

# “There’s always another counter”: Detecting Micro-architectural Attacks in a Probabilistically Interleaved Malicious/Benign Setting

Upasana Mandal , Rupali Kalundia , Nimish Mishra , Shubhi Shukla ,  
Sarani Bhattacharya , and Debdeep Mukhopadhyay 

Indian Institute of Technology Kharagpur, India  
{mandal.up98, rupalikalundia, nimish.mishra,  
shubhishukla}@kgpian.iitkgp.ac.in, {sarani, debdeep}@cse.iitkgp.ac.in

**Abstract.** Modern micro-architectural attacks use a variety of building blocks chained to develop a final exploit. However, since in most cases, the footprint of such attacks is not visible architecturally (like, in the file-system), it becomes trickier to defend against these. In light of this, several automated defence mechanisms use Hardware Performance Counters (HPCs) detect when the micro-architectural elements are being misused for a potential attacks (like flush-reload, Spectre, Meltdown etc.). In order to bypass such defences, recent works have proposed the idea of “probabilistic interleaving”: the adversary interleaves the actual attack code with benign code with very low frequency. Such a strategy tips off the HPCs used for detection with a lot of unnecessary noise; recent studies have shown that probabilistically interleaved attacks can achieve an attack evasion rate of 100% (i.e. are virtually undetectable).

In this work, we contend this folklore. We develop a theoretical model of interleaved attacks using lightweight statistical tools like Gaussian Mixture Models and Dip Test for Unimodality and prove they are detectable for the correct choices of HPCs. Furthermore, we also show possible defence strategy against a *stronger* threat model than considered in literature: where the attacker interleaves *multiple* attacks instead of a single attack. Empirically, to instantiate our detector, in contrast to prior detection strategies, we choose LLMs for a number of reasons: (1) LLMs can easily contextualize data from a larger set of HPCs than generic machine learning techniques, and (2) with simple prompts, LLMs can quickly switch between different statistical analysis methods. To this end, we develop an LLM-based methodology to detect probabilistically interleaved attacks. Our experiments establish that our improved methodology is able to achieve 100% speculative attacks like Spectre v1/v2/v3, Meltdown, and Spectre v2 (with improved gadgets that even evade recent protections like Enhanced IBRS, IBPB conditional, and so on). This makes our methodology suitable for detecting speculative attacks in a non-profiled setting: where attack signatures might not be known in advance. All in all, we achieve a 100% attack detection rate, even with very low interleave frequencies (i.e.  $10^{-6}$ ).

Our detection principle and its instantiation through LLMs shows how probabilistically interleaving attack code in benign execution is not a

perfect strategy, and more research is still needed into developing and countering better attack evasion strategies.

**Keywords:** LLMs · Micro-architectural attacks · Interleaved attacks

## 1 Introduction

Research into malware hiding strategies [6] is mostly focused upon better ways of evading the ever-evolving defence mechanisms. Such attacks/defences can be thought of as *architectural* attacks: they leave traces observable at the architectural level (for instance, the file-system). However, *micro-architectural* attacks [3,7,9,10,12,14,15,17,19,20,21,16,18,5] are trickier to detect since they do not leave any observable architectural trace. Micro-architectural attacks are capable of leaking sensitive cryptographic materials, breaking software/hardware isolation, and using bad speculation (pipeline flushes) to drastic ends. Such attacks target aspects of the micro-architecture (like cache, execution ports, on-core buffers, and so on), and thus require different methods of detection than traditional malware.

**Strategies of detection.** All modern vendors like Intel/AMD ship their processors with documented/undocumented counters (otherwise technically named Hardware Performance Counters or HPCs) that measure different aspects of the hardware. Since micro-architectural attacks exploit aspects of the hardware, a long line of research [1,2,4,8,13] into detecting micro-architectural attacks has focused on using HPCs. The core idea is to monitor HPCs and look for statistical variations that might flag adversarial execution from benign. Such defences use a variety of tests: template-based matching, anomaly detection, machine learning, and so on, as their detection strategy.

**HPC design and Issues with Detection.** Hardware Performance Counters, by design, are tied to the specific hardware unit they target. For a concrete example, on Intel systems, `INST_RETIRED.ANY` counts the number of x86 instructions retired (and is thus tuned to the instruction retirement unit in the backend execution unit <sup>1</sup>. Likewise, `BR_INST_RETIRED.ALL_BRANCHES` counts the number of retired branch instructions (and is thus attuned to the branch predictor unit in the frontend execution unit) <sup>2</sup>. As such, HPCs do not conform to software isolation boundaries maintained in userspace. In other words, HPCs keep counting irrespective of the software executing atop the processor. This means that attack detection strategies that use HPCs are vulnerable to noise due to context switches from kernel scheduling, hyperthreading, and other events in the hardware. HPC-based detection strategies [1,2,4,8,13] then leverage statistical methods to factor out such noise.

**Emergence of “Interleaved” Attacks:** Exploiting this software-agnostic nature of HPC design, the authors in [13] have proposed a new micro-architectural

<sup>1</sup> <https://perfmon-events.intel.com/>

<sup>2</sup> Similar HPC counterparts exist for other vendors as well, like AMD.

attack strategy: *interleave attack code, with low frequency, among executions of benign code*. On a high-level, such an interleaved attack strategy executes the benign code for multiple iterations, and *interleaves* the attack in some iterations (driven by a small probability threshold). Since the probability threshold of interleaving is small (like 1 in million), this strategy tips off all existing defence mechanisms [1,2,4,8,13], which mainly capture characteristics of the predominant benign execution and miss characteristics of the otherwise sporadic attacker execution. This leads us to the following question, which we deal in this work:

*Can we develop a lightweight statistical analysis of interleaved attacks that still allows detection through HPCs, which have no detection strategy in literature?*

### 1.1 Our Contributions

We answer this question in the affirmative. This work contributes to a general understanding of the statistical behaviour of interleaved attacks, as well as a generic detection strategy. We also contribute a detection strategy against an even *stronger variant* of the threat model considered by interleaved attacks. We summarize our contributions thus:

