unfiltered energy consumption status to software?

In this work, we show that even when RAPL filtering is enabled, there exist interfaces which report unfiltered energy consumption, which an adversary can exploit remotely to compromise Intel SGX and TDX[1]. Concretely, on Intel server-grade processors (like Intel Xeon server products), there exists an out-of-band management interface (named Baseboard Management Controller or BMC) which can access the in-band processor Package Configuration Space (or PCS). The PCS comprises several telemetric readouts, including (but not limited to) package energy consumption. We show this PCS energy consumption readout is *unfiltered*, essentially allowing us to mount remote power side-channels on Intel Xeon processors, even when filtering is enabled. To summarize, we make the following contributions:

[1]Intel Trust Domain Extensions (TDX): the successor to Intel SGX for their Trusted Execution Environment solutions.

1) We first uncover the communication protocol structure between in-band processor package and out-of-band BMC, which is used by the BMC to access in-band processor components. We then detail two access commands allowing readout of in-band telemetric information: `RdIAMSR`, and `RdPkgConfig`. We then state our main observation that `RdPkgConfig` allows BMC access to unfiltered in-band package energy consumption.
2) There does not exist any documented mechanism of communicating from in-band processor to out-of-band BMC. This complicates aligning in-band workload execution with out-of-band telemetric measurements. To circumvent this, we construct a novel unidirectional communication channel to allow in-band software to signal the BMC when to start/stop collecting telemetric measurements.
3) Using the unfiltered package energy consumption reported by `RdPkgConfig`, we mount key recovery attacks on MbedTLS RSA executing within Intel SGX. This attack assumes the presence of a mechanism to perform single-step and zero-step.
4) Likewise, we also use `RdPkgConfig` to perform key recovery attacks against AESNI executing within Intel TDX without any single-step/zero-step requirement. For this, we first establish that energy consumption readouts from `RdPkgConfig` are outside the scope of TDX virtualization, implying `RdPkgConfig` energy consumption readouts leak the execution context of TDX. We then detail how to mount Correlation Power Attack (CPA) styled key recovery against AESNI executing within Intel TDX.

Responsible Disclosure: We responsibly disclosed our findings to Intel in June 2024. Intel completed their analysis by November 6, 2024. Intel has clarified side-channel attack using. OOB power telemetry as out of scope for SGX and TDX in this generation of products. Side channels based on in-band power telemetry are in scope for SGX and TDX.

II. Uncovering out-of-band and in-band communications

We first discuss details of how the out-of-band management interface is configured on our test setup, and how it communicates with in-band system management interface. We then discuss different kinds of in-band measurements possible to be readout from out-of-band, and conclude with key takeaways that eventually allow us to mount remote power side-channel attacks in-band even in the presence of filtering.

A. Threat Model and Experimental Setup

Throughout this work, all analysis and results are reported on Intel(R) Xeon(R) Platinum 8481C (i.e. Intel Xeon Generation 4) processor (codenamed: Xeon Sapphire Rapids) with microcode version `0x2b0004d0`. The Baseboard Management Controller on our system runs the OpenBMC Linux Distribution [5], which is a stripped down version of Linux for management controllers. It uses Yocto, OpenEmbedded, systemd, and D-Bus. In addition to this, the BMC also has three Field Replaceable Units (or FRUs): a single builtin FRU device, one external SOLUM IS162F22 FRU, and one external Intel FFPANEL FRU.

On the BMC side, we assume presence of vanilla software as shipped by the OpenBMC Linux Distribution (i.e. without any adversarial modifications). The adversary can only readout in-band telemetric measurements which the platform management protocols of Intel Xeon natively allow. On the in-band side, we assume all victim code to execute within a Trusted Execution Environment (either Intel SGX or Intel TDX). This implicitly assumes that all non-SGX (alternatively non-TDX) software is inherently untrusted.

As the later sections also detail, our "adversary" is a *combination* of the ① BMC and ② the in-band non-SGX/non-TDX software. Note that both the BMC as well as the in-band non-SGX/non-TDX software are outside the Trusted Computing Base (TCB) of Intel SGX/TDX. This makes a *combination* of the two to be also outside the Trusted Computing Base (TCB) of Intel SGX/TDX.

