Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels

Andreas Kogler$^1$ Jonas Juffinger$^1$ Lukas Giner$^1$ Lukas Gerlach$^2$
Martin Schwarzl$^1$ Michael Schwarz$^2$ Daniel Gruss$^1$ Stefan Mangard$^1$

$^1$Graz University of Technology $^2$CISPA Helmholtz Center for Information Security

Abstract
Differential Power Analysis (DPA) measures single-bit differences between data values used in computer systems by statistical analysis of power traces. In this paper, we show that the mere co-location of data values, e.g., attacker and victim data in the same buffers and caches, leads to power leakage in modern CPUs that depends on a combination of both values, resulting in a novel attack, Collide+Power. We systematically analyze the power leakage of the CPU’s memory hierarchy to derive precise leakage models enabling practical end-to-end attacks. These attacks can be conducted in software with any signal related to power consumption, e.g., power consumption interfaces or throttling-induced timing variations. Leakage due to throttling requires 133.3 times more samples than direct power measurements. We develop a novel differential measurement technique amplifying the exploitable leakage by a factor of 8.778 on average, compared to a straightforward DPA approach. We demonstrate that Collide+Power leaks single-bit differences from the CPU’s memory hierarchy with fewer than 23,000 measurements. Collide+Power varies attacker-controlled data in our end-to-end DPA attacks. We present a Meltdown-style attack, leaking from attacker-chosen memory locations, and a faster MDS-style attack, which leaks 4.82 bit/h. Collide+Power is a generic attack applicable to any modern CPU, arbitrary memory locations, and victim applications and data. However, the Meltdown-style attack is not yet practical, as it is limited by the state of the art of prefetching victim data into the cache, leading to an unrealistic real-world attack runtime with throttling of more than a year for a single bit. Given the different variants and potentially more practical prefetching methods, we consider Collide+Power a relevant threat that is challenging to mitigate.

1 Introduction
Power analysis attacks exploit differences in the power consumption of hardware circuits for different operations or data operands [2, 17, 26]. Most power analysis attacks use external measurement equipment on small devices such as smart cards [29, 30] to extract their internal secret information (e.g., cryptographic keys). Without physical access, an attacker can still resort to software-based side channels, exploiting, e.g., timing [15], or the cache state [28, 42]. While the attack techniques are generic, these physical and software-based side-channel attacks are typically specific to a victim application, e.g., a cryptographic implementation. Hence, for all of these side-channel attacks, there is clear guidance on how developers can mitigate them, e.g., constant-time implementations, blinding, masking, or adding randomness [24].

Meltdown [20] and MDS [3, 34, 37] can still leak secret information even when developers followed best practices on mitigations. In this sense, they are generic attacks that are not tailored to a specific algorithm but rather a CPU, meaning they can leak arbitrary data from a victim context regardless of the algorithm executed. However, Meltdown and MDS are mitigated through hardware and software patches.

Recently, software-based power side channels gained more traction [8, 9, 19, 21, 25, 27, 33, 38, 41], especially after the discovery that software-level interfaces are precise enough to mount power analysis attacks on cryptographic implementations [19]. One stop-gap solution against these attacks is making the corresponding software-level interfaces privileged (e.g., Intel RAPL). However, adaptive power management leads to constant frequency adjustments to comply with energy and heat limits. When the CPU works with data operands that consume more energy, the CPU reaches these limits more frequently. Consequently, power consumption variations translate directly into timing differences [21, 38]. Hence, software-based power side channels are still practical. Still, so far, they targeted specific applications and have not been able to demonstrate Meltdown-style or MDS-style generic leakage from arbitrary memory locations and victim contexts.

In this paper, we present Collide+Power, a novel attack showing that software-based power side channels constitute a much more fundamental and generic security threat. Our central observation is that the mere co-location of data values, e.g., attacker and victim data in buffers and caches, in modern CPUs introduces subtle but exploitable power leakage that
depends on the combination of both values. This combination has several components, including e.g., the Hamming distance between attacker and victim data. Thus, we can exploit this combined leakage by varying the attacker-controlled data value and learning the precise victim value from the combined power leakage. Consequently, Collide+Power overcomes all power side channels, enabling practical attacks leaking 4.82 bit/h from other security domains.

The foundation of our end-to-end attacks is amplifying the subtle leakage signal, which is far below other components of the power consumption. For this purpose, we develop a novel differential measurement technique where each sample is based on two measurements with inverted attacker-controlled values. Due to the nature of the power leakage in the CPU’s memory hierarchy, this approach cancels out unwanted noise terms and amplifies the desired signal by a factor of 8.778 on average compared to a straightforward DPA approach.

Another building block for our end-to-end attacks is a precise power leakage model of the CPU’s memory hierarchy. Prior models [19, 21, 38] have not captured the subtly combined leakage we exploit and cannot be used in our attack to leak data from arbitrary memory locations. We develop precise leakage models for various attack scenarios, depending on the microarchitectural element where the attacker and victim data are colliding, e.g., leakage models considering L1 cache eviction when targeting L1 caches. Based on our precise models, we demonstrate that Collide+Power can even leak single-bit differences with fewer than 23,000 measurements.

In our end-to-end attacks, Collide+Power varies attacker-controlled data during differential power analysis. In this work, we demonstrate two powerful attack scenarios: The first is MDS-Power, an MDS-style attack [34, 37], leaking arbitrary data accessed by the victim (data in use), without assumptions on the algorithm run by the victim. Our end-to-end MDS-Power attack leaks 4.82 bit/h from another security domain co-located on a sibling hardware thread. The second scenario is Meltdown-Power, a Meltdown-style attack [20], leaking data at rest, with 0.136 bit/h (with amplification) from arbitrary memory locations in the kernel, using the same attack mechanism as Meltdown to interact with the cache hierarchy [13, 35]. Nevertheless, our Meltdown-Power proof-of-concept has severe practical limitations due to the state-of-the-art of prefetching data into the memory hierarchy in a real-world scenario, leading to an unrealistic attack runtime of more than a year per bit with throttling. However, discovering faster ways to prefetch data into the memory hierarchy improves the leakage rates of Collide+Power.

Collide+Power is a generic attack that works on any modern CPU that co-locates attacker and victim data in the microarchitecture, e.g., caches. Thus, conceptually, it is the same step as from cache side channels to Meltdown applied to power side channels (cf. Table 1). Our Collide+Power attack framework is agnostic to the type of leakage traces and works with traces from the RAPL interface and timing differences without any modifications alike \(^1\). Our evaluation shows that an end-to-end Collide+Power attack instantiated with timing side-channel traces only requires 133.3 times more samples than direct power measurements. To facilitate the genericity and reproducibility of our results, we perform most of the evaluation and analysis with the generic RAPL interface.

We conclude that software-based power analysis attacks are more generic and extend beyond the leakage of well-known and structured attack targets from cryptographic contexts. In contrast to attacks relying on the design of microarchitectural elements such as Spectre or Flush+Reload, Collide+Power, like Rowhammer, exploits fundamental physical properties present in CPUs. Therefore, mitigating these attacks poses a much larger challenge than previous works anticipated, and fully mitigating Collide+Power, regardless of whether the victim performs side-channel hardened cryptographic or general-purpose operations, remains a significant challenge.

To summarize, we make the following contributions:

1. We systematically analyze the leakage of different operations on the memory hierarchy and develop a leakage model including the attacker-victim combined leakage.
2. We present a novel and generic differential measurement technique combining multiple guess measurements per victim data value, amplifying the leakage by 8.778x.
3. We demonstrate unprivileged end-to-end Collide+Power attacks with throttling, leaking arbitrary secret data without targeting the specific algorithm the victim uses.
4. We evaluate Collide+Power in theoretical and practical end-to-end attacks and, in the more practical attacks, observe average leakage rates of 4.82 bit/h, with a success rate of 98.9 % (n=1000, σ=0.32 %).

Outline. Section 2 provides background, and Section 3 the high-level idea. Section 4 presents the leakage analysis, and Section 5 our novel differential measurement. We discuss the implementation in Section 6, the evaluation in Section 7, and mitigations and limitations in Section 8. Section 9 concludes.

