Large Pages, Large Leaks? Hugepage-Induced Side-Channels vs. Performance Improvements in Cryptographic Computations

Xinyao Li and Akhilesh Tyagi *

Posted Date: 20 November 2025

doi: 10.20944/preprints202511.1567.v1

Keywords: hugepages; Page-level side channels; performance monitoring unit (PMU); microarchitectural leakage; machine learning–based analysis; AES / RSA / ECC workloads; memory translation and TLB behavior

 doi:10.20944/preprints202511.1567.v1

Article

Large Pages, Large Leaks? Hugepage-Induced Side-Channels vs. Performance Improvements in Cryptographic Computations

Xinyao Li and Akhilesh Tyagi *

Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010, USA
* Correspondence: tyagi@iastate.edu

Abstract

Side-channel attacks leveraging microarchitectural components such as caches and translation lookaside buffers (TLBs) pose increasing risks to cryptographic and machine-learning workloads. This paper presents a comparative study of performance and side-channel leakage under two page-size configurations—standard 4KB pages and 2MB huge pages—using paired attacker–victim experiments instrumented with both Performance Monitoring Unit (PMU) counters and precise per-access timing using rdtscp(). The victim executes repeated, key-dependent memory accesses across eight cryptographic modes (AES, ChaCha20, RSA, and ECC variants) while the attacker records eight PMU features per access (`cpu-cycles`, `instructions`, `cache-references`, `cache-misses`, etc.) and precise rdtscp() timing. The resulting traces are analyzed using a multilayer perceptron classifier to quantify key-dependent leakage. Results show that the 2MB huge-page configuration achieves a comparable key-classification accuracy (mean 0.79 vs. 0.77 for 4KB) while reducing average CPU cycles by approximately 11%. Page-index identification remains near random chance (3.6–3.7% for PMU side-channels and 1.5% for timing side-channel), indicating no increase in measurable leakage at the page level. These findings suggest that huge-page mappings can improve runtime efficiency without amplifying observable side-channel vulnerabilities, offering a practical configuration for balancing performance and security in user-space cryptographic workloads.

Keywords: hugepages; Page-level side channels; performance monitoring unit (PMU); microarchitectural leakage; machine learning–based analysis; AES / RSA / ECC workloads; memory translation and TLB behavior

1. Introduction

Side-channel attacks have evolved from traditional cache-timing exploits to a broader class of microarchitectural channels that exploit shared system resources such as page tables, Translation Lookaside Buffers (TLBs), and Performance Monitoring Units (PMUs). These hardware components, while designed for performance optimization, can inadvertently expose timing and other public state-based variations that correlate well with private program states. Consequently, even unprivileged user-space processes can sometimes infer cryptographic keys, memory-access patterns, or control-flow paths from indirect hardware measurements.

While extensive research has analyzed cache-based side channels and speculative (transient)-execution vulnerabilities, the impact of *memory page granularity* on application performance and security—specifically the distinction between standard 4KB pages and 2MB hugepages—remains relatively underexplored. Page size fundamentally shapes how virtual addresses are translated into physical memory: 4KB pages produce frequent TLB lookups and fine-grained page-table entries, whereas 2MB hugepages consolidate 512 contiguous 4KB regions into a single TLB entry, significantly reducing translation overhead and improving spatial locality. These architectural differences can affect

 doi:10.20944/preprints202511.1567.v1

both (1) the amount of information leaked through memory translation events and (2) the overall runtime performance of memory-intensive workloads.

In particular, **page-level leakage**—the ability of an attacker to infer which page or memory region the victim accessed—represents an important but under-examined threat model. TLB collisions can form leakage paths similar to cache side-channels with larger granularity revealing an in-fix of collision address. Previous studies[1,1–3] have shown that page faults and TLB misses can reveal control-flow boundaries or secret-dependent memory regions, but it is unclear how the adoption of large pages alters this leakage surface. On one hand, hugepages may reduce fine-grained address observability of a cache because a single translation covers a broader address range; on the other hand, they may strengthen and aggregate microarchitectural correlations consolidated through pages, exposed by PMU counters.