- ① We propose a theoretical modeling of interleaved attacks using lightweight statistical tools like Gaussian Mixture models and Dip Test for Unimodality. We demonstrate that for carefully chosen measurements (i.e. HPCs), interleaved attacks are still detectable.
- ② We demonstrate the detection of interleaved attacks claimed to be undetectable by the prior art. In particular, we test against Spectre v1, Spectre v3, and Meltdown. Furthermore, we also test for Spectre v2, in both the traditional setting as well as through advanced gadgets which evade detection even by state-of-the-art Spectre v2 mitigations like Enhanced IBRS, IBPB conditioning and so on.
- ③ We analyze the HPC data collected during the execution of an interleaved attack using a large language model (LLM) to perform the Dip test. The LLM allows easy user-friendly detection by identifying multimodal distributions through the Dip Test for Unimodality, which helps reveal potential interleaved attacks like Spectre v1 and other similar threats.
- ④ We also consider a *stronger* threat model of interleaved attacks: instead of a single attack, several different attacks are interleaved with benign execution. We extend our model trivially to Multimodal Gaussian mixtures and show that multiple attacks interleaved with benign execution are still detectable by HPCs.

## 2 Background

### 2.1 Interleaved Attacks

The idea of interleaved attacks was first presented in [13]. The main objective of the work is to systematically evaluate the effectiveness of HPC-based detection mechanisms against cache-based side-channel attacks. This analysis is done across four dimensions: ① threat model, ② performance overhead, ③ detection speed, and ④ detection reproducibility. About 50 state-of-the-art defence mechanisms are evaluated against these dimensions, and relevant comparisons are drawn between them. Through this systematization, the authors consolidate a commonality between different classes of defences wrt. the assumptions they place on the adversary. Concretely, all defences expect the attacks to occur in isolation for which HPC traces (with minor operational noise) can be collected.

From this understanding, the authors put forward a new threat model where attacks are *interleaved* with benign executions with low frequency. As a concrete example, consider the case of the Flush+Reload covert channel, which forms an exfiltration channel for several other attacks. In a traditional Flush+Reload attack, the adversary flushes a shared memory line from the cache, waits for the victim to access the cache line, and then measures the time taken to reload the data. This timing reveals whether the victim accessed the flushed cache line. However, in [13] this Flush+Reload exfiltration activity is *interleaved* among several iterations (like 10000 or 1000000) of the benign function. This low-frequency execution reduces the footprint of the attack in the HPC data, making it appear as though the system was executing typical benign tasks. In some sense, interleaved attacks achieve *temporal blending* of the attack with benign execution: by strategically placing the malicious execution alongside frequent benign execution, the attack produces no clear statistical deviation in data from HPCs such as cache hits, cache misses, or branch mispredictions. A basic outline of such an attack is given in Listing 1.1. The key idea is that by running the attack code at a low frequency and interspersing it with frequent benign tasks, the resulting behavior looks just like normal, harmless execution. In contrast, traditional methods like Flush+Reload create noticeable patterns in cache activity that can be detected. However, when an attack is interleaved with normal operations, this distinctive cache activity gets mixed with normal cache activity, making it much harder for detection systems that rely on HPC data to identify the attack.

Briefly stating, existing detection mechanisms rely purely on large-scale statistical anomalies in HPC data, and thus are unable to detect interleaved attacks. This is because such systems operate under the assumption that malicious activity will always manifest as an obvious anomaly, but when the attack is subtle and well-hidden, such as by blending with normal behavior, these systems fail to perform effectively.

```

1 void interleaved() {
2     benign();
3     // this threshold is very small
4     if(sample_probability() < threshold)
5         attack();
6     benign();
7 }

```

Listing 1.1: A skeleton of interleaved attack from [13].

## 2.2 Dip Test of Unimodality

The Dip Test of Unimodality [11] is a statistical test that quantifies multimodality in a sample by measuring the statistical distance between the “actual” distribution and the expected unimodal distribution. Concretely, the “actual” distribution of a set of data points  $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ , formally called the *Empirical Distribution Function* (EDF) or  $\mathcal{F}_n(\mathbf{x})$ , is given as:

$$\mathcal{F}_n(x_j) = \frac{1}{n} \sum_{i=1}^n \mathcal{I}(x_i) \quad : \quad j \in \{x_1, x_2, x_3, \dots, x_n\}$$

Where  $\mathcal{I}$  is an indicator function that returns 1 iff  $x_i \leq x_j$  and 0 otherwise. Informally,  $\mathcal{F}_n(\dots)$  is a step function that adds  $\frac{1}{n}$  mass for every point in  $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ .

Likewise, the unimodal distribution  $\mathcal{F}_U$  is given such that for its domain  $\mathcal{D}$ ,

$$\exists x_m \in \mathcal{D} : \quad \frac{d}{dx}(\mathcal{F}_U(x)) \geq 0 \text{ for } x \leq x_m \quad \text{and} \quad \frac{d}{dx}(\mathcal{F}_U(x)) \leq 0 \text{ for } x \geq x_m$$

In other words, there exists a *mode* (i.e.  $x_m$ ) in the input domain of the function such that the function is *increasing* for all  $x \leq x_m$  and *decreasing* for all  $x \geq x_m$ .

Given this definition, the Dip Test of Unimodality measures how far  $\mathcal{F}_n(\mathbf{x})$  deviates from  $\mathcal{F}_U$ . Concretely, the Dip statistic is the following computation:

$$D_{stat} = | \mathcal{F}_n(\mathbf{x}) - \mathcal{F}_U(\mathbf{x}) |_{\infty} : \mathbf{x} \in \text{supp}$$

Informally,  $D_{stat}$  is then the infinite norm of the differences of the EDF and the best fitting unimodal distribution over the support of these functions. It is straightforward to see that since we compute the infinite norm, considering the kernel of these functions is wasteful computation, hence the statistic restricts to computing over the support **supp** of  $\mathcal{F}_n$ .