Responsible Disclosure. We disclosed our findings to Intel on November 23, 2022, and ARM and AMD on February 9, 2023. Collide+Power was assigned CVE-2023-20583 and was held under embargo until August 1, 2023. The vendors responded with advisories and guidelines to mitigate the risk.

\(^1\)The source code of the framework and the proof-of-concepts can be found at: https://github.com/iaik/CollidePower

### Table 1: Collide+Power fills a significant gap, broadening software-based power analysis from attacks on specific algorithms to generic attacks like Meltdown and MDS.

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Software Implementation</th>
<th>Generic (CPU and Hardware)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flush+Reload [42]</td>
<td>Foreshadow [39]</td>
</tr>
<tr>
<td></td>
<td>BranchScope [7]</td>
<td>MDS [34, 37]</td>
</tr>
<tr>
<td>Software-based Power Side Channel</td>
<td>Platypus [19]</td>
<td>Collide+Power (our work)</td>
</tr>
<tr>
<td></td>
<td>Hertzbleed [38]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FTS-CA [31]</td>
<td></td>
</tr>
</tbody>
</table>
2 Background

In this section, we present background on memory within CPUs, transient-execution, and power-analysis attacks.

2.1 The CPU’s Memory Hierarchy

Modern CPUs have an internal hierarchy from small memories close to the execution pipeline to large memories such as an SRAM last-level L3 cache or even a DRAM-based victim L4 cache. Data is always served from the fastest hierarchy level that holds the data, dramatically lowering average memory access latencies. Buffers typically serve a dedicated purpose, e.g., load and store buffer (the memory order buffer) track load and store operations. Caches are the next larger storages, following a similar design across different CPUs: They are organized in \( n \)-way sets with cache-line sizes of 64 B. The smallest, L1 cache, is split between data (which we focus on) and instructions. The L2 cache is slightly larger and slower. The last-level L3 cache is significantly larger, organized in independent cache slices, and shared across cores.

Caches have been a popular target of side-channel attacks, such as Prime+Probe [22, 28, 31]. In a Prime+Probe attack, the attacker creates and uses an eviction set to constantly prime an entire cache set. When the victim accesses a cache line mapping to the primed cache set, an attacker-controlled entry is evicted, which the attacker can observe by the access latency to its own eviction set during the subsequent priming.

When data is used, it travels through the memory hierarchy and is placed in buffers and caches, involving busses and fill buffers to transmit or temporarily store the data. Besides caches, the line-fill buffer (LFB), a temporary storage for, e.g., data loads and evictions, uncacheable, and non-temporal accesses, has been exploited in different attacks [20, 34, 37].

2.2 Transient-Execution Attacks

Out-of-order and speculative execution contribute to performance substantially. Instructions are retired in order, and the outcomes of predictions and faults are checked before committing the results. Mispredictions are rolled back and undone. Instructions executed out-of-order or speculatively which are never committed, are called transient [4, 16]. Transient execution may change the microarchitectural state, e.g., cache accesses. If these state changes depend on secret data, attackers can extract the secrets by leveraging a side channel [4, 16]. Canella et al. [4] systematized transient-execution attacks into Spectre-type attacks and Meltdown-type attacks. Meltdown-type attacks [4, 20] leverage transient execution in out-of-order execution since exceptions are raised in the retirement phase of out-of-order execution. In contrast, Spectre-type attacks [4, 16] exploit transient execution caused by speculatively executing mispredicted branches. Various attacks have been demonstrated exploiting different prediction mechanisms in modern CPUs [4, 10, 14, 16, 23]. Spectre attacks on the kernel require code snippets (gadgets) in the kernel.

2.3 Power-Analysis Attacks

Power analysis attacks exploit differences in the energy consumption of a hardware circuit when computing with secret data. The switching behavior of CMOS circuits (the building block of CPUs), i.e., the transitions between ‘0’ and ‘1’ bits, dictates the power consumption in a data-dependent way. There are mainly two analysis methods: First, Simple Power Analysis (SPA) [17] uses direct observations in a power trace to identify secret information, e.g., different energy signatures of secret-dependent control flow. Second, Differential Power Analysis (DPA) [17] uses statistical methods like the difference of means or correlations (then also called Correlation Power Analysis, CPA [2]) to infer secret information when SPA is insufficient. Regardless of the power analysis technique, external measurement equipment is typically used to measure energy consumption.

Lipp et al. [19] transformed traditional power-analysis attacks into software-based attacks on modern x86 CPUs using the Running Average Power Limit (RAPL) [11] interface. They demonstrate SPA and CPA on modern x86 CPUs and extract AES-NI and RSA keys despite the comparably low sampling rate of the software interface. In response to this attack, the RAPL interface is no longer accessible to unprivileged users. Furthermore, to protect Intel’s Trusted Execution Environment (SGX), the interface no longer reports the exact energy consumption when SGX is enabled. However, as modern CPUs use adaptive power management to comply with thermal and power limits, the CPU frequency and performance depend on energy consumption. Computing on data that consumes more energy leads to more frequent throttling, observable by an unprivileged attacker, even in remotely measurable timing differences [38]. Statistical methods reduce the noise far enough to enable inference of processed data. Wang et al. [38] demonstrated software-based power side-channel attacks based on the CPU frequency as a proxy and replacement for energy consumption interfaces. Liu et al. [21] later also showed that frequency and timing could replace direct power consumption measurements. Both works leak cryptographic keys from other security domains, Wang et al. [38] even remotely across the network.

3 High-Level Overview of Collide+Power

In this section, we present the high-level idea of Collide+Power. Collide+Power is a software-based power side-channel attack exploiting the fundamental design of how modern CPUs handle data. We exploit that the mere co-location in the memory hierarchy, e.g., attacker and victim data in a cache, introduces subtle but exploitable leakage in the power consumption. As illustrated in Figure 1 for the example of
This Hamming distance between two distinct cache line values, the less energy is consumed, reaching its minimum for identical values and the maximum for inverse values. Thus, an attacker varying its own data value accordingly can infer the precise victim’s data value without being able to access it.

3.1 A Precise Model for DPA

Exploiting subtle power differences is challenging. Collide+Power uses DPA, and more specifically CPA, which can exploit arbitrarily small power differences as long as the number of measurements can be increased. This is the case in our attack scenario, as attacker-controlled data and victim data are co-located in the cache for an arbitrarily long time. Previous work by Lipp et al. [19] showed that repeating byte-wise loads on x86 CPUs follow the Hamming weight model. This Hamming weight model does not capture the leakage components that combine attacker-controlled and victim data, i.e., the Hamming distance. However, for Collide+Power, the Hamming distance between two distinct cache line values G and V of different security domains is the relevant signal for the attack. Furthermore, the power consumption is much more significantly influenced by other constant and non-constant factors, outweighing the signal by orders of magnitude. These factors include data-dependent noise, i.e., components depending on the attacker-controlled value G and the victim value V but not in a combined and exploitable way. These factors include independent noise, i.e., from other processes or environmental influences. Our generalized power model (cf. Section 4) for the CPU’s memory hierarchy,  

\[
\mathcal{P}(G,V) = a_0 \cdot \text{hd}(G,V) + w_0 \cdot \text{hw}(G) + w_1 \cdot \text{hw}(V) + \omega, \tag{1}
\]

includes all of these factors as well as the Hamming weights and the Hamming distance, where the guess G is attacker-controlled, and V is the targeted constant secret victim value. From Toy Example to Real-World Attack. As the power observations directly only reveal a combination of Hamming distance between G and V and their Hamming weights, an attacker needs to vary the parameter G to infer the exact value of V. Figure 2 shows a simplified instance of this problem. By choosing guesses \( G = 2^i, i \in \mathbb{N} \) with constant Hamming weights, i.e., hw(G) = 1, the data-dependent noise of Equation 1 is constant, but the Hamming distance hd(G, V) changes based on G. Thus by finding guesses that reduce \( \mathcal{P} \), we can infer that the corresponding bit is also set in the secret victim value V. While this toy example provides a great intuition of the basic idea, Collide+Power uses a more sophisticated and robust correlation-based approach. Leaking the value V is a search for the maximum correlation between the model and observations. Our CPA uses the recorded samples as observations \( \mathcal{O} \) and maximizes the correlation for a given model \( \mathcal{M}(G,V) \). Crucial for the success rate of the CPA is the accuracy of the model and its component and the ratio between the signal and the noise (cf. Equation 1), i.e., the signal-to-noise ratio. Therefore, we design in-depth experiments to recover the structure in Sections 4.1 and 4.2 and the coefficients for the different components in Section 4.3.