The **execution time** of a cryptographic algorithm may improve with hugepages if spatial locality exceeding 4KB exists in the application. Spatial locality may lead to improved time performance for hugepage transfers on a page-fault. The hit rate of the TLB may also improve due to fewer TLB page table entries needed for the same working set.

This research gap motivates our central question:

Do large (2MB) pages amplify or mitigate side-channel leakage, particularly at the page-number granularity, and can they improve performance without degrading security?

To answer this question, we design a controlled attacker–victim framework that accommodates both configurations. The victim and attacker share a 2MB memory region that can be backed either by 512 independent 4KB pages or by a single 2MB hugepage. This design ensures that both setups operate over identical virtual address ranges and memory access sequences, allowing a fair comparison of leakage strength and runtime behavior. The goal is to classify the secret key of a cryptographic algorithm along some attributes as a leakage metric. Each experiment iterates across 512 4KB offsets with 25 repetitions per key, producing 12,800 PMU-based readings per cryptographic workload.

Our evaluation spans eight cryptographic modes—AES (GCM, ECB, CBC), ChaCha20, RSA (2048, RAW-256), and ECC (P-256, ECIES-256)—representing both block and stream ciphers as well as asymmetric primitives. Using attacker-side PMU tracing of eight hardware performance counters, we analyze both (i) *key-dependent leakage*, where classifiers attempt to infer the victim's cryptographic key class, and (ii) *page-number leakage*, where classifiers attempt to identify the accessed 4KB or 2MB page.

The results indicate that the 2MB hugepage configuration achieves **key-classification accuracy** comparable to the 4 KB baseline, showing no increase in side-channel vulnerability. Moreover, page-number identification accuracy remained near random-chance levels (3.6–3.7%), indicating that page-level information was not reliably recoverable through the PMU event set used. At the same time, the hugepage configuration delivers an average **11% runtime performance gain** as measured by CPU-cycle counters, reflecting reduced TLB misses and lower page-walk overhead.

These findings provide an important insight: adopting hugepages can enhance performance without amplifying observable side-channel leakage. From both a security and system-design standpoint, our study suggests that memory management configurations—often treated as purely performance optimizations—can play a nuanced but stabilizing role in balancing efficiency and side-channel resilience.

2. Related Work

Side-channel attacks exploiting microarchitectural behaviors have been widely studied in both cryptographic and machine learning contexts. Early attacks such as Prime+Probe and Flush+Reload [4] targeted shared cache states to leak sensitive operations. More recent work extends these ideas to TLB-based attacks [2], page-fault side channels [5], and speculative execution [6,7].

PMU-based side channels have also gained traction as they provide fine-grained visibility into architectural events without requiring privileged access [8]. Tools like `perf` and custom instrumenta-

 doi:10.20944/preprints202511.1567.v1

tion allow attackers to measure metrics like load/store uops, cache misses, and cycles per instruction, which can correlate with victim computation state.

Page size differences, specifically the use of hugepages, have been studied in the context of performance optimization [9] and TLB pressure [3], but their impact on leakage amplification is relatively underexplored. A few research efforts like HugeLeak [10] have hinted at the potential for hugepages to magnify signal strength by reducing TLB noise and improving spatial correlation.

In the ML domain, side-channel risks have been explored in weight and input recovery [11,12], architectural fingerprinting [13], and membership inference [14]. Our work contributes to this growing literature by showing that 2MB hugepages can maintain leakage fidelity while improving runtime efficiency, filling a gap in cross-page-size side-channel evaluations.