B. Platform Management Protocols

Intel Xeon systems are a combination of several heterogeneous components glued cohesively together. As such, there exist several system management protocols servicing each of these components. We summarize a few below:

- **In-Band Management**: This refers to managing the components within the silicon (i.e. the die) which communicate through a four-lane ring interconnect [6]. Management of the in-band resources can be done through Control and Status Registers (CSRs) or through Model Specific Registers (MSRs).
- **Message Control Transport Protocol** (MCTP): The MCTP protocol is used over the system management bus (SMBus) for management of peripherals (like Intel FPGA Programmable Acceleration Card) over Inter-Integrated Circuit (I2C) or Peripheral Component Interconnect Express (PCIe).
- **Intelligent Platform Management Interface** (IPMI): This protocol allows the BMC to poll platform sensors for availability, to efficiently readout sensor data, and make such data available to higher level management systems. IPMI predominantly helps in ensuring platform health by consistently reporting on data-points including (but not limited to) core voltage, core temperature, fan speed, Dual-In-Line Memory Module (DIMM) temperatures, and so on.
- **Platform Environment Control Interface** (PECI): PECI is an Intel proprietary interface allowing out-of-band management of in-band processors. PECI is predominantly used for power management and real-time configuration of in-band processor features.

Out of these protocols, IPMI and PECI report telemetric information, and are thus of direct interest to us[2]. IPMI

[2]As a side-note, SMBus has been exploited in [7] for BMC based fault attacks. Fault attacks are outside the scope of our analysis though.

reports telemetric information like CPU energy consumption, CPU average power, DIMM average power, each logical CPU's temperature, fan speeds, and a lot more. Likewise, PECI allows access to in-band measurement interfaces like Model Specific Registers (MSRs) and Package Configuration Space (PCS) which also allow readouts of similar telemetric information. However, PECI offers much more granular measurements than IPMI. To summarize:

Takeaway: Telemetric Measurements from PECI vs IPMI. IPMI offers coarse-granular measurements (i.e. in units of Watts and degree Celsius) which are unsuitable for the adversarial objectives considered in this work. On the other hand, PECI allows readouts from MSRs and PCS, which report more fine-grained measurements (like micro-joules for energy consumption) and are suitable for our adversarial objectives.

We thus restrict our discussion to PECI henceforth. IPMI still may be of interest for *other* adversarial objectives like neural network model fingerprinting as considered in [8] and we leave its investigation to future work.

C. Reversing PECI protocol structure

PECI is an Intel proprietary technology, and as such, not much is documented about it. However, Intel actively contributes to maintaining the PECI ioctl [9] in the OpenBMC Linux Distribution [5], and it provides a handle to reverse engineer PECI protocol structure.

Each PECI command has a fixed base address of `0x30` against which requests are made by the BMC over PECI. A set of completion codes signal what kind of response was received for the PECI request. For instance `PECI_DEV_CC_SUCCESS` (i.e. `0x40`) signals a successful response to a PECI request. Likewise, `PECI_DEV_CC_UNAVAIL_RESOURCE` (i.e. `0x82`) signals the requested resource was unavailable. In addition to this, out of the commands supported by PECI on Intel Xeon, only `GetTemp` and `RdIAMSR` have documented counterparts in-band. We also note that although `WrIAMSR` is an available command in [9], it is (by default) disabled on the Intel Xeon platform experimented upon in this work. For our adversarial objectives, we are only interested in `RdIAMSR` and `RdPkgConfig` since they allow readouts of telemetric information like processor package power consumption.

1) `RdIAMSR`: We first detail `RdIAMSR` since it offers direct access to in-band MSRs, and hence aligns with our adversarial goals. We summarize the protocol structure of `struct peci_rd_ia_msr_msg` on our platform in Fig. 1. For the `RdIAMSR` Request packet, the write length, read length, and the command are always fixed at `0x05`, `0x09`, and `0xB1` respectively. Hyperthread ID allows defining the logical processor whose MSR to read, and MSR address defines the address of the MSR to read. The Response packet comprises the completion code and (if successful) the value of MSR which was read. We now perform two experiments to further investigate `RdIAMSR`'s mapping with in-band MSRs.