Differential Measurement. To increase the signal-to-noise ratio further, we exploit that the power consumption can be influenced by two extremes: the maximum Hamming distance
and the \emph{minimum} Hamming distance. Instead of just one measurement, with one attacker-controlled value and an unknown victim value, we perform two measurements in a differential measurement technique (cf. Section 5). The first measurement takes a chosen \( G \), yielding \( P_G \), and the second takes the inverted guess \( \tilde{G} \), yielding \( P_{\tilde{G}} \). By using the difference between \( P_G \) and \( P_{\tilde{G}} \) as a single combined sample, this measurement technique increases the signal-to-noise ratio by a factor of 8.778 on average. It not only doubles the weight of the Hamming distance in the power leakage but also reduces other constant and non-constant factors.

3.2 End-to-End Attacks

Collide+Power is a generic attack for various scenarios and environments. The commonality is that the attacker triggers a situation where attacker-controlled values \( G \) and victim-controlled values \( V \) compete for storage in the same shared microarchitectural element. Depending on the target, this may involve cache eviction or flushing, loading of cache lines, or read and write accesses without cache interaction. Consequently, Collide+Power has no more requirements than typical cache attacks, \( i.e. \), the ability to perform memory accesses. We build two end-to-end attack variants on Collide+Power:

In \textit{MDS-Power}, we target data in use by a victim program. The victim constantly uses a secret value, \( i.e. \), in a loop, which effectively keeps it in the memory hierarchy of the core (e.g., buffers like the LFB or the L1 cache). In this scenario, identical to MDS attacks, Collide+Power leaks precise secret values from a victim co-located on a sibling thread with 4.82 bit/h.

In \textit{Meltdown-Power}, we target data at rest from arbitrary memory addresses. The attacker uses the same mechanisms as in Meltdown to pull victim data into caches, \( i.e. \), leakage of data is facilitated by prefetch gadgets in the kernel [35], which are still present in kernels today [4, 13, 40]. Meltdown-Power leaks amplified data values with 0.136 bit/h in exactly the same scenario, making it a drop-in replacement for the now mitigated Meltdown attack. Finally, mounting Meltdown-Power in a real-world setting, \( i.e. \), using frequency throttling attacks [21, 38], we estimate that an attacker requires 2.86 years to leak an amplified bit from the kernel. However, this low security risk might drastically change if new architectural or microarchitectural ways of prefetching victim data in co-location with attacker-controlled data are discovered.

3.3 Threat Model

We assume the attacker runs unprivileged native code. If specific interfaces, \( e.g. \), RAPL, are unavailable [19], Collide+Power has the minimal requirement that the attacker can measure time (\( e.g. \), with \texttt{rdtscp}), serving as a proxy for the power consumption [21, 38]. Collide+Power is agnostic to the type of leakage traces and works identically with any direct or indirect power trace. Furthermore, similar to prior works on software-based power side channels [38], we either \textit{stress} the other CPU cores, \( i.e. \), a multi-threaded attack, or use the throttling effect through default or adjusted power limits, translating energy consumption into timing differences. The only additional assumption MDS-Power makes is that attacker and victim are co-located on sibling threads of a physical core, identical to the threat model of RIDL and ZombieLoad [34, 37].

Meltdown-Power has no core co-location requirement. As in Meltdown, we assume the attacker has a target address to attack [20]. When targeting the CPU’s caches with Meltdown-Power, interaction with the caches is required, \( i.e. \), regular memory accesses to load and evict data from L1 and L2 cache. Meltdown-Power also requires the presence of a prefetch gadget in the kernel, which Meltdown also requires for non-L1 cache data leakage [35]. Recent work confirmed that prefetch gadgets are still present in kernels today [4, 13, 40]. Like Meltdown, Meltdown-Power uses the prefetch gadget to load the victim cache line \( V \) into the cache. We evaluate Meltdown-Power with two different gadgets: an artificial Spectre-RSB prefetch gadget for a comprehensive evaluation and a real-world Spectre-PHT prefetch gadget to demonstrate the signal. We detail and evaluate all attacks in Sections 6 and 7.

4 Memory-Hierarchy Leakage Analysis

In this section, we analyze the power leakage of the CPU’s memory hierarchy and derive a precise power leakage model. We find the general structure of the model in Section 4.1, analyze the effect of data location and bus widths in Section 4.2, and compute the precise coefficients in Section 4.3.

4.1 Determining the Structure of the Leakage

In this section, we design experiments to generate activity in certain cache levels. We analyze all pairwise combinations of \textit{Hamming distance} and \textit{Hamming weight} between slices of values within attacker-controlled and victim cache lines, revealing the components of the power leakage. We use an \textit{Intel Core i7-8700K CPU} for our analysis. Section 7.1 shows that this analysis applies to a broad range of CPUs. We find three zones within a cache line that influence the leakage strength and structure, which can be represented by \textit{Hamming distance} and \textit{Hamming weight} expressions.
Accessing and Filling the Cache Lines. For our analysis, we focus on two different values placed in the cache lines: the attacker controlled guess $G$ and the victim value $V$. Listing 1 shows the code used to access the cache lines with movq loads, i.e., 8 B loads, and generate one sample for a randomly chosen $V$ and $G$. Each sample is generated by measuring the RAPL energy consumption over a tight loop running $L$ iterations of 16 accesses interacting with the memory hierarchy. The number of iterations determines the measurement duration. According to our analysis, hardware prefetchers also have a minor influence on the leakage. However, we disable hardware prefetchers to derive a precise leakage model that the attacker can utilize. We detail the influences of the hardware prefetchers in Section 7.6 and emphasize that the attacks also work with hardware prefetchers enabled. MDS-Power does not trigger any prefetching as all data accesses are already cached and served from the cache (cf. Section 6). Meltdown-Power is influenced by the prefetcher loading the adjacent victim cache line, which we analyze in Section 7.6.

Cache Line Eviction. The i7-8700K we use for our analysis has an 8-way L1 cache and a 4-way L2 cache with pseudo-LRU replacement [1], allowing us to precisely determine the cache line to be evicted from the cache. For the 8-way L1 cache design, we use a setup with 16 distinct cache lines. Out of the 16 cache lines, we use one as the victim cache line filled with $V$. The remaining 15 cache lines represent attacker-controlled lines, each filled with the guess $G$. To determine the influence of accessing these 16 cache lines on the leakage model, we evaluate three eviction types with chosen L1 and L2 cache sets as shown in Figure 3, one without eviction, one with L1 eviction only, and one with L1 and L2 eviction: No Eviction: All cache lines are in different L1 sets; all 16 accesses are served from the L1 cache without self-eviction. L1 Eviction: All 16 cache lines are in one L1 but different L2 cache sets, resulting in constant L1 but no L2 cache eviction. L1+L2 Eviction: 8 cache lines are in one L1 and L2 cache set; the other 8 are in one L1 but unique L2 cache sets. Based on these experiments, we determine precisely how these cases influence the power leakage of the CPU.