### 3 Statistical Modeling of Interleaved Attacks

It is useful to first devise a statistical model of interleaved attacks in their most generic form: without any assumptions on both the attack itself as well as the benign program within which the attack is interleaved. As intended by [13], this approach allows us to view *interleaved* attacks as a new mechanism of hiding *any* micro-architectural attack. Such an attack-agnostic modeling of interleaved attacks then informs a generic detection mechanism independent of considerations of the actual use case and thereby extends beyond case studies presented in this work to any micro-architectural attack in general.

#### 3.1 Viewing HPCs as Gaussian Samplers

The first intermediary requirement is to understand what kind of distribution do HPCs adopt. By design, all Hardware Performance Counters (HPCs) are monotonic in nature and increment at the frequency of the occurrence of events they count. For example, on Intel’s systems <sup>3</sup>, `INST_RETIRED.ANY` counts the number of x86 instructions retired and will increment at the frequency with which the backend’s instruction retirement unit retires instructions. Likewise, `BR_INST_RETIRED.ALL_BRANCHES` counts the number of retired branch instructions and will increment at the frequency with which the frontend’s branch predictor unit retires branches.

Now the *frequency* of HPC update may or may not be affected by certain *assumptions* on the software executing on the processor. As a concrete example, `ASSISTS.HARDWARE` <sup>4</sup> counts the number of hardware assists during speculation. It is evident that not all software will thereby lead to increments in this HPC; one has to write software with corner cases [5] that has opportunities for hardware assists in order to increment this HPC. Informally, we call such HPCs as requiring *assumptions* on the software: they require special conditions in software to trigger increments. On the other hand, HPCs like `INST_RETIRED.ANY` are more relaxed in assumptions since *all* software triggers the instruction retirement unit in the backend execution unit. Informally stating, no matter what software executes, `INST_RETIRED.ANY` will always increment. Using this, we state an observation.

**Observation 1 (Modeling HPCs as Gaussian Samplers)** . *HPCs that do not enforce assumptions on software can be reliably modeled as Gaussian samplers. Because variations in their increments are not from assumptions in the software executing, but from infrequent events like non-maskable interrupts, non-voluntary context switches, priority scheduling from the kernel, and so on.*

Some concrete examples <sup>5</sup> of such HPCs: `INST_RETIRED.ANY`, `BR_INST_RETIRED.ALL_BRANCHES`, and `LONGEST_LAT_CACHE.REFERENCE`. Note that `BR_INST_RETIRED.`

<sup>3</sup> This with without loss of generality. A similar argument holds on other systems (like AMD) as well.

<sup>4</sup> <https://perfmon-events.intel.com/> ; Alder Lake and Sapphire Rapids

<sup>5</sup> <https://perfmon-events.intel.com/>

ALL\_BRANCHES (counts the number of branches) and LONGEST\_LAT\_CACHE.REFERENCE (counts the number of memory accesses) do impose *some* assumptions on software in order to increment: ① BR\_INST\_RETIRED.ALL\_BRANCHES requires `call` and `jmp` instructions, while ② LONGEST\_LAT\_CACHE.REFERENCE requires `load/store` instructions. We also consider them in our *preliminary analysis* since loops, function calls, memory operations, if-else branches, etc. are fairly common programming constructs. However, note that INST\_RETIRED.ANY increments for *any* instruction, and therefore imposes *absolutely* no assumptions on the software. Thereby, in *deployment* of our eventual detection mechanism, we only consider INST\_RETIRED.ANY.

### 3.2 Viewing Interleaved attacks as Bimodal Gaussian Samplers

We borrow the skeleton of an interleaved attack from [13]. Refer Listing 1.2: in a benign function, with very constrained probability, some micro-architectural attack is executed. For now, we place no assumptions on `attack()` or `benign()`. With this understanding, we combine our observation of Sec. 3.1 to state another observation:

```

1 void interleaved() {
2     benign();
3     // this threshold is very small
4     if(sample_probability() < threshold)
5         attack();
6     benign();
7 }
```

Listing 1.2: A skeleton of interleaved attack from [13].

#### Observation 2 (Modeling Interleaved attacks as Bimodal Gaussians)

. Considering measurements from HPCs that do not enforce assumptions on software, interleaved attacks can be reliably modeled as samples from Bimodal Gaussian mixtures.

Briefly, for the same HPC, the execution in Listing 1.2 either executes the function sequence (`benign()` + `benign()`) or the function sequence (`benign()` + `attack()` + `benign()`). Since the HPCs considered do not enforce any assumptions on these functions, measurements from such HPCs are essentially Gaussian samples. Finally, since there are two execution paths, the overall distribution of Listing 1.1 shall be a mixture of two Gaussian (i.e. a Bimodal Gaussian), written as  $p(\mathbf{x})$ :

$$p(\mathbf{x}) = w_1 \cdot \mathcal{N}(\mathbf{x}; \mu_1, \sigma_1) + w_2 \cdot \mathcal{N}(\mathbf{x}; \mu_2, \sigma_2)$$

Where  $\mathcal{N}(\mathbf{x}; \mu_1, \sigma_1)$  corresponds to the execution path (`benign()` + `benign()`), and  $\mathcal{N}(\mathbf{x}; \mu_2, \sigma_2)$  corresponds to the execution path (`benign()` + `attack()` + `benign()`). Also note that since `threshold` is small,  $w_2 \ll 1$ . Finally,  $w_1 + w_2 = 1$  since either of the two execution paths will definitely execute in each run of `interleaved()`.

### 3.3 Motivating Example of an Interleaved Attack

Consider a concrete instantiation of Listing 1.2 as an actual attack as in Algo. 1. Concretely, in Algo. 1, the execution pattern of the interleaved attack comprises a Flush+Reload covert channel as the malicious code and a matrix-matrix multiplication operation as the benign execution. The interleaved attack is governed by a threshold mechanism, where the threshold dictates whether only the benign code or both the benign and malicious code are executed concurrently. Initially, the threshold is set to 10000, with `num_of_samples` configured as  $5 \times \text{threshold}$  and `num_of_traces` set to 100000. Under this configuration, the malicious code is executed at an interval of  $\frac{1}{10000}$ , while the benign code dominates execution during the remaining iterations.