In the first experiment, we fix a generic AESNI code [3] on a CPU core `x` in-band, and measure `IA32_FIXED_CTR0`[4] on CPU core `x` through `msr-tools`[5] from another CPU core `y`. Our experimental setup internally uses `smp_call_function_single`[6], which runs the majority of measurement code on CPU core `y`, but schedules a *low-latency* function on CPU core `x` leveraging the operating system scheduler. In our case, this low-latency function is a simple `rdmsr` on `IA32_FIXED_CTR0`. This setup allows CPU core `y` to read MSRs of CPU core `x` without adding too much noise to MSRs of CPU core `x`. Once we have in-band measurements of `IA32_FIXED_CTR0` on CPU core `x`, we now leverage `RdIAMSR` by fixing the hyperthread ID (see Fig. 1) to `x` and MSR address as the address of `IA32_FIXED_CTR0`. We then re-run AESNI *in-band* on CPU core `x` and collect measurements through `RdIAMSR`. We have the following observation:

Takeaway: Hyperthread mapping in-band and out-of-band is identical. MSR measurements in-band on CPU core `x` are *highly* correlated (**average correlation** 0.84 **across** 100 **experimental runs**) with measurements through `RdIAMSR` with Hyperthread ID set to `x`. This suggests that the logical-core-to-hyperthread-ID mapping out-of-band is exactly same as that in-band, saving the adversary additional effort of first reverse engineering this mapping.

Our second experiment is identical to the first, except that we now measure the RAPL MSR `MSR_PKG_ENERGY_STATUS` in place of `IA32_FIXED_CTR0`. We first disable RAPL filtering through `IA32_MISC_PACKAGE_CTLS`[7] and collect RAPL MSR measurements both in-band and out-of-band. Then, we re-enable RAPL filtering by setting the relevant bit of `IA32_MISC_PACKAGE_CTLS` and redo measurements both in-band and out-of-band. We have the following observation:

Takeaway: Both in-band and out-of-band RAPL MSR measurements are filtered. RAPL MSR measurements in-band on CPU core `x` are *highly* correlated (**average correlation** 0.74 **over** 100 **experimental runs**) with measurements through `RdIAMSR` with Hyperthread ID set to `x` for both cases (when filtering was enabled and when it was disabled).

Our experiments lead to the conclusion that enabling filtering in-band also enables filtering on `RdIAMSR` measurements. Moreover, the second MSR reported in [1] (i.e. `MSR_PP0_Energy_Status`) is disabled on Intel Xeon platform. *This makes out-of-band measurements of RAPL MSRs*

[3]https://github.com/intel/cryptography-primitives

[4]Fixed performance counter: counts number of x86 retired instructions.

[5]https://github.com/intel/msr-tools

[6]https://github.com/torvalds/linux/blob/master/kernel/smp.c

[7]And also by additionally disabling Intel SGX and Intel TDX through platform BIOS.

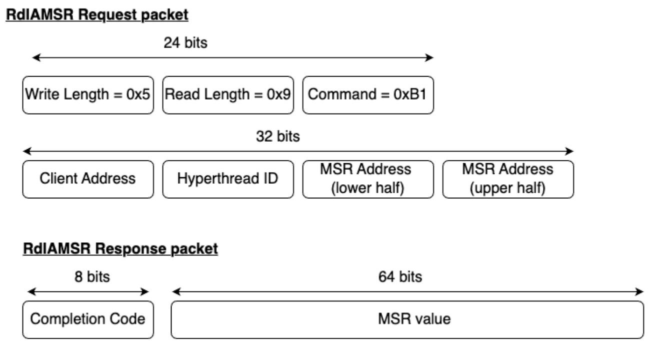

Fig. 1. Uncovered protocol structure of `RdIAMSR` on Intel Xeon (Saphhire Rapids). Each individual box is 8-bits wide.