Leakage Structure Analysis. We split the values $V$ and $G$ into consecutive slices and determine the impact on the sliced components $hd(V_i, G_j)$, $hd(V_i, V_j)$, $hd(G_i, G_j)$, $hw(G_i)$, and $hw(V_i)$ on the power consumption. We perform a linear regression with all the resulting variables and visualize the coefficients, i.e., a non-zero coefficient indicates if a slice is relevant for the power leakage function. Overall, we record 24 130 228 samples with the test code (cf. Listing 1) for the three eviction types. Figure 4 shows the results for 8 B slice sizes. This indicates that the Hamming distance and weight model the leakage components very well. Visually, we see three distinct effects: First, regardless of the eviction technique the first 8 B component $hd(V_0, G_0)$ shows a clear signal. Second, $hd(V_i, G_k)$, $k \in \{1 \to 7\}$ expose a leakage signal for the L1 and L1+L2 eviction case, albeit weaker than $hd(V_0, G_0)$. Finally, when considering L1+L2 eviction, an additional shifted signal components appear for $hd(V_k, G_h)$, $hd(V_k, V_h)$, $hd(G_k, G_h)$ with $k \in \{0 \to 7\}$ and $h = k + 4 \mod 8$. The period of this shift is 32 B, indicating that this effect could originate from a bus-size change from 64 B to 32 B, e.g., the interconnect between L2 and L3, meaning that the two halves of the cache line are transmitted after each other. Combining these observations that relate to the widths with which data is moved through the memory hierarchy, we can distinguish the influence of three zones within a cache line on the power...
leakage model: (1) the bytes accessed by the movq instruction, (2) the lower half of a cache line, and (3) the upper half of a cache line. Reducing the slice sizes down to single bits, i.e., all possible cases, confirms that the leakage still follows the same structure, confirming the suitability of the model.

The power leakage is influenced by the bytes accessed, the lower and upper cache line half, in distinct ways that can be modeled by Hamming distance and Hamming weight.

4.2 Modelling the Leakage Function

In this section, we derive and quantify a power leakage model that predicts the power consumption of loads, stores, and evictions from different points in the cache hierarchy, depending on the data. This model is the basis of our correlation power analysis attack and our end-to-end attacks (see Section 6). We extend the initial experiments with stores (movq) and prefetches (prefetchct0) to distinguish a load that fills a register from a prefetch that only brings data into the cache. To measure the effects of dirty cache lines, we clear the lowest 8 bytes of a cache line, marking it as dirty. Based on our previous insights into influence factors within a cache line, we optimize the regression analysis by operating with repeating nibbles (4-bit values), reducing the runtime by several orders of magnitude. Figure 5 shows the structure of two 64 B values: the attacker controlled guess \( G \) and the victim value \( V \). Each value is split into the lower \( (G_L, V_L) \) and upper \( (G_U, V_U) \) 32 B parts, resembling the discovered zones. Finally, we randomly sample four nibbles, i.e., \( v_L, v_U, s_L, \) and \( s_U \), to fill \( V = V_L \| V_U \) and \( G = G_L \| G_U \) respectively. We consider 4-bit aligned nibbles, resulting in 128 nibbles per value. Our experiments reveal that the power model consists of four distinct leakage components depending on the cache eviction strategy used.

The Full Leakage Model. Due to the different energy consumption of the instructions and the eviction strategies, we use the average power \( \bar{P} \) as our measurement, i.e., the energy over time. This measurement compares how substantial the leakage is and optimizes for the fastest leakage method. Furthermore, we determine the influence of the Hamming distance and the Hamming weights of \( G \) and \( V \) on the average power consumption. We perform a linear regression with the model shown in Equation \( 2 \) to get the exact scaling factors that model the \( \mu W \) changes due to single-bit changes. We choose a least-squares linear regression as it minimizes the error between the noisy measurements and the model. The model is visualized in Figure 6 and features the aligned leakage terms \( (a_i) \), i.e., the leakage between aligned bits in the cache line. Second, the model contains the cross leakage terms \( (c_i) \), i.e., the leakage across the lower and upper parts of different cache lines. Third, we also include the self leakage terms \( (s_i) \), i.e., the leakage between the upper and the lower parts of the same cache line. Finally, we add the Hamming weights \( (w_i) \) of each upper and lower cache-line part yielding the model

\[
\bar{P} = a_0 \cdot \text{hd}(V_L, G_L) + a_1 \cdot \text{hd}(V_U, G_U) + c_0 \cdot \text{hd}(V_L, G_U) + c_1 \cdot \text{hd}(V_U, G_L) + s_0 \cdot \text{hd}(G_L, G_U) + s_1 \cdot \text{hd}(G_U, G_L) + w_0 \cdot \text{hw}(V_L) + w_1 \cdot \text{hw}(V_U) + w_2 \cdot \text{hw}(G_L) + w_3 \cdot \text{hw}(G_U).
\]

4.3 Quantifying the Leakage Function

Table 2 shows the results of the regression analysis (cf. Section 4.2). The results indicate that eviction strategy and access instruction influence the power leakage model of the cache hierarchy. Furthermore, the correlation and model coefficients change based on the data placement. The results account for the repeating nibbles and show the coefficients for single nibbles, allowing us to quantify and compare the bit leakage in \( \mu W \). Furthermore, we introduce the aligned signal-to-noise ratio (SNR\(_A\)) between the aligned leakage and all other components, i.e., the noise, for simple comparison across techniques. This is not the overall signal-to-noise ratio of the model but rather the part important for our attacks.

Instructions and Data Placement. In line with our regression analysis (cf. Section 4.1), we observe that the position of the data and the instructions used to access the data influence the model coefficients. First, using no eviction never yields a significant signal for the upper cache line parts \( V_U \) and \( G_U \), as data is never moved between L1 and L2 cache. Furthermore, for prefetch with no eviction, we never observe any significant signal for any of the cache line parts, as no part of the cache line is moved into a register, and prefetch moves no data when it is already present in the L1 cache. For the store instruction, we overwrite the lower 8 B with zeros. We observe no signal with no eviction, as the dirty cache line never leaves the cache, and no actual victim or guess nibbles are stored. Therefore,
we conclude that explicitly loading data introduces observable leakage, e.g., 159.2 µW per bit difference for our test CPU.

The model coefficients are influenced by the instruction performing the access and from where to where data is transmitted in the CPU’s internal memory hierarchy.

### Cache Eviction

Model coefficients are influenced by different types of cache eviction, influencing the aligned signal-to-noise ratios of the techniques. The SNR_A for load instructions decreases by a factor of 2.413 when switching from no eviction to L1 eviction. We see a further decrease of factor 1.351 when switching from L1 to L1+L2 eviction, similar to the factor of 1.753 when using prefetch instructions. However, for store instructions, we see an increase of factor 1.712 in the aligned signal-to-noise ratio when performing L1+L2 eviction. We conclude that the mechanism to write back dirty cache lines add additional leakage (which we exploit in Section 7.5). Overall, we observe that for higher cache eviction activity, the correlation coefficients are increasing, indicating that the overall model represents the power consumption more accurately, but the desired signal is not increasing as strongly.

The model is more accurate with more cache eviction. However, the aligned signal-to-noise ratio decreases for the load and prefetch instructions, whereas for store instructions, it increases when using additional evictions.

### Memory Bus Effects

The results in Table 2 support the leakage patterns from our regression analysis (cf. Section 4.1). First, we observe that aligned leakage, i.e., coefficients $a_0$ and $a_1$, is present regardless of the eviction used. This is fundamental for our two attack variants MDS-Power and Meltdown-Power (cf. Section 6). When using L1+L2 eviction we observe additional effects: cross-leakage and self-leakage (cf. Figure 6). The self-leakage of the attacker-controlled guess in the case of the load instructions is a factor of 4.604 times stronger than the aligned leakage, 9.955 times for prefetches, and 11.302 times for stores, respectively. However, our novel differential measurement technique removes all the influences of self-leakage in the signal, as we demonstrate in Section 5.

Finally, we observe cross-leakage between the upper and lower parts of $\mathcal{V}$ and $G$. In all cases, the cross-leakage terms $c_0$ and $c_1$ are smaller than the aligned leakage terms. Therefore, the main influence in the CPA is still the aligned leakage. L1+L2 eviction adds additional leakage effects. The self-leakage of the attacker-controlled guess overshadows all other components. However, we address this with our differential measurement technique. The aligned leakage terms outweigh the cross-leakage terms.