3. Background

3.1. Side-Channel Attacks

Side-channel attacks exploit unintended microarchitectural leakage—such as timing variations, cache behavior, or memory access patterns—to infer sensitive program state without directly reading secret data. Classical cache-based attacks such as Flush+Reload and Prime+Probe [4,5] have demonstrated that shared cache sets can expose fine-grained victim access patterns. More recent work expands the attack surface to branch predictors, speculative execution [6], row buffer locality, and TLB behavior [2]. These attacks remain effective even when cryptographic algorithms are mathematically secure and written in constant-time form.

3.2. PMU-Based Side Channels

Performance Monitoring Units (PMUs) expose hardware event counters including load/store uops, cache misses, branch mispredicts, and TLB refills. While intended for profiling, PMU events correlate with execution behavior and can leak control-flow or secret-dependent operations. Prior work shows that PMU-based leakage can extract AES keys, RSA exponent usage, ECC ladder behavior, or even ML model output class distributions [8,12,13,15]. PMU attacks are particularly practical in cloud and container environments. PMU-based side-channel attacks—especially those using the perf subsystem—do not require shared memory or precise timing mechanisms (e.g., rdtsc). In many configurations, they can be performed without root privileges and without relying on tightly coupled hardware resources between an attacker and the victim, making them feasible even in constrained or sandboxed environments.

3.3. Huge Pages

Huge pages increase virtual memory page size (e.g., from 4KB to 2MB), reducing the number of required page table entries and decreasing TLB pressure. They are widely used in HPC, GPU workloads, and ML frameworks for improved throughput [16,17]. However, their impact on side-channel leakage is nuanced: reducing TLB churn may reduce noise, thereby reducing variance in PMU trace signatures [18]. Whether this stability improves or weakens security depends on the workload and threat model—motivating our empirical study.

3.4. Translation Lookaside Buffers (TLBs)

The TLB caches virtual-to-physical address translations. With 4KB pages, large memory regions require many entries, making TLB misses frequent and observable via PMU events. With 2MB pages, the same memory footprint requires far fewer entries [17]. This reduces page walks and improves runtime efficiency, but it also changes microarchitectural observability. If repeated accesses fall within the same hugepage region, leakage patterns may become more traceable since attacker can create collisions with a victim page more easily. On the flip side though a hugepage's observable activity is more clustered reducing the attributability to a more specific code segment, making it difficult to draw fine granularity correlations.

 doi:10.20944/preprints202511.1567.v1

3.5. Cryptographic Workloads

Cryptographic kernels such as AES, RSA, and ECC exhibit repeated table lookups, modular arithmetic, or structured memory fetch patterns tied to key material. Even constant-time implementations can exhibit identifiable PMU signatures [15]. Repeated execution enables classification of key-dependent features. Our evaluation tests eight cryptographic workloads to isolate how page size affects computation state leakage.

3.6. Machine Learning (ML) Inference

ML inference workloads allocate large contiguous tensors for activations and weights, making them well-suited for huge pages. However, ML class decisions affect memory traversal order in DNN layers, attention, and embedding lookup. Prior work shows attackers can infer output labels, recover weights, or extract model structure using PMU or cache-based traces [12,13]. Thus, evaluating hugepages in cryptographic leakage must also consider ML leakage implications.

4. Experimental Setup

We evaluate page-level side-channel leakage using a custom attacker / victim pipeline built from three components: attacker, victim, and machine learning classifier. The pipeline simulates inference-style workloads by issuing repeated, key-dependent memory accesses from the victim while the attacker records microarchitectural activity via Performance Monitoring Unit (PMU) counters and timing-based leakage using the rdtscp() counter; these traces are then aggregated and used for classification and leakage analysis.

4.1. Rationale of the Experimental Leakage Model

Although our experimental setup abstracts the full complexity of cryptographic libraries, its structure—reading plaintext from shared memory, performing encryption, and writing ciphertext—faithfully represents the dominant leakage mechanisms of real-world implementations. In deployed cryptographic systems, data movement across memory pages (e.g., key loading, block I/O, and ciphertext writes) generates observable TLB and cache activity that often dominates side-channel leakage. Our framework isolates this process using a controlled *read–compute–write* cycle, allowing precise attribution of leakage sources.