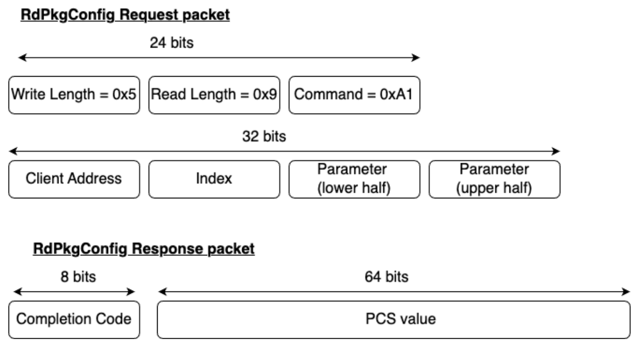

Fig. 2. Uncovered protocol structure of `RdPkgConfig` on Intel Xeon (Saphhire Rapids). Each individual box is 8-bits wide.

through `RdIAMSR` unsuitable for our adversarial goals. We thus shift focus on `RdPkgConfig`.

2) `RdPkgConfig`: We now detail `RdPkgConfig` as it also allows access to telemetric information, and aligns with our adversarial objectives. We summarize the protocol structure of `struct peci_rd_pkg_cfg_msg` in Fig. 2. Compared to the protocol structure of `RdIAMSR`, there are two differences: ① the command is `0xA1`, and ② `RdPkgConfig` takes an 8-bit `index` and 16-bit `parameter` which signifies the aspect of the Package Configuration Space (PCS) to be read. A list of `index` can be found in the definition of `struct peci_rd_pkg_cfg_msg` in [9]. The `parameter`, on the other hand, is undocumented and needs to be fuzzed across all 16-bit possibilities. The Response packet has a completion code and a 64-bit value of the PCS parameter if the request was successful.

We now design the following experiment. First, on the in-band side, we execute a generic AESNI code [8] on a fixed CPU core. We also attach high-priority threads to other CPU cores on the package with negligible workload intensity to reduce noise in captured energy consumption. Then, on the out-of-band side, we first fix the `index` of our `RdPkgConfig` request packet to `PECI_MBX_INDEX_ENERGY_COUNTER`[9] (i.e. the "Energy counter") and fuzz through all possibilities of the 16-bit `parameter`. We summarize our observation:

Takeaway: `RdPkgConfig` with `index = 3` and `parameter = 0xff` readout PCS energy consumption at intervals of 1 **millisecond**. Moreover, any `parameter` value other than `0xff` readouts a 0, implying `0xff` is the only `parameter` for `index = 3` which readouts valid energy consumption data. Finally, we observe a *poor* correlation (**average correlation** 0.04 **over** 100 **runs**) between data collected from `RdIAMSR` and data from `RdPkgConfig`, with standard deviation of data from `RdPkgConfig` being lesser (i.e. less noise).

[8]https://github.com/intel/cryptography-primitives

[9]There are other `index` which also report RAPL domain data, like `PECI_MBX_INDEX_PKG_RAPL_PERF` but we did not observe any exploitable leakage with it. It is an interesting future work direction to try out other `index` wrt. the adversarial objectives we explore here.

Recall that the RAPL filtering algorithm [4] explicitly adds more noise, increasing the standard deviation of RAPL MSR data. In our experiments, we observe a poor correlation of `RdPkgConfig` with the data from `RdIAMSR`[10] (with filtering enabled), as well as the fact that the distribution of `RdPkgConfig` is much *thinner* wrt. standard deviation (refer Fig. 3). For completeness, we also correlate readouts from `RdPkgConfig` with the *unfiltered* `RdIAMSR` readings; in this experiment, we observe a strong correlation: 0.712. We thus conclude that energy readings from `RdPkgConfig` (`index=3` and `parameter=0xff`) are not filtered. In the next section, we use this fact to craft remote power side-channels.

III. Using `RdPkgConfig` to craft remote power side-channel attacks on Intel SGX and TDX

In this section, we explain how the `RdPkgConfig` (`index=3` and `parameter=0xff`) measurements can be utilized to mount remote power side-channel attacks on Intel Xeon Generation 4 processors in presence of Trusted Execution Environments like Intel SGX and TDX. We first reason why energy consumption is visible across Intel SGX/TDX trust boundaries. We then state (and fix) a roadblock to mounting successful attacks: synchronizing victim execution in-band and adversarial measurements out-of-band by using intentionally disabled MSR `IA32_PMC0`. Finally, we state our attack results.

A. Investigating Visibility of Energy Consumption outside Intel SGX/TDX Trust Boundaries

One precondition of a successful remote power attack is the fact that energy consumption of Intel SGX/TDX is visible across their Trusted Computing Base (TCB) boundary to a non-SGX/non-TDX process. We investigate this further here.