We conclude that an attacker has several different ways to influence and model the leakage. By using memory accesses, cache loads and stores, and eviction, the attacker can precisely control and optimize the leakage rate for victim data in different locations in the CPU’s internal memory hierarchy.

### 5 Differential Measurement

As outlined in Section 3.1, increasing the signal-to-noise ratio is crucial to make Collide+Power practical. We propose a *differential* measurement technique to eliminate some noise influences, amplify the leakage, and reduce the required samples. We exploit that the power consumption can be influenced by two extremes: the maximum and minimum Hamming distance between attacker and victim value. However, the measurements are affected by additive noise $\omega$ that, based on our analysis, is near constant between temporally close samples.

#### Masking Victim Data

An attacker cannot influence the victim data value $\mathcal{V}$ in a real-world scenario. Therefore, we introduce the mask $m$, a 64 B value, indicating which bits of $G$ should be inverted; all remaining bits are unchanged. We measure one sample $\mathcal{P}_G$ for the guess $G$ and directly afterward the sample $\mathcal{P}_{\tilde{G}}$ with the inverse guess $\tilde{G} = G \oplus m$ and subtract the two samples $\Delta \mathcal{P}(G, \tilde{G}) = \mathcal{P}_G - \mathcal{P}_{\tilde{G}}$. The mask $m$ selects only a fraction of the cache line of $\mathcal{V}$ for the differential measurement, reducing the search complexity in a divide-and-conquer-style approach. The differential measurement changes the simplified leakage model of

$$\mathcal{P}(G, \mathcal{V}) = a_0 \cdot \text{hd}(G, \mathcal{V}) + a_1 \cdot \text{hd}(\mathcal{V}) + \text{hd}(G) + \omega,$$

(3)
by subtracting the model for $P(\mathcal{G}, \mathcal{V})$ to

$$
\Delta P = 2a_0 \cdot \text{hd}(G_m, V_m) + 2w_2 \cdot \text{hw}(G_m) - (a_0 + w_2) \cdot \text{hw}(m),
$$

where $G_m = \mathcal{G} \wedge m$ and $V_m = \mathcal{V} \wedge m$ are the masked cache lines with which we mask non-targeted data. The noise term $\phi$ and the unknown but constant value $\text{hw}(\mathcal{V})$ cancel out. Furthermore, subtracting $\text{hd}(\hat{G}, \mathcal{V})$ from $\text{hd}(G, \mathcal{V})$ yields a 2x amplification and a constant offset $\text{hw}(m)$. Finally, $\text{hw}(G) - \text{hw}(\hat{G})$ also results in a 2x amplification and another offset $\text{hw}(m)$. Thus, the final model amplifies the leakage by a factor of 2 and eliminates some additive noise. We show the full derivation of Equation 4 in Appendix A.

Quantifying the Differential Measurements. Table 3 shows the results with our differential measurement technique. We see that the derived differential model $\Delta P(G, \hat{G})$ in Equation 4 holds, and we gain a twofold amplification in $a_0$, $a_1$, $w_2$, and $w_3$ compared to Table 2. We can also see that $w_0$ and $w_1$, i.e., the influence of $\text{hw}(\mathcal{V})$, are reduced to 0. Similarly, using L1+L2 eviction completely removes the self-leakage effects $s_0$ and $s_1$. The correlation coefficients between our model and the actual measurements increase up to 0.907 in the case of loads with L1 eviction, i.e., a 23% increase, indicating that this significantly reduces the overall noise $\phi$ of the measurements. Finally, we also see that the aligned signal-to-noise ratio is increased by up to a factor of 11.27 (8.778 on average). Therefore, the measured results support our derivation of the differential model and its properties.

Our differential measurement model amplifies the signal by a factor of 2 and eliminates a significant part of the additive noise and self-leakage effects.

CPA Model Coefficients. In this section, we discuss how to minimize profiling for the CPA model used for Collide+Power (cf. Section 3). We derive the leakage structure and coefficients in Equation 2 and Table 3. As these exact coefficients require additional time to profile, we discuss three different approaches. First, we can profile the coefficients for the target CPU once and use a full model of Equation 2. An attacker could target its own cache lines to obtain these coefficients. Second, we try to approximate the coefficients without profiling the target. The correlation coefficient $\rho$ is scaling and location invariant, meaning that linear scaling $\cdot a$ and offsets $+b$ of the model do not influence the CPA attack. Due to the scaling invariance, we only need the ratio between some of the coefficients based on the used setup, e.g., $w_3/w_0$. If we consider Table 3, we observe that this ratio can be approximated based on the attack technique used. The ratio is approximately 0.7 for no eviction, 10.2 for L1 cache eviction, and 11.1 for L1+L2 cache eviction when using load instructions. The resulting model, then, is

$$
\mathcal{M}(G_m, V_m) = \text{hd}(G_m, V_m) + \frac{w_3}{w_0} \cdot \text{hw}(G_m).
$$

Finally, we fix the attacker-controlled parameter $\text{hw}(G_m)$ to a constant value simplifying the differential model to

$$
\mathcal{M}(G_m, V_m) = \text{hd}(G_m, V_m).
$$

Due to the mask $m$, and the differential measurement, we can partition the brute-force approach of recovering $\mathcal{V}$ into smaller problems by only recovering $V_m$. The attacker can choose an arbitrary mask $m$ and, according to Equation 4, only the selected bits will produce a measurable Hamming distance $\text{hd}(G_m, V_m)$ because the unmasked parts cancel out. We further verify that the unmasked portions do not influence the CPA success probability in Section 7.2.

6 End-to-End Attack Implementation

In this section, we describe the implementation of our two Collide+Power end-to-end attacks: MDS-Power and Meltdown-Power. While Meltdown-type attacks are mitigated in recent CPU generations, Collide+Power forms a drop-in replacement, achieving leakage rates of 4.82 bit/h in the MDS-Power case and 0.136 bit/h in the amplified Meltdown-Power case. The MDS-Power variant targets data-in-use, whereas the Meltdown-Power variant targets data-at-rest. Generally, Collide+Power is agnostic to the power side channel used. Depending on the system configuration and hardware, high-accuracy channels like Intel RAPL may be available to the at-
Listing 2: In the RIDL PoC, a victim program frequently accesses the victim cache line \( V \). Collide+Power extracts the accessed data without relying on any MDS vulnerabilities.

```c
while (true) {
    access(&victim_cache_line);
}
```

Listing 3: The artificial Spectre-RSB prefetch gadget in a kernel module used for the first Meltdown-Power evaluation.

```c
# rdx = &victim_cache_line
module_ioctl:
    # rdx = &victim_cache_line
    call retпольпle($rip)
    misпpeculation:
        mov (%rdx), %rax
    ud2
    retпolпe:
        lea retpoline_end($rip), %rax
    mov %rax, (%rip)
    ret
retpoline_end:
    xor %eax, %eax
    ret
```

6.1 MDS-Power Implementation

MDS-Power follows the exact scenario and threat model of MDS attacks like RIDL and ZombieLoad [34, 37] (cf. Section 3.3). Listing 2 shows the RIDL PoC® where the victim program accesses a cache line in a loop while being co-located with the attacker on the same physical core. Although a real-world victim program is unlikely to perform secret accesses in a loop, it offers a fair comparison of MDS-Power with RIDL and ZombieLoad as they use similar victim programs. Intel fixed this MDS hardware flaw in the 9th CPU generation. With MDS-Power, we demonstrate MDS-style leakage without relying on any hardware MDS vulnerability.

The basic setup of MDS-Power is illustrated in Figure 7. MDS-Power exploits that due to the victim’s own load, the victim’s secret value constantly moves through the CPU’s memory hierarchy, e.g., in and out of internal buffers. The attacker simultaneously repeatedly loads guesses \( G \), which then move through the same parts of the CPU’s memory hierarchy. MDS-Power then instantiates Collide+Power either with direct (e.g., the Intel RAPL interface if available) or indirect (e.g., via timing differences due to throttling) power side-channel leakage. MDS-Power is evaluated in Section 7.3.