Importantly, this design captures two distinct leakage channels.

(1) PMU-based leakage (aggregate microarchitectural effects). Standard hardware performance counters (e.g., `cpu-cycles`, `cache-misses`, `branch-misses`) capture coarse-grained microarchitectural variation across the *entire* victim execution. These events reflect instruction-, data-, and control-flow– dependent differences during the computation phase in addition to effects from page reads and writes. Because PMU counters are architecture-portable and do not directly expose TLB-hit or TLB-miss timing, they provide a broad but lower-resolution view of leakage that aggregates all microarchitectural behaviors. Thus, PMU-based leakage is a coarse-grained indicator of how page size influences overall side-channel visibility.

(2) Timing-based leakage (fine-grained TLB Prime+Probe analogue). To directly measure translation-sensitive leakage, we also evaluate a timing-centric channel using the `rdtscp()` instruction to record precise *per-access* latency. This mirrors classical Prime+Probe attacks: the attacker primes translation structures, the victim executes, and the attacker measures probe latency that reveals TLB hits, TLB misses, and page-walk collisions. In our model, translation-sensitive effects arise only from the shared-file plaintext read and ciphertext write phases— meaning that the timing leakage we observe represents a *lower bound* on the full leakage surface of a complete cryptographic implementation.

Together, these two channels provide a faithful yet analytically tractable view of how page-size configuration (4 KB vs. 2 MB) influences side-channel behavior in cryptographic workloads: PMU counters capture global execution effects, whereas `rdtscp()` exposes fine-grained translation timing analogous to classic TLB Prime+Probe attacks.

 doi:10.20944/preprints202511.1567.v1

4.2. Mappings and Page Sizes

The attacker and victim both `mmap` the same shared backing file of size 4096×512 bytes (2MB). The file therefore covers 512 standard 4KB pages in the process address space; as shown in figure 1, we run two mapping configurations that differ only in how the kernel backs those virtual addresses:

- **4KB (standard pages):** the file is mapped using a normal `mmap` (no `MAP_HUGETLB`) so the kernel backs the region with 512 independent 4KB Page Table Entries (PTEs). This is our baseline: attacker and victim touch identical byte offsets, but the kernel/hardware treat each 4KB page separately (TLB entries, page-walks, etc.).
- **2MB (huge pages):** the file is mapped so it is backed by true 2MB hugepage PTE (each hugepage covers entire 512×4KB region). This is done by reserving and mounting hugepages on the host. In this configuration, a single TLB entry covers the entire 2MB region.

4.2.1. How We Provision Hugepages (Host Commands)

To run the 2MB experiment we reserve and mount hugepages on the host (root required), for example:

```
sudo sh -c 'echo 512 > /proc/sys/vm/nr_hugepages'
sudo mkdir -p /mnt/hugepages
sudo mount -t hugetlbfs nodev /mnt/hugepages
truncate -s 2M /mnt/hugepages/tlbtestfile_2mb
```

For the 4KB baseline we create the same-size file in a normal filesystem (no special privileges):

```
truncate -s $((4096*512)) /tmp/tlbtestfile_2mb
```

4.2.2. Why We Use the Same 2MB Shared File for Both Page-Size Configurations

To ensure that the two mapping configurations are directly comparable, we use the same 2MB backing file for both the 4KB (baseline) and 2MB (hugepage) experiments. Using the identical file keeps user-space offsets, address arithmetic, and code paths identical while allowing the kernel to back the virtual addresses either with 512 independent 4KB PTEs or with (pre-reserved) 2MB hugepages. Critically, this design also guarantees the same number of measured accesses in both settings: the attacker and victim iterate over the same set of 512 distinct 4KB page offsets, and each offset is touched for 25 independent repetitions. Thus, for every experimental key we collect

$$512 \text{ page offsets} \times 25 \text{ repetitions} = 12{,}800 \text{ PMU readings per key},$$

which makes aggregated comparisons (means, trimmed-means, classifier training, etc.) statistically comparable across the 4KB and 2MB experiments.