1) Intel SGX: The work in [1] already demonstrates that energy consumption is visible across Intel SGX boundaries. We attribute the success of [1] to the fact that RAPL MSRs are Package scoped (i.e. a single MSR services *all* cores in the Package), while Intel SGX context is applicable only on the logical core where it is executing. Contrastingly to this

[10]We read `MSR_PKG_Energy_Status` from the out-of-band interface.

observation, MSR `IA32_SPEC_CTRL`[11] is a Core scoped MSR. Thus, when Intel SGX context is applicable on a CPU core, this MSR can be made to behave differently (like disallowing any disables on the mitigations) than it would under non-SGX context [10]. However, a similar restriction cannot be made for a Package scoped MSR (like RAPL MSRs) since they are associated with *multiple* CPU cores (and thus multiple execution contexts). It is possibly for this reason why Intel (in response to [1]) opted for a filtering countermeasure rather than disabling RAPL MSR when Intel SGX context is operational. We note that `RdPkgConfig` (`index=3` and `parameter=0xff`) is also a Package scoped measurement (since it is readout from the Package Configuration Space), and is thus outside the scope of Intel SGX trust boundaries following a similar reasoning.

2) Intel TDX: Intel TDX follows a slightly different approach to defining its trust boundaries than Intel SGX. Concretely, Intel TDX performs MSR virtualization [11] which virtualizes a given set of MSRs when TDX is operational. Intel TDX also supports context-switching the virtualized MSRs as well. This approach allows TDX to access its own virtualized MSRs as well as save MSR context across context-swiches without reflecting the increments in TDX's MSRs on the MSRs of the non-TDX processes. A concrete example of such virtualized MSRs are Performance Monitoring Counters (PMCs) [11]. However, we note the following:

Takeaway: RAPL MSRs are not virtualized by Intel TDX. Intel TDX chooses not to explicitly virtualize RAPL MSRs [11], and willingly accepts reflecting energy consumption of Intel TDX execution context to the Package scoped RAPL MSRs. We hypothesize that such a design choice was made since RAPL MSRs are anyway protected by RAPL filtering. However, **with the introduction of our side-channel through `RdPkgConfig`, the design choice of not virtualizing energy consumption allows us to craft remote power-channels on Intel TDX.**

B. Aligning in-band execution and out-of-band measurements

The experiments detailed in the previous section aimed at uncovering properties of `RdPkgConfig`, and thus we resorted to manual synchronization. However, for actual exploits this violates the threat model we cannot assume any control over victim code within Intel SGX/TDX. We thus construct an unidirectional communication channel from in-band non-SGX/non-TDX context to out-of-band `RdPkgConfig` that helps synchronize the two. To do so, we leverage the fact that non-SGX/non-TDX MSRs related to performance monitoring unit accept general purpose programming. We can thus *disable* these MSRs (i.e. make their values architecturally invalid), and then use them for communicating between in-band and out-of-band. The detailed experimental setup is enumerated below. We have three entities: Intel SGX/TDX execution context (abbr. **V**), in-band non-SGX/non-TDX execution context (abbr. **A1**), and out-of-band `RdPkgConfig` measurement context (abbr. **A2**). **V** is the victim, and the *combined* adversary is **(A1, A2)**.

1) **A1** first disables MSR `IA32_PMC0` by programming its control MSR `IA32_PERFEVTSEL0`. **A1** ensures `IA32_PMC0` remains disabled throughout the duration of the attack. Meanwhile, **A2** continuously polls the LSB of `IA32_PMC0` through `RdIAMSR`.
2) **A1** then sets the least-significant-bit (LSB) of `IA32_PMC0`. This triggers **A2** to initiate a measurement from `RdPkgConfig` (`index=3` and `parameter=0xff`). Call this measurement M_1.
3) **A1** then triggers execution of **V** by relinquishing a CPU core to let **V** execute. **A1** also attaches several high-priority threads (with negligible workload intensity) on other CPU cores in the package to reduce overall noise in the reported energy consumption of the package.
4) Once **V** is done executing, **A1** unsets the LSB of `IA32_PMC0`. This triggers **A2** to initiate another measurement from `RdPkgConfig` (`index=3` and `parameter=0xff`). Call this measurement M_2.
5) **A2** reports a *single* sample point as $(M_2 - M_1)$, which captures the energy consumed throughout execution of **V**. The entire process is reported for N sample points.