6.2 Meltdown-Power Implementation

Meltdown-Power follows the scenario and threat model of the original Meltdown attack [20], leaking arbitrary kernel data from userspace (cf. Section 3.3). Like Meltdown on non-L1 data, Meltdown-Power depends on the victim code to load the data, possibly triggered by the attacker. We use a Spectre prefetch gadget inside the kernel [35]. They are more widespread [4, 13] than regular Spectre gadgets, as they only load data but do not leak it. The attacker then alternatingly primes the cache with 15 distinct cache lines filled with the guess \( G \) (cf. Section 4) and then reloads the victim cache line \( V \) using a prefetch gadget, as shown in Figure 8. We evaluate Meltdown-Power with two prefetch gadgets, an artificial one for an in-depth evaluation and a real-world gadget to show the practicality of Meltdown-Power.

### Artificial Spectre-RSB Prefetch Gadget

The artificial Spectre-RSB gadget in Listing 3 is used in our first Meltdown-Power evaluation. The gadget is similar to the RSB gadget shown in Section 4. The gadget first calls a function (Line 3), which in Lines 8 and 9, modifies the return address on the stack to point to the ioctl entry function of a kernel module. During misspeculation, it dereferences rdx containing an attacker-controlled pointer to the victim cache line \( V \). Similar to the retпolпe mitigation [36], the gadget first calls a function (Line 3), which in Lines 8 and 9, modifies the return address on the stack to point to the ioctl entry function. Then, the CPU misпpeculates the return address and transiently accesses the victim cache line \( V \) (Line 5). This \( V \) into the cache, exposing it to Collide+Power. Identical to the prefetch gadget exploited by Meltdown [35], this gadget can fetch any kernel-accessible memory location into the cache, including all physical memory, via the direct-physical map.

### Real-World Spectre-PHT Prefetch Gadget

To demonstrate the practicality of Meltdown-Power, we also evaluate Meltdown-Power with a real-world Spectre prefetch gadget in

---

2RIDL PoC source code: https://github.com/vusec/ridl/blob/be7e2bd16df5a1ec78c4bf9b82912c230971d2/c/pocs/ridl_basic.c
7 Evaluation

In this section, we evaluate Collide+Power on multiple CPUs to demonstrate that cache-hierarchy leakage is a widespread problem. Furthermore, in an end-to-end scenario, we demonstrate MDS-Power leaking data from the sibling thread with both the RAPL interface and throttling side channels. Finally, we evaluate Meltdown-Power in one artificial setting to demonstrate that we can extract single bits from the kernel and measure an amplified real-world prefetch gadget to verify that we observe a signal usable for Collide+Power.

7.1 Affected CPUs

We systematically analyze on which CPUs the differential model Equation 4 of Collide+Power works. We perform the same experiments as in Section 5 and report the maximum observed correlation coefficient and the maximum SNR, i.e., the significance of the aligned Hamming distance leakage, which is the foundation of Collide+Power. Table 4 shows the result on 14 CPUs from both AMD and Intel, spanning a release period from 2011 until 2021, for nearly all of which we can demonstrate to be affected by Collide+Power. We observe that 12 out of 14 CPUs have a maximum correlation coefficient above 0.1, reaching 0.907 on the Intel Core i7-8700K (cf. Table 3). Furthermore, we found a maximum signal-to-noise ratio of 79.132 \times 10^{-3} on the Intel Core i9-9980HK, which is 9.9% stronger than the Intel Core i7-8700K. For the CPUs with lower metrics, we cannot exclude that a slightly different cache eviction (cf. Figure 3) could increase leakage. Finally, we conclude that Collide+Power is widespread due to how we build and design CPU memory hierarchies.

7.2 Evaluation of the Model for the Channel

Our evaluation of the CPA models using our differential measurement shows the general capabilities of Collide+Power. We focus on load instructions with L1 cache eviction in two settings. First, we fill the complete values \( V \) and \( G \) with the repeating nibbles \( v \) and \( g \), respectively (128x). Second, we use a single nibble \( v \) and \( g \) and zero the remaining parts, representing no-amplification (1x). To compute the CPA success probability (cf. Figure 9), we repeatedly take a specific number of random samples from our measured samples, perform the CPA, and compute how often \( v \) was recovered without any bit errors. The number of samples used determines the leakage rate. In the amplified case, we can recover 505.81 bit/h of \( v \) with a success probability of 99.0% (\( n=1000, \sigma_5=0.31\% \)). Without amplification, we still achieve a leakage rate of 10.99 bit/h with 99.8% (\( n=1000, \sigma_5=0.14\% \)), corresponding to 23 000 differential measurements per nibble.

The differential measurement technique allows the masking of specific data within a cache line. To verify that the differen-
We perform an experiment that targets the unamplified (1x) victim nibble $v$ in three distinct scenarios: We fill the remaining parts of the cache line with either zero, random data that stays constant, or random data that changes between measurements and compute the influences on the CPA success probability. We use movq loads with the no eviction pattern (cf. Figure 3) for this analysis and show in Figure 10 that the unmasked victim data does not influence the CPA success probability. This matches the derivation of Equation 4 where the unmasked terms cancel out. We discuss changing the masked data as mitigation against Collide+Power in Section 8.

In Figure 11, we compare coefficients of the CPA model (cf. Section 5) based on the amplified (128x) L1 eviction experiment from the first paragraph. First, when only using samples with fixed Hamming weight ($hw(G) = 2$), we achieve a 2.5 times higher leakage rate than the exact model to achieve a success rate of 99.8\% ($n=1000, \sigma_x=0.14\%$). Second, fixing the ratio $\frac{w_2}{a_0}$ to 10.2 does not result in an observable difference to the exact model coefficients. Finally, using the incorrect ratio $\frac{w_2}{a_0}$ of 3.9 requires reducing the leakage rate 20 times to reach a success rate of 99.3\% ($n=1000, \sigma_x=0.26\%$). Therefore, we conclude that small inaccuracies in the model coefficients can be compensated with more samples.

### 7.3 Evaluation of MDS-Power

We evaluate MDS-Power on an Intel Core i9-9900K CPU with Ubuntu 20.04.5 LTS and Linux kernel 5.4. To compare the leakage of the different approaches, we report the leakage rate in bits per hour with the CPA success probability of recovering the victim nibble $v$ without errors. We disabled the hardware prefetchers during this experiment. However, as we discuss in Section 4.1, the best attack scenario is unaffected by hardware prefetchers. Figure 12a compares the leakage for different eviction strategies when using load instructions to access the guess cache lines, measured with RAPL. We observe that the no eviction technique achieves a leakage rate of 188.80 bit/h and recovers the amplified nibble (128x) with 99.2\% ($n=1000, \sigma_x=0.28\%$) success rate. No eviction achieves 9.33 times the leakage rate of the L1 cache eviction and 18.07 times the leakage rate of the L1+L2 cache eviction. When targeting a single unamplified nibble (1x), no eviction achieves a leakage rate of 4.82 bit/h with a success rate of 98.9\% ($n=1000, \sigma_x=0.32\%$).