 doi:10.20944/preprints202511.1567.v1

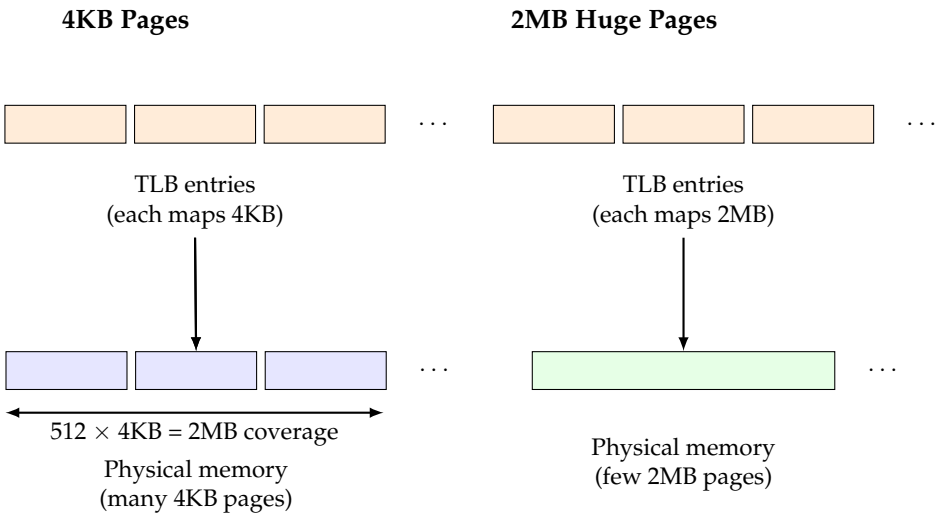

Figure 1. Comparison of TLB mappings under 4KB and 2MB pages. Each 2MB TLB entry covers 512 contiguous 4KB regions, reducing translation pressure and page walks.

4.3. Victim Behavior

The victim maps a 2MB shared file (mmap, MAP_SHARED) but performs its work on a single 4KB page chosen randomly per experimental run from the 512 pages in the mapping. For example, if `TARGET_OFFSET = 1572864` (bytes) it corresponds to page index $1572864/4096 = 384$ (page #385 in one-based numbering). For reproducibility, we fixed the selected page indeices per run. For each run, it seeds the PRNG deterministically from the provided hex key, selects the requested mode (`aes-gcm`, `aes-ecb`, `aes-cbc`, `chacha20`, `rsa-2048`, `ecc-p256`, `rsa-raw-256`, `ecc-ecies-256`), and then:

1. reads the plaintext from the mapped page starting at `TARGET_OFFSET` (full 4KB for page-sized modes; first 256 B for small-block modes);
2. executes the cryptographic kernel over that buffer (e.g., AES/ChaCha20 on 4KB; RSA-2048 hybrid or ECC-P256 hybrid on 4KB; RSA-RAW-256 or ECIES-256 on 256 B);
3. writes the result to a temporary file and drops a "`/tmp/victim_ready`" flag to synchronize with the attacker.

The victim is pinned to the same CPU core with attacker via `sched_setaffinity`.

4.4. Attacker Measurement

4.4.1. PMU-Based Measurements

Attacker measurements were collected on a local Intel® Core™ i5-6500 (Skylake, 4 cores, base up to 3.6 GHz) development machine. The CPU exposes a set of hardware performance monitoring events (e.g., `cpu-cycles`, `instructions`, `ref-cycles`, `cache-references`, `cache-misses`, `branch-instructions`, `branch-misses`, and `memory load/store` events) through the kernel `perf` subsystem. We instrument the attacker process using Linux `perf` (via `perf_event_open`) to gather eight PMU-derived features per measurement (recorded as `ev00`–`ev07` in the CSV traces).