Note that **V** can be execution context of either Intel SGX or Intel TDX. We now use this setup to mount remote power side-channels on Intel SGX and TDX.

C. Leaking RSA keys across Intel SGX trust boundary

We now show how to use the power side-channel from `RdPkgConfig` to mount side-channel attacks on the control flow of RSA executions, eventually extracting the RSA private keys from MbedTLS implementation within SGX. We now summarize our attack principle: Depending upon the private key bits, RSA implementations take varying control flow paths which execute different sequence of instructions. Isolation of specific instructions in these different control flow paths and an eventual statistical comparison of their associated power side-channel footprints leads to leaking secret dependent control flow. Moreover, as was the case in [1], we make an additional assumption: single/zero stepping (like [12] for instance) is operational on Intel SGX.

We perform our attack on MbedTLS implementation (version 3.5.1) running inside Intel SGX. We follow similar attack semantics as [1]. MbedTLS supports the windowed modular exponentiation implementation, where a left-to-right sliding window of size d (dependent on the size of input secret exponent) contributes to the computation of the final result in each iteration. Briefly, an adversary needs to perform three sequential steps: ① leak the number of leading zeros, ② determine the secret exponent dependent square/multiply control flow sequence, and ③ from the leaked control flow sequence, recover the secret key.

It is sufficient to perform steps ① and ②, and then rely upon known key extraction algorithms [13] to recover the

[11]Controls enabling/disabling mitigations against speculative attacks.

secret key. If the bit of the secret exponent is 1, a square operation occurs which relies upon Montgomery multiplication (`mpi_montmul`). Among other operations, `mpi_montmul` relies upon a `memset` to initialize and segregate memory. In Intel SGX, this `memset` is replaced by Intel's `fast_memset` implementation, which in turn uses AVX instructions. In summary, whenever the current bit of the secret exponent is 1, an AVX memory instruction (like `movdqa`) is executed; otherwise, a non-scalar `x86` instruction (like `movzx`) is executed. Using a similar observation as [1], the energy consumption profile of `movdqa` and `movzx` is substantially different once they are zero-stepped, easily allowing an adversary to recover secret-dependent control flow, and hence the secret. In our experiments with RSA-2048 (for which MbedTLS chooses the default window size as 5), we invoke zero-steps and measure 10000 samples from `RdPkgConfig` at a sampling rate of 1 millisecond. Subsequently, **we are able to leak the 97.4% of the 2048 bit RSA key in about 5 hours**[12].

D. *Leaking AESNI keys across Intel TDX trust boundaries*

For AESNI, we follow a similar setup as for RSA but with two main differences: ① we place the AESNI code within Intel TDX, and ② we require no single/zero step assumption.

We follow a similar AESNI setup as [1] and [3]. Within Intel TDX, we implement AESNI through Intel IPP library [13]. As a first experiment, to establish that different keys to AESNI can be distinguished by our side channel, we perform two similar experiments. In the first experiment, the AESNI key is `0x0000000000000000`, while in the second experiment, the AESNI key is `0xffffffffffffffff`. We show the result in Fig. 3, where the energy consumption footprint of the two experiments is clearly distinguishable.

Next we perform CPA on AESNI by choosing the hypothetical power model as the Hamming Weight of Round-0 (in order to recover the round 0 key, from which other keys can be trivially leaked by simulating the AESNI key schedule). We collect 10000 sample points in each trace, with 100000 invocations of `ippsAESEncryptECB` (16 bytes of plaintext in each invocation) contributing to a single sample point. We collected over a million traces for the overall CPA experiment. Overall, we collect over 400 hours worth of data in our setup. We were successfully able to recover **13 bytes out of the unknown 16 bytes of victim AESNI key executing within Intel TDX**[14], thereby demonstrating the applicability of using `RdPkgConfig` to mount CPA attacks on AESNI executing within Intel TDX.