From the discussion in Sec. 3.1, for *preliminary analysis*, we consider `BR_INST_RETIRED.ALL_BRANCHES`, `INST_RETIRED.ANY`, and `LONGEST_LAT_CACHE.REFERENCE`<sup>6</sup>. In Fig. 1a, Fig. 1b, and Fig. 1c, we give concrete distributions of these HPCs for Algo. 1 (but without `attack()`, i.e. no interleaving). It is clear how these HPCs follow a Gaussian distribution.

Should we enable `attack()` (i.e. enable interleaving), the concrete distributions of `BR_INST_RETIRED.ALL_BRANCHES`, `INST_RETIRED.ANY`, `LONGEST_LAT_CACHE.REFERENCE` are as captured in Fig. 2a, Fig. 2b, and Fig. 2c. Note how the **red** curves in the figures have a *short* second peak, denoting the occurrence of interleaved attacks<sup>7</sup>, signifying the low threshold with which `attack()` is interleaved in `benign()` execution.

---

#### Algorithm 1 Interleaved Attack Example

---

```

1: procedure ATTACK_CODE
2:   flush_reload();
3: procedure BENIGN
4:   matmul();
5: procedure MAIN
6:   for outer = 1 to num_of_traces do
7:     for inner = 1 to num_of_samples do
8:       if inner % threshold = 0 then
9:         attack();
10:      benign();

```

---

<sup>6</sup> Actual deployment of the detection mechanism is only using `INST_RETIRED.ANY`, for reasons detailed in Sec. 3.1

<sup>7</sup> The bimodal nature of `interleaved()` is software-agnostic because of the choice of HPCs, that do not enforce any assumptions on software. As such, this phenomenon is true when `benign()` executes SPEC2017, which is the actual `benign()` implementation in [13]. Refer Sec. 5 for details on runs through SPEC2017.



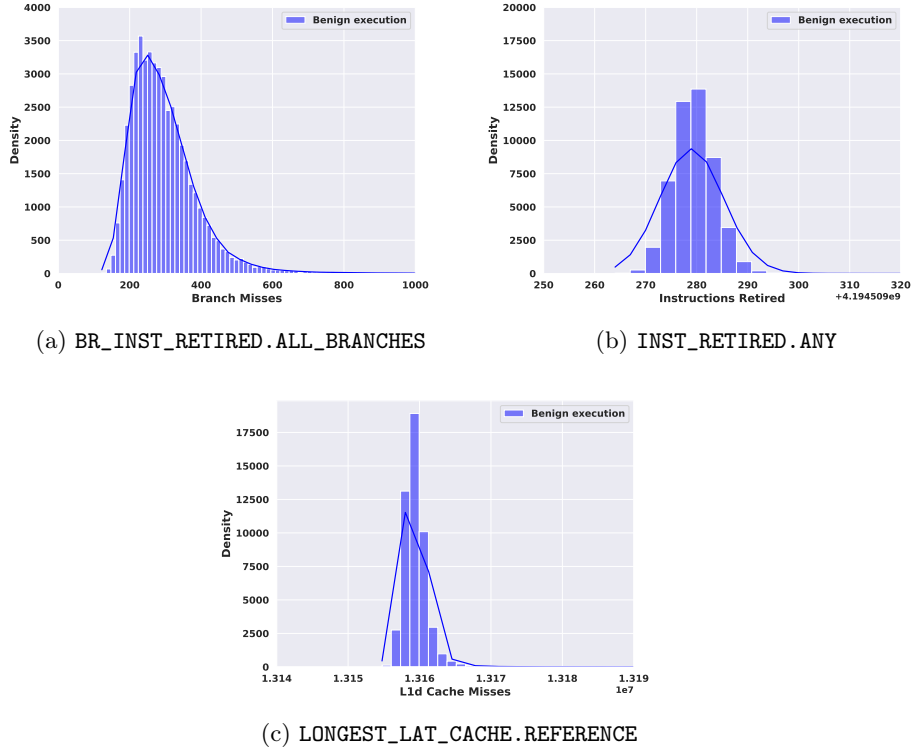


Fig. 1: Distribution of different HPCs with respect to Algo. 1, but without `attack()`.

### 3.4 Fitting Gaussian Mixture Model with BIC/AIC criteria

Finally, to formally establish the bimodality of interleaved attacks, we fit a Gaussian Mixture Model (GMM) with multiple modalities and leverage the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) to determine the optimal number of Gaussian components. This approach facilitates the identification of distinct latent distributions within the HPC readings, corresponding to various operational modes, including both normal execution and attack-induced anomalies. The GMM approach allows us to rigorously model the heterogeneity in the HPC readings and conclusively compute the modality of the distribution enforced by interleaved attacks.

Tab. 1 captures the BIC and AIC for when GMM is fitted to the readings collected for `INST_RETIRED.ANY` while the interleaved attack is operational (i.e. the red curve in Fig. 2b). Concretely, the GMM models the data as a mixture of  $k$  Gaussian distributions. The likelihood of data  $x$  under a GMM with  $k$  components is:

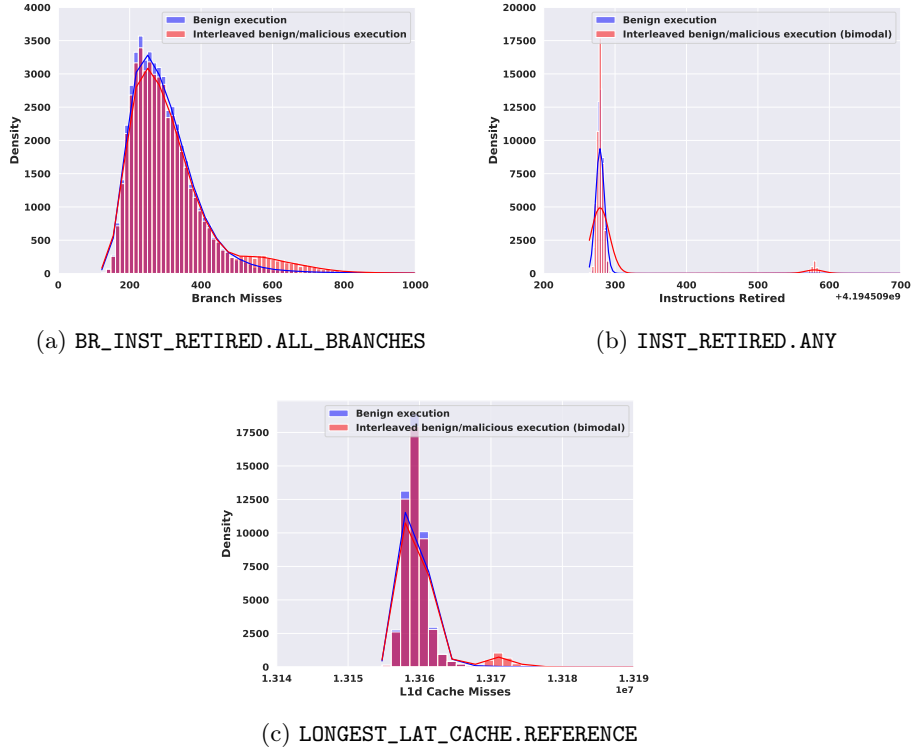


Fig. 2: Distribution of different HPCs with respect to a sample program (matrix-matrix multiplication) and interleaved attack code (Flush+Reload covert channel) with interleaving probability  $\frac{1}{10000}$ .

$$p(x) = \sum_{i=1}^k \pi_i \mathcal{N}(x|\mu_i, \sigma_i) \tag{1}$$

where  $\pi_i$  are the mixing coefficients<sup>8</sup>,  $\mu_i$  is the mean of the  $i$ -th Gaussian, and  $\sigma_i$  is the standard deviation of the  $i$ -th Gaussian. We fit GMMs with  $k \in \{1, 2, 3, 4, 5\}$  to parameterize the underlying distribution enforced by INST\_RETIRED.ANY. Final model selection is based on BIC and AIC, which balance model fit and the complexity (i.e. number of modes in the multi-modal Gaussian fitted):

$$\text{AIC} = 2p - 2 \ln(L), \quad \text{BIC} = p \ln(n) - 2 \ln(L) \tag{2}$$

Where  $p$  is the number of model parameters,  $L$  the log-likelihood, and  $n$  the number of data points. Lower BIC/AIC values indicate accurate modeling. In

<sup>8</sup> The probability that  $x$  belongs to the  $i$ -th Gaussian in  $p$ .

case of Tab. 1, notice how for two components (i.e. when GMM fits a bimodal to the red curve in Fig. 2b), we observe the optimal BIC and AIC, thereby further establishing the bimodality of interleaved attacks. The primary peak corresponds to normal behavior, while the secondary peak represents deviations caused by interleaved attacks. A similar result of bimodality was also observed while trying other HPCs in a similar setting of interleaved attacks.

Components in GMM	BIC	AIC
1	901113.744073	901094.718222
2	404847.192319	404799.627691
3	405288.252292	405212.148888
4	405601.373257	405496.731076
5	405717.546513	405584.365557

Table 1: BIC and AIC for INST\_RETIREDA.NY while interleaved attack is operational. red marking denotes the optimal/minimum BIC and AIC.

## 4 Generic Detection Strategy for Interleaved Attacks

Sec. 3 conclusively establishes interleaved attacks to follow a bimodal gaussian distribution, without any assumptions on `attack()` or `benign()` (cf. Listing 1.2). It is straightforward to then infer that a suitable statistical test for unimodality is sufficient to detect such attacks. In this work, we rely upon the Dip test for unimodality<sup>9</sup>. Therefore, to determine whether the system is in an attack-free state or actively under attack, we employ the *Dip Test* as a first step to validate the dataset’s modality, followed by a detailed analysis of the relative prominence of these peaks.

### 4.1 Dip Test Confirmation of Bimodality

The *Dip Test of Unimodality* [11] is a non-parametric hypothesis test used to assess whether a dataset deviates from unimodality by measuring the maximum deviation of the empirical distribution function (EDF) from the closest unimodal distribution. In this context, the Dip Test allows us to statistically confirm the presence of two distinct modes in the dataset, representing normal (benign) and attack-induced behaviors. The null hypothesis  $H_0$  of the Dip Test is: the dataset follows a unimodal distribution, while the alternative hypothesis  $H_1$  posits that the data is of modality  $> 1$ .

Upon applying the Dip Test to the dataset, the rejection of the null hypothesis  $H_0$  provides evidence that the distribution is indeed bimodal. This confirms that

<sup>9</sup> Note that the widely used Welch’s t-test is unsuitable here, as it assumes unimodal normality in source distribution.

the dataset contains two distinct Gaussian components—one associated with benign system behavior and the other with interleaved attack patterns.

#### 4.2 Interpreting the Results: Determining Attack Presence

While the Dip Test confirms the dataset’s bimodality, it does not directly indicate the current operational state— whether the system is functioning normally or is under attack. To address this, we proceed by analyzing the relative distribution of data points between the two identified peaks (corresponding to the benign and attack modes). The following criteria are employed:

**benign() execution.** The system can be considered to be in a normal (attack-free) state if the majority of data points are concentrated around the primary peak, which corresponds to normal system behavior. In this case, the contribution of the secondary peak, representing attack activity, is negligible or non-existent. Mathematically, this can be expressed by evaluating the probability density under the primary Gaussian component. If the proportion of data points assigned to the primary peak exceeds a predefined threshold  $\theta$ , the system is classified as *attack-free*:

$$P(\text{primary peak}) > \theta$$

where  $\theta$  is a high confidence threshold, typically set around 95%, to ensure robust detection of normal behavior.

**benign() + attack() execution.** Conversely, an ongoing attack is indicated if a non-trivial portion of data points lies within the secondary peak, suggesting frequent deviations from the normal operational mode. The magnitude of the secondary peak, characterized by its mixing coefficient in the Gaussian Mixture Model (GMM), becomes a key metric for identifying an attack. Specifically, we monitor the ratio between the mixing coefficients  $\pi_2$  of the secondary peak (attack behavior) and  $\pi_1$  of the primary peak (normal behavior):

$$\frac{\pi_2}{\pi_1} > \delta$$

where  $\delta$  is a predefined threshold (0.05 in our experiments) that reflects the level of sensitivity to anomalies. If the mixing coefficient ratio exceeds this threshold, it indicates that the system is under attack, as the secondary peak is no longer merely an outlier but has become significant in relation to the primary peak.