Figure 12b evaluates the no-eviction technique with throttling attacks [21, 38]. With adaptive power management [11], the CPU regulates its frequency and voltage to meet its power targets. However, the time-stamp counter frequency, read by rdtscp, is fixed, and therefore, throttling-based attacks can be mounted with this simple primitive. We evaluate the two distinct methods to achieve frequency throttling, as described in Section 3.3. First, we set the power limit of the CPU to 5.625 W over 0.977 ms in the MSR_PKG_POWER_LIMIT [12]. We observe that when using the MSR to set the power limits, the no-eviction technique achieves a leakage rate of

---

**Table 4: Evaluation of the leakage model for Collide+Power on different CPUs and microarchitectures.**

<table>
<thead>
<tr>
<th>CPU</th>
<th>Microarchitecture</th>
<th>Microcode</th>
<th>Stepping</th>
<th>Release Year</th>
<th>L1 Ways</th>
<th>L2 Ways</th>
<th>$\hat{\rho}$</th>
<th>SNR$_4 \cdot 10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core i5-2520M</td>
<td>Sandy Bridge</td>
<td>0xb1</td>
<td>7</td>
<td>2011</td>
<td>8</td>
<td>8</td>
<td>0.011</td>
<td>0.107</td>
</tr>
<tr>
<td>Core i3-7100T</td>
<td>Kaby Lake</td>
<td>0xc0</td>
<td>9</td>
<td>2017</td>
<td>8</td>
<td>4</td>
<td>0.024</td>
<td>0.416</td>
</tr>
<tr>
<td>Xeon E-2176M</td>
<td>Coffee Lake</td>
<td>0xf0</td>
<td>10</td>
<td>2018</td>
<td>8</td>
<td>4</td>
<td>0.820</td>
<td>17.997</td>
</tr>
<tr>
<td>Core i7-10510U</td>
<td>Comet Lake</td>
<td>0xc6</td>
<td>12</td>
<td>2019</td>
<td>8</td>
<td>4</td>
<td>0.549</td>
<td>7.188</td>
</tr>
<tr>
<td>Core i7-10710U</td>
<td>Comet Lake</td>
<td>0xe0</td>
<td>0</td>
<td>2019</td>
<td>8</td>
<td>4</td>
<td>0.130</td>
<td>0.008</td>
</tr>
<tr>
<td>Core i7-1185G7</td>
<td>Tiger Lake</td>
<td>0xb2</td>
<td>1</td>
<td>2020</td>
<td>12</td>
<td>20</td>
<td>0.728</td>
<td>7.643</td>
</tr>
<tr>
<td>Ryzen 5 2500U</td>
<td>Zen</td>
<td>0xb100b</td>
<td>0</td>
<td>2017</td>
<td>8</td>
<td>8</td>
<td>0.428</td>
<td>5.429</td>
</tr>
<tr>
<td>Ryzen 5 3550H</td>
<td>Zen+</td>
<td>0xb1012</td>
<td>1</td>
<td>2019</td>
<td>8</td>
<td>8</td>
<td>0.585</td>
<td>3.025</td>
</tr>
<tr>
<td>EPYC 7252</td>
<td>Rome</td>
<td>0xa5000c</td>
<td>0</td>
<td>2019</td>
<td>8</td>
<td>8</td>
<td>0.160</td>
<td>0.178</td>
</tr>
<tr>
<td>Ryzen 9 5900HX</td>
<td>Zen 3</td>
<td>0xb3055</td>
<td>0</td>
<td>2021</td>
<td>8</td>
<td>8</td>
<td>0.650</td>
<td>7.269</td>
</tr>
</tbody>
</table>
We verify that the leakage models derived in Section 5 hold when exchanging the average power measurements with timing measurements. Table 6 shows the correlation coefficients, the aligned signal-to-noise ratio, and the model coefficients for the RAPL interface, throttling attacks via the energy limit MSR, and throttling via stress (cf. Section 3.3). We denote that the unit changes from Watt to Seconds due to the different measurements. We reuse the recorded data of the MDS-Power experiment on our Core i9-9900 CPU (cf. Section 7.3). In line with the differential leakage model, we observe that the hw(\mathcal{V}) component is minimal compared to the other components due to the differential measurements. Furthermore, we see a decrease in the correlation coefficient of factor 2 when switching from RAPL to the power limit MSR and a decrease of 13.034 when switching from RAPL to stress. We conclude that although the signal-to-noise ratio and the correlation coefficients are decreasing, we still observe a measurable signal usable for MDS-Power as shown in Section 7.3.

### 7.5 Evaluation of Meltdown-Power

Meltdown-Power has a significantly lower performance than MDS-Power due to the amount of code executed for the prefetch gadget and the reliability of its speculation. Generally, this activity drastically increases the time to obtain a single measurement sample and reduces the signal-to-noise ratio. Therefore, to enable a fair and robust comparison to MDS-Power, we evaluate Meltdown-Power primarily with the RAPL interface. Furthermore, we disable the hardware prefetchers during this experiment. We discuss and evaluate the resulting implications in Section 7.6. We estimate the number of samples required to observe the same leakage with throttling side channels (cf. Section 7.4), based on our RAPL measurements, taking the evaluation of MDS-Power into account (see Section 7.3). Some Meltdown mitigations switch the cr3 register during context switches [20], which implicitly flushes the TLB. When the kernel flushes the TLB upon entry, the real-world prefetch gadget found by Johannesmeyer et al. [13] has a low prefetch rate on our test machine. As Meltdown-Power is particularly relevant on systems not affected by Meltdown, we assume that such mitigations are not
We identify that Meltdown-Power is influenced by the
in place, and the TLB is not flushed after entering the kernel,
which is commonly the case on newer microarchitectures.

We evaluate Meltdown-Power with Ubuntu 20.04 LTS on
an Intel Core i7-8700K (CPU A), an Intel Core i9-9980HK
(CPU B), and an Intel Core i7-6700K (CPU C). CPU A and
C run Linux kernel version 5.4, and CPU B version 5.13. We
find that L1+L2 eviction with dirty cache line works best
for Meltdown-Power and did not observe a signal with no-
eviction, in line with our assumptions as the guess $G$
will not reach the value $V$. We disabled the hardware prefetchers.
However, in this scenario, we show in Section 7.6 that the influences
of the hardware prefetchers are minimal.

First, we evaluate the artificial Spectre-RSB gadget from
Listing 3 located within a custom kernel module in Figure 13a.
In the amplified nibble (128x) case, we achieve a leakage
rate of 12.47 bit/h with 99.9 % ($n=1000$, $\sigma_z=0.10 \%$) success
probability on CPU B which is 2.843 times higher than the
leakage rate of CPU A with 99.2 % ($n=1000$, $\sigma_z=0.28 \%$) suc-
cess probability. Furthermore, in an unamplified scenario (1x),
we achieve a leakage rate of 0.84 bit/h with a success probabil-
ity of 99.7 % ($n=1000$, $\sigma_z=0.17 \%$) on CPU B.

Second, we evaluate Meltdown-Power with the real-world
Spectre-PHT gadget. The real-world Spectre-PHT gadget
loads the cache line with a 90.51 % ($n=10000$, $\sigma_z=0.03 \%$)
8 Mitigations and Limitations

In this section, we discuss potential mitigations and limitations of Collide+Power. A principled mitigation against Collide+Power should eliminate or reduce the root cause of the leakage. We discuss mitigation ideas on the hardware and software level, although entirely preventing power-based leakage is still an open problem for general-purpose CPUs.

Hardware Mitigations. Power analysis attacks have been studied for decades on smaller form factor devices, such as smart cards [2, 17, 24]. Hence, hardware-based mitigations were researched and deployed in the wild for these systems. Such mitigations include blinding, masking, or adding randomness [24]. However, these mitigations require a fundamental hardware redesign and are usually tailored to protect cryptographic algorithms. Furthermore, these mitigations often require additional hardware for partitioning the computation into multiple parts, adding a significant performance impact [32]. Therefore, while it is theoretically possible with newer CPUs, such effective but costly mitigations are unlikely to be added to consumer CPUs.