IV. Conclusion

Following the public disclosure of remote power attacks through RAPL MSRs, Intel followed up with a "filtering" defense. We shared our findings regarding `RdPkgConfig` side channel on SGX and TDX with Intel. Intel has clarified

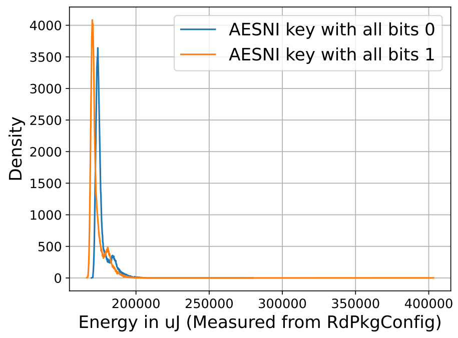

Fig. 3. Distinguishing two AESNI keys through `RdPkgConfig` measurements. T-test score of these experiments is 12.41, implying the two distributions are clearly distinguishable.

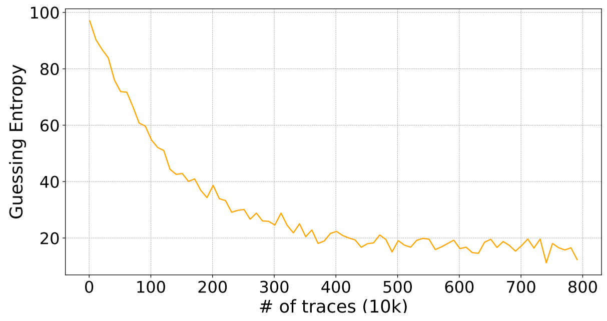

Fig. 4. Guessing Entropy trend (for AESNI key) with increase in number of collected traces.

to us that it was a conscious design decision made based on upon conversations with its customers and their threat models. Intel considers side-channel attack using OOB power telemetry as out of scope in this generation of products. Our work underlines the importance of a OOB power telemetry as a potential source of side channel to confidential compute and it should be comprehended as the threat model for SGX and TDX evolves in the future.

Acknowledgment

The authors would like to thank the anonymous reviewers whose comments helped in improving the final version of the manuscript. Nimish Mishra, Sarani Bhattacharya, and Debdeep Mukhopadhyay would additionally like to acknowledge the Centre on Hardware-Security Entrepreneurship Research and Development (C-HERD), MeitY, Govt. of India, and Information Security Education and Awareness (ISEA) initiative, MeitY, Govt. of India, for partially funding this research.

[12]Remaining secret bits can be easily leaked through lattice reduction.

[13]https://github.com/intel/cryptography-primitives

[14]Entropy of remaining 3 bytes is insufficient to prevent direct extraction, thereby allowing leakage of the entire 16 byte key.

References

[1] M. Lipp et. al., “Platypus: Software-based power side-channel attacks on x86,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 355–371.

[2] Y. Wang et. al., “Hertzbleed: Turning power {Side-Channel} attacks into remote timing attacks on x86,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[3] C. Liu et. al., “Frequency throttling side-channel attack,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[4] Intel, “Rapl advisory intel-sa-00389,” 2022.

[5] OpenBMC, “Openbmc linux distribution,” https://github.com/openbmc/openbmc/tree/master, 2024.

[6] R. Paccagnella et. al., “Lord of the ring (s): Side channel attacks on the {CPU}{On-Chip} ring interconnect are practical,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 645–662.

[7] Z. Chen et. al., “Pmfault: Faulting and bricking server cpus through management interfaces: Or: A modern example of halt and catch fire,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–23, 2023.

[8] N. Mishra et. al., “Too hot to handle: Novel thermal side-channel in power attack-protected intel processors,” in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 378–382.

[9] Intel, “Peci ioctl,” https://github.com/openbmc/linux/blob/dev-5.4/include/uapi/linux/peci-ioctl.h, 2024.

[10] ——, “Speculative store bypass. intel-sa-00115,” 2018.

[11] ——, “Intel tdx specification documentation,” 2023.

[12] D. Skarlatos et. al., “Microscope: Enabling microarchitectural replay attacks,” in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 318–331.

[13] D. J. Bernstein et. al., “Sliding right into disaster: Left-to-right sliding windows leak,” in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 555–576.