We summarize the takeaways from Sec. 3 and Sec. 4 below:

1. HPCs like `INST_RETIRED.ANY` do not enforce any assumption on the software, and thus are closely modeled by unimodal Gaussians, with operational noise.
2. Interleaved attacks take one of two possible paths: `benign()` or `benign() + attack()`. Thereby, when interleaved attacks are sampled through `INST_RETIRED.ANY`, we observe two distinct modes, i.e. a bimodal Gaussian. The peak corresponding to `benign() + attack()` is of much shorter since `benign() + attack()` executes with very less probability.
3. Since the difference between `benign()` and `benign() + attack()` is the modality of the Gaussian, statistical tests for unimodality suffice to detect when `benign() + attack()` executes, irrespective of the frequency of execution.

## 5 Case Studies

In this section, we demonstrate the detection of various attack types concealed within benign code, collectively forming an interleaved attack. All experiments were conducted on Intel systems, with specifications detailed in Table 2. In newer generation Intel systems, speculative execution attacks like Spectre v1, Spectre v2, Spectre v3, and Meltdown are mitigated. For our experiments, we disabled all such mitigations, wherever needed.

Fig. 3 summarizes the end-to-end detection framework. Unlike Sec. 3, the `benign()` execution throughout this section is `gcc_r` from SPEC2017 (as used in [13] as well). Finally, it is clear from Sec.3 and Sec. 4 that the detection strategy follows trivially from modeling the statistical nature of interleaved attacks, and using the correct test to detect. However, throughout this section, we choose to *instantiate* the detector through LLMs, for a number of reasons. First, LLMs abstract away the working details of the detection and allow users to interact with it in natural language; this allows for: ① an almost non-existent learning curve to *use* our detector in deployment. Secondly, LLMs also abstract away contextualization of data from multiple sources (i.e. HPCs); this allows ② the flexibility to *add* more HPC sources to the detection methodology than what we currently demonstrate. Finally, again with simple prompts in natural language, users can easily switch between statistical tests; this allows for ③ flexibility to add other statistical tests to our detector in addition to the Dip Test for Unimodality to cover other use-cases outside the scope of this work<sup>10</sup>.

<sup>10</sup> For example, a real *deployment* of detection would like to detect *both* state-of-the-art legacy cache side-channel attacks, as well as interleaved attacks. Thus such detectors would need to switch between ① statistical tests, ② data sources (HPCs), ③ detec-

Generation	Model Name	Code Name
13th Generation	Intel(R) Core(M) 17-13700	Raptor Lake
12th Generation	Intel(R) Core(M) 15-12500	Alder Lake
8th Generation	Intel(R) Core(TM) 17-8700 CPU @ 3.20GHz	Coffee Lake

Table 2: Processor details for systems on interleaved attacks are performed

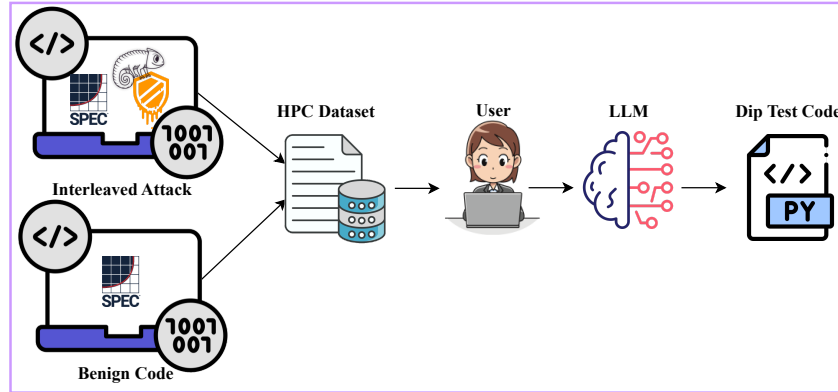


Fig. 3: Detection Strategy for Interleaved Attacks leveraging LLM

### 5.1 Spectre v1

In Algorithm 1, we initially employed a basic matrix multiplication operation as the benign workload, alongside a simple Flush+Reload attack as the malicious component. This served as the foundation for developing a generalized attack detection methodology. Expanding on this, we now extend the framework to cover a broader range of attack types. For this, we use the `gcc_r` benchmark from the SPEC CPU2017 suite as the benign code, which reflects a more realistic and complex execution environment, as discussed in Kosasih et al.[13]. In contrast, the malicious workload consists of a Spectre v1 attack, configured to execute with a very low probability to ensure it remains hidden within the legitimate activity of the benign program. In spectre v1 attack, the attacker manipulates the CPU’s branch predictor, causing it to incorrectly predict the direction of a branch. This leads the CPU to execute instructions speculatively that would not normally be executed according to the correct program logic. As a result, the attacker can access sensitive data stored in the program’s memory. During

---

tion intervals, and ④ other parameters to effectively operate. While such a *combined* detector is essentially programmable in a high-level programming language, using LLMs allows abstracting out much inner details, allowing for an easier operational interface for the end-user.

execution of the interleaved attack, we collect hardware performance counter (HPC) values to capture the system’s runtime behavior.

As detailed in Section 4, the captured HPC dataset is subsequently input into a large language model (LLM), which is tasked with performing the Dip test. The Dip test serves as a crucial tool for detecting multimodal distributions within the data, which are indicative of interleaved attacks, such as Spectre v1. Leveraging the LLM for conducting the Dip test enhances the detection process by utilizing its advanced pattern recognition capabilities, which may outperform traditional statistical approaches in identifying subtle anomalies. By running the Dip test on the HPC dataset, we can ascertain whether interleaved malicious activity is occurring, thus providing a robust mechanism for detecting stealthy attacks embedded within legitimate workloads.

#### ChatBox 1: Spectre v1 Interleaved Attack detection

**User:** Hi! Can you kindly perform a Dip test on the provided HPC dataset file to check whether my system is currently undergoing any interleaved attacks?