Software and Operating System Mitigations. Meltdown-Power shown in Section 6.2 relies on a prefetch gadget in the kernel. Hence, a potential mitigation is eliminating all prefetch gadgets in the kernel. However, orthogonal research from Johannesmeyer et al. [13] on gadget finding suggests that a plethora of such gadgets exist in the kernel. Exacerbating the problem further: The prefetch gadgets required for Collide+Power are simpler than traditional transient-execution gadgets as they do not require the encoding part, e.g., a secret-dependent gadget. While Collide+Power requires co-location with the victim on a hyperthread, which could be prohibited if a group scheduling policy is implemented. However, the effects described in this paper likely apply to additional shared buffers in the CPU. Following the suggestions of Wang et al. [38], Turbo Boost and SpeedStep on Intel or Cool’n’Quiet on AMD CPUs can be disabled to curb userspace attacks, causing the CPU to reach the power limit less frequently. However, the question arises of which other power-related signals an attacker could use instead of the throttling side channels. Finally, in Section 7.2, we show that changing data co-located in the victim cache line does not impact the attacker’s success probability. However, dynamically changing the victim values, e.g., cryptographic keys, breaks the assumptions of Collide+Power that the victim data is constant during the attack, effectively preventing leakage due to the relatively low leakage rates. Nevertheless, in contrast to traditional rekeying, the changing interval must depend on wall-clock time, not usage count, as unused secrets could be reachable with Collide+Power. Another alternative mitigation for MDS-Power is to sandwich secret data loads between victim-controlled loads, preventing the collisions of the attacker-controlled guesses and the victim value. However, this mitigation is ineffective against Meltdown-Power.

Limitations. While Collide+Power exploits the energy differences induced by cache loads, our primitives are not limited to the cache. In theory, the contents of any microarchitectural element with data-dependent energy consumption can be leaked. In practice, we require that the energy consumption becomes observable via performance counters for a privileged attacker or with frequency scaling in an unprivileged scenario. Future work could explore the impacts of Collide+Power on other mi-
Croarchitectural buffers. The current Meltdown-Power proof-of-concept has severe practical limitations reflected in the low security risk when using the Spectre-PHT prefetch gadget. Therefore, Collide+Power benefits from research identifying optimized prefetch gadgets.

9 Conclusion

Collide+Power shows that mere co-location of data values in microarchitectures introduces combined leakage in the power domain. Our systematic analysis of the CPU’s memory hierarchy led to precise leakage models that enable the exploitation of this combined leakage. We demonstrated that Collide+Power works with power consumption interfaces or throttling-induced timing variations alike. Our novel differential measurement technique amplifies the signal-to-noise ratio by a factor of 8.778 on average, compared to a straightforward DPA approach. We demonstrated that Collide+Power can even leak single-bit differences from the CPU’s memory hierarchy with fewer than 23,000 measurements. In MDS-style end-to-end attacks, Collide+Power leaks 4.82 bit/h in the same scenario as RIDL and ZombieLoad but without relying on the MDS hardware flaw. However, in real-world Meltdown-style attacks, we encounter practical limitations leading to leakage rates of more than a year per bit with throttling. Future work is required to find more practical prefetching methods to replace the current Spectre-PHT gadget and to reevaluate the potential security risk of Meltdown-Power. Since Collide+Power is a generic attack with different variants, applying to any modern CPU, it poses a significant challenge for future work to develop mitigations against this threat. For commodity systems, mitigating Collide+Power is more challenging, as it exploits the very basics of microarchitecture design.

Acknowledgments

We thank the anonymous reviewers, especially our shepherd, for their guidance, comments, and suggestions. We thank Robert Primas, Daniel Weber, and Moritz Lipp for engaging discussions. Furthermore, we thank Brian Johannesmeyer and Jakob Koschel for their help and expertise with the Spectre-PHT prefetch gadget. This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe F8504), and the Semiconductor Research Corporation (SRC) Hardware Security Program (HWS). A generous gift from AWS, Red Hat, and Google provided additional funding. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References


We add an adaptive noise term $\omega$ to model a realistic measurement. We model the power consumption of two distinct attacker-controlled guesses, the normal guess $G$ and its masked inverse $\tilde{G} = G \oplus m$. We assume that the noise between two successive samples is constant due to time locality. The victim data $V$ stays constant during both of these guesses. This yields the two equations

$$P(G, V) = a_0 \cdot \text{hd}(G, V) + w_0 \cdot \text{hw}(V) + w_2 \cdot \text{hw}(G) + \omega$$

and

$$P(\tilde{G}, V) = a_0 \cdot \text{hd}(\tilde{G}, V) + w_2 \cdot \text{hw}(\tilde{G}) + w_0 \cdot \text{hw}(V) + \omega.$$  

Subtraction of the equations $P_G$ and $P_{\tilde{G}}$ yields the power difference between both of the attacker’s guesses,

$$P_G - P_{\tilde{G}} = a_0 \cdot (\text{hd}(G, V) - \text{hd}(\tilde{G}, V)) + w_0 \cdot \text{hw}(V) + \omega + w_2 \cdot (\text{hw}(G) - \text{hw}(\tilde{G})) - \omega.$$  

First, we derive the results of the subtraction of the Hamming difference $\text{hd}(G, V)$ with the Hamming difference where one parameter is the mask inverse $\text{hd}(\tilde{G}, V)$. We consider the following equation $\text{hd}(x, y) - \text{hd}(-x, y)$. To change a value from $x$ to $y$, $\text{hd}(x, y)$ bits need to be flipped. Similarly, to change from $x$ to $-x$, the number of bits in a word $N$ need to change. Therefore, we can flip from $-x$ to $x$ and undo the flips we require to get to $y$, yielding $\text{hd}(-x, y) = N - \text{hd}(x, y)$. Therefore,

$$\text{hd}(x, y) - \text{hd}(-x, y) = 2 \cdot \text{hd}(x, y) - N.$$  

If we consider $\text{hd}(G, V) - \text{hd}(G \oplus m, V)$, we observe that only the bits selected by $m$ are active as the other bit differences cancel out due to the Hamming distance. Therefore, the word size is reduced to $N = \text{hw}(m)$, and we derive

$$\text{hd}(G, V) - \text{hd}(G \oplus m, V) = 2 \cdot \text{hd}(G_m, V_m) - \text{hw}(m).$$  

where $G_m = G \land m$ and $V_m = V \land m$ are the masked cache lines with which we mask away non-targeted data. Second, we derive the results of the subtraction of the Hamming weight $\text{hw}(G)$ with the Hamming weight of the mask inverse $\text{hw}(\tilde{G})$. We use the following property $\text{hw}(-x) = N - \text{hw}(x)$ to derive

$$\text{hw}(x) - \text{hw}(-x) = 2 \cdot \text{hw}(x) - N.$$  

With the same reasoning about the active bits, we derive

$$\text{hw}(G) - \text{hw}(G \oplus m) = 2 \cdot \text{hw}(G_m) - \text{hw}(m).$$  

Resulting in the final differential power leakage model of Equation 4,

$$P_{G} - P_{\tilde{G}} = a_0 \cdot (2 \cdot \text{hd}(G_m, V_m) - \text{hw}(m)) + w_2 \cdot (2 \cdot \text{hw}(G_m) - \text{hw}(m)).$$  

### B Kernel Spectre-PHT Prefetch Gadget

The prefetch gadget in Listing 4 was discovered by Johannesmeyer et al. [13]. First, Line 3 iterates over a list of keyrings $\text{ns} \rightarrow \text{keyring name list}$ of the running process’s namespace. During this iteration, the CPU’s branch predictor is mistrained and speculatively accesses one additional element at the end of the loop. Due to the given memory layout [5, 6, 13, 18], during speculative execution, a type confusion occurs, where a $\text{struct user namespace}$ is interpreted as a $\text{struct key}$. Therefore, the speculative access of $\text{keyring} \rightarrow \text{user} \rightarrow \text{uid}$ in Line 5 actually accesses $\text{ns} \rightarrow \text{projid map} \rightarrow \text{entry}[3]$, which is controllable by an attacker, resulting in the desired prefetch gadget [13]. The array $\text{ns} \rightarrow \text{projid map} \rightarrow \text{entry}[3]$ can be filled by the attacker by writing to $/proc/self/projid_map$.  

Listing 4: The Spectre-PHT prefetch gadget of the Linux kernel key management used to load arbitrary data.

```c
struct key *find_keyring_by_name(const char *name,  
  -> bool uid_keyring) {  
  if(!uid_has_mapping(ns, keyring->user->uid))  
    continue;  
  if(test_bit(KEY_FLAG_REVOKED, &keyring->flags))  
    continue;  
  if(strcmp(keyring->description, name) != 0)  
    continue;  
  // ...  
}  
```