**LLM:** Sure!! Here is the code for the dip test you may execute it on our system to check for interleaved attacks, executing on your system.

```
import pandas as pd
from diptest import diptest
file_path = "file_path.txt"
data = pd.read_csv(file_path, header=None)
data_values = data[0].values
dip_stat, p_value = diptest(data_values)
print("Dip Test statistic:", dip_stat, "p-value:", p_value)
if p_value < 0.05:
    print("Yes!, Interleaved attack is executing.")
else:
    print("Your system is safe")
```

After running this, the dip test will tell you if there are multiple peaks in your dataset. A p-value below 0.05 would refer that your system is undergoing an interleaved attack.

Moreover, the integration of an LLM in the detection process not only adds a layer of sophistication in interpreting multimodal data distributions but also strengthens the overall detection framework. This hybrid approach, combining statistical testing with machine learning techniques, ensures more accurate and efficient detection of low-probability malicious executions such as Spectre v1, which are designed to evade conventional detection mechanisms.

## 5.2 Spectre v2

In the preceding case study 5.1, we demonstrated the detection of an interleaved attack where the malicious activity was attributed to Spectre v1. In this section,

we extend our analysis by considering Spectre v2 as the malicious component within the algorithm 1, while the benign workload remains unchanged, specifically the `gcc_r` benchmark. In spectre v2 the attacker targets a specific code sequence, or 'gadget,' in the victim's memory by exploiting the Branch Target Buffer (BTB). By mis-training the BTB to predict the wrong target for an indirect branch, the attacker redirects speculative execution to the gadget's address. While the CPU corrects the mis-prediction, changes to the cache persist, allowing sensitive data extraction via cache side-channel attacks. The attack is successful by aligning virtual addresses between the attacker and victim memory, even without executable code at the attacker's gadget address

As with 5.1, we collect HPC events, particularly `BR_INST_RETIRED.ALL_BRANCHES` and `INST_RETIRED.ANY`, during the execution of the interleaved attack.

The collected HPC event data is processed through an LLM, which outputs a corresponding Dip test code. This code is executed on our system to assess whether the interleaved attack is successfully mounted.

### 5.3 Spectre v2 with advanced gadgets

Spectre v2 revealed critical weaknesses in modern CPU architectures, exploiting branch target injection to force speculative execution down unintended paths, resulting in data leakage through side channels. Although hardware and software mitigations like IBRS and Retpoline have been introduced, attackers have developed advanced gadgets to overcome these defenses and exploit speculative execution more effectively. One prominent class of these advanced techniques includes dispatch gadgets, which allow attackers to gain greater control over CPU registers and execution flow. Key types of dispatch gadgets include:

**Dispatch-to-Call** attackers redirect control to unexpected but legitimate code segments, manipulating CPU registers through intermediary steps.

**Dispatch-to-Any** allows jumps to arbitrary locations when Indirect Branch Tracking (IBT) is disabled, expanding the attack surface.

**Dispatch-to-Dispatcher** multiple dispatchers are chained, allowing attackers to orchestrate more complex speculative execution paths.

So we have replaced the malicious code present in Algo. 1, one of these three gadgets at a time, and measured the HPC events. Following a similar process shown in 5.1, we get the Dip test code, with which we can check for the interleaved attack getting mounted on the system or not. With this too, we achieve a 100% detection rate.

### 5.4 Spectre v3

Spectre v3 is a variant of Spectre family of attacks that rely upon Branch History Injection. Concretely, the attack poisons the branch predictor by maliciously



injected branches, which forces the processor (under speculation) to mis-issue incorrect branches. As with previous case studies, we use `INST_RETIRED.ANY` to detect Spectre v3, and succeed with a 100% detection rate.

## 5.5 Meltdown

Finally, we also test our detection methodology with Meltdown which follows a completely different attack strategy than Spectre v1 and its later variants. Precisely, in face of a faulted `load`, data belonging to the victim address space gets incorrectly forwarded to the adversarial address space under speculation. The adversary then uses a covert channel (like Flush+Reload) to encode such transiently forwarded data into the cache. The predominant reason for Meltdown is aggressive speculation in on-core buffers, which do not respect software isolation boundaries.

Our detection methodology, along with the choice of HPC `INST_RETIRED.ANY`, is generic enough to warrant no additional change when we shift from Spectre to Meltdown. In this case as well, when Meltdown executes, a bimodal distribution is observed, that is captured by the Dip Test of Unimodality. This is irrespective of the frequency of interleaving Meltdown with `gcc_r`. In this case too, we achieve 100% detection rate.

## 6 Statistical Modeling of Interleaved Attacks with $> 1$ multiplicity

The interleaved attack strategy from [13] considers a single attack interleaved in benign execution (cf. Listing 1.1). Here, we consider an even stronger threat model: the adversary can interleave  $N > 1$  attack vectors within the same benign code. Concretely, such interleaved attacks with  $> 1$  multiplicity can be considered as what Listing 1.3 depicts.

```

1 void interleaved() {
2     benign();
3     // this threshold is very small
4     if(sample_probability() < threshold){
5         attack = pick_one(attack1, attack2, ..., attackN)
6         attack();
7     }
8     benign();
9 }
```

Listing 1.3: A skeleton of interleaved attack from [13].

Borrowing our discussion from Sec. 3, we can statistically model such an attack vector as a Multimodal Gaussian of form:

$$p(\mathbf{x}) = w_b \cdot \mathcal{N}(\mathbf{x}; \mu_b, \sigma_b) + w_1 \cdot \mathcal{N}(\mathbf{x}; \mu_1, \sigma_1) + w_2 \cdot \mathcal{N}(\mathbf{x}; \mu_2, \sigma_2) + \dots + w_N \cdot \mathcal{N}(\mathbf{x}; \mu_N, \sigma_N)$$

Where  $w_b \cdot \mathcal{N}(\mathbf{x}; \mu_b, \sigma_b)$  denotes the benign execution (`benign()` + `benign()`) and  $w_i \cdot \mathcal{N}(\mathbf{x}; \mu_i, \sigma_i); \forall i \in \{1, \dots, N\}$  denotes the execution (`benign()` + `attacki` + `benign()`). We recall that the *Dip test for Unimodality* is able to detect multimodal distributions as well. As with Sec. 4.2, the following criteria is applied:

**benign() execution.** As before, the system is in a normal state if the majority of data points are concentrated around the primary peak. As before, this is mathematically represented as:

$$P(\text{primary peak}) > \theta$$

**benign() + attack<sub>i</sub>() execution.** In this case, unlike before, we now have the ratio parameterized by  $i$ , to denote the  $i$ -th attack executing, such as:

$$\frac{\pi_i}{\pi_1} > \delta$$

If the ratio of mixing coefficients exceeds this threshold (0.05 in our experiments), it indicates that the system is under attack, as the secondary peak has become significant in relation to the primary peak.

## 7 Conclusion

In this work, we introduce a novel detection methodology for micro-architectural attacks in a probabilistically interleaved malicious/benign execution environment. By leveraging lightweight statistical tools such as Gaussian Mixture Models (GMM) and the Dip test for Unimodality, we demonstrate the ability to detect subtle deviations in hardware performance counter (HPC) data caused by low-frequency interleaved attacks. The detection process is further streamlined by utilizing Large Language Models (LLMs) to automate the generation of Dip test code, allowing users to efficiently identify multimodal distributions indicative of interleaved attacks. Our experiments show a 100% detection rate for attacks like Spectre v1/v2/v3 and Meltdown, even when interleaved with benign processes at very low frequencies.

This work refutes the assumption that probabilistically interleaved attacks are inherently undetectable by showcasing the effectiveness of lightweight statistical analysis in identifying such threats. While the LLM in this context primarily serves to provide automated Dip-test code, its integration facilitates quick and accessible implementation of detection strategies without requiring deep expertise in statistical analysis. Our findings highlight the importance of adaptable detection techniques and the need for further research into both advanced evasion tactics and more sophisticated countermeasures, ensuring continued resilience against evolving micro-architectural attacks.

## Acknowledgment

The authors would like to thank the reviewers for their suggestions for improving the paper. They would also like to thank the Department of Science and Technology (DST), Govt of India, IHUB NTIHAC Foundation, C3i Building, Indian Institute of Technology Kanpur, and Centre on Hardware-Security Entrepreneurship Research and Development, MeitY, Govt of India, for partially funding this research.

## References

1. Ahmad, B.A.: Detecting Spectre and Meltdown Attacks Using Hardware Performance Counters and Machine Learning. Ph.D. thesis, PhD thesis. University of the Punjab (2019)
2. Akram, A., Mushtaq, M., Bhatti, M.K., Lapotre, V., Gogniat, G.: Meet the sherlock holmes' of side channel leakage: A survey of cache sca detection techniques. *IEEE Access* **8**, 70836–70860 (2020)
3. Aldaya, A.C., Brumley, B.B., ul Hassan, S., García, C.P., Tuveri, N.: Port contention for fun and profit. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 870–887. IEEE (2019)
4. Aweke, Z.B., Yitbarek, S.F., Qiao, R., Das, R., Hicks, M., Oren, Y., Austin, T.: Anvil: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices* **51**(4), 743–755 (2016)
5. Chakraborty, A., Mishra, N., Mukhopadhyay, D.: Shesha: Multi-head microarchitectural leakage discovery in new-generation intel processors. *arXiv preprint arXiv:2406.06034* (2024)
6. Eresheim, S., Luh, R., Schrittwieser, S.: The evolution of process hiding techniques in malware-current threats and possible countermeasures. *Journal of Information Processing* **25**, 866–874 (2017)
7. Genkin, D., Poussier, R., Sim, R.Q., Yarom, Y., Zhao, Y.: Cache vs. key-dependency: Side channeling an implementation of pilsung. *IACR Transactions on Cryptographic Hardware and Embedded Systems* pp. 231–255 (2020)
8. Gonzalez-Gomez, J., Bauer, L., Henkel, J.: Cache-based side-channel attack mitigation for many-core distributed systems via dynamic task migration. *IEEE Transactions on Information Forensics and Security* **18**, 2440–2450 (2023)
9. Gullasch, D., Bangerter, E., Krenn, S.: Cache games—bringing access-based cache attacks on aes to practice. In: 2011 IEEE Symposium on Security and Privacy. pp. 490–505. IEEE (2011)
10. Gulmezoglu, B., Zankl, A., Tol, M.C., Islam, S., Eisenbarth, T., Sunar, B.: Undermining user privacy on mobile devices using ai. In: Proceedings of the 2019 acm asia conference on computer and communications security. pp. 214–227 (2019)
11. Hartigan, J.A., Hartigan, P.M.: The dip test of unimodality. *The annals of Statistics* pp. 70–84 (1985)
12. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., et al.: Spectre attacks: Exploiting speculative execution. *Communications of the ACM* **63**(7), 93–101 (2020)
13. Kosasih, W., Feng, Y., Chuengsatiansup, C., Yarom, Y., Zhu, Z.: Sok: Can we really detect cache side-channel attacks by monitoring performance counters? In: *AsiaCCS* (2024)

14. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., et al.: Meltdown: Reading kernel memory from user space. *Communications of the ACM* **63**(6), 46–56 (2020)
15. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE symposium on security and privacy. pp. 605–622. IEEE (2015)
16. Moghimi, D.: Downfall: Exploiting speculative data gathering. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 7179–7193 (2023)
17. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of aes. In: *Topics in Cryptology—CT-RSA 2006: The Cryptographers’ Track at the RSA Conference 2006*, San Jose, CA, USA, February 13–17, 2005. Proceedings. pp. 1–20. Springer (2006)
18. Ragab, H., Barberis, E., Bos, H., Giuffrida, C.: Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 1451–1468 (2021)
19. Yarom, Y., Falkner, K.: {FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In: 23rd USENIX security symposium (USENIX security 14). pp. 719–732 (2014)
20. Yarom, Y., Genkin, D., Heninger, N.: Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering* **7**, 99–112 (2017)
21. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-vm side channels and their use to extract private keys. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. pp. 305–316 (2012)