We selected these eight counters to cover orthogonal microarchitectural effects that are likely to correlate with key-dependent memory activity: (1) `cpu-cycles` — coarse timing; (2) `instructions` — instruction throughput differences; (3) `ref-cycles` — stable cycle reference; (4) `cache-references` and (5) `cache-misses` — cache traffic and eviction/occupancy changes; (6) memory loads/stores — memory access intensity; and (7–8) `branch-instructions`/`branch-misses` — control-flow differences. Together these capture timing, data-flow, and control-flow signals used by downstream classifiers.

We opted to use these standard hardware events (via `perf`) rather than low-level raw/model-specific event encodings for several pragmatic reasons. Raw events can indeed expose more specialized counters (including some page-walk or TLB micro-events) but they are model-specific, fragile, and harder to reproduce: raw encodings require precise knowledge of the CPU's event MSR encodings,

 doi:10.20944/preprints202511.1567.v1

often need elevated privileges to program reliably, vary across generations and vendors, and are more likely to be multiplexed or filtered by platform microcode. In contrast, the standardized events exposed by the kernel are far more portable across machines and OS versions, can be collected with user-level privileges for the attacker process (subject to `perf_event_paranoid` settings), and are simpler to audit and reproduce across experimental runs and reviewers' systems. For these reasons we prioritized portability, reproducibility, and ease of collection for the main body of experiments.

A few practical notes that influenced collection and interpretation: (i) Skylake processors typically provide 4 programmable PMU counters plus several fixed counters (so roughly 7–8 hardware counters per core without multiplexing); requesting more events than supported triggers kernel time-multiplexing of counters, which can reduce temporal resolution. We therefore verified event availability under `/sys/bus/event_source/devices/cpu/events` and with `perf stat -v`, (ii) Occasional near-zero values indicated unsupported events or negligible signal for the microbenchmark; such cases were checked against the sysfs listing and re-run with alternate counters. (iii) Measuring the attacker process via `perf` requires only user-level sampling of the process itself, which simplifies experimental setup compared to system-wide tracing that can require root or relaxed `perf_event_paranoid` settings. These design choices produced a compact, portable fingerprint per repetition that supports the 1-vs-rest classification experiments reported in the paper.

4.4.2. Timing-Based Measurements

In addition to PMU-based measurements, we also implement a timing-centric attacker using the `rdtscp()` instruction to record *per-access* cycle latency. Whereas PMU counters provide aggregate microarchitectural statistics over an entire encryption operation, `rdtscp()` exposes fine-grained latency differences triggered by TLB hits, TLB misses, and page-walk contention between attacker and victim accesses. This mirrors classical Prime+Probe attacks on caches and TLBs ([19], [20], [21]): the attacker first primes the translation structures, the victim runs, and then the attacker probes the same address while logging cycle-level access time.

Data Aggregation and Feature Construction

From the raw CSV logs we construct features as follows:

1. Group raw samples by `(key_id, page_offset)`.
2. For each (key, page) grouping, we compute the mean across the 25 repetitions of each recorded measurement counter. This yields an 8-dimensional feature vector $\mathbf{x} \in \mathbb{R}^8$ for the PMU-based setting, and a 1-dimensional feature vector $\mathbf{x} \in \mathbb{R}^1$ when using timing-based (`rdtscp`) measurements.

These aggregated mean vectors are the inputs used for classification.

Cryptographic Workloads and Keys

We evaluate eight cryptographic modes that appear as victim workloads:

- AES: CBC, ECB, GCM (key lengths: 128/256 bits as used);
- ChaCha20 (typical 256-bit key);
- Elliptic-curve-based: ECIES-256 and P-256 signature/decryption flows;
- RSA: 2048-bit (standard) and a RAW-256 variant used as an experimental short-key workload.

(Exact key sizes used per run are recorded and reported in the result tables; the CSV trace files include a `key_size` column for reproducibility.)

 doi:10.20944/preprints202511.1567.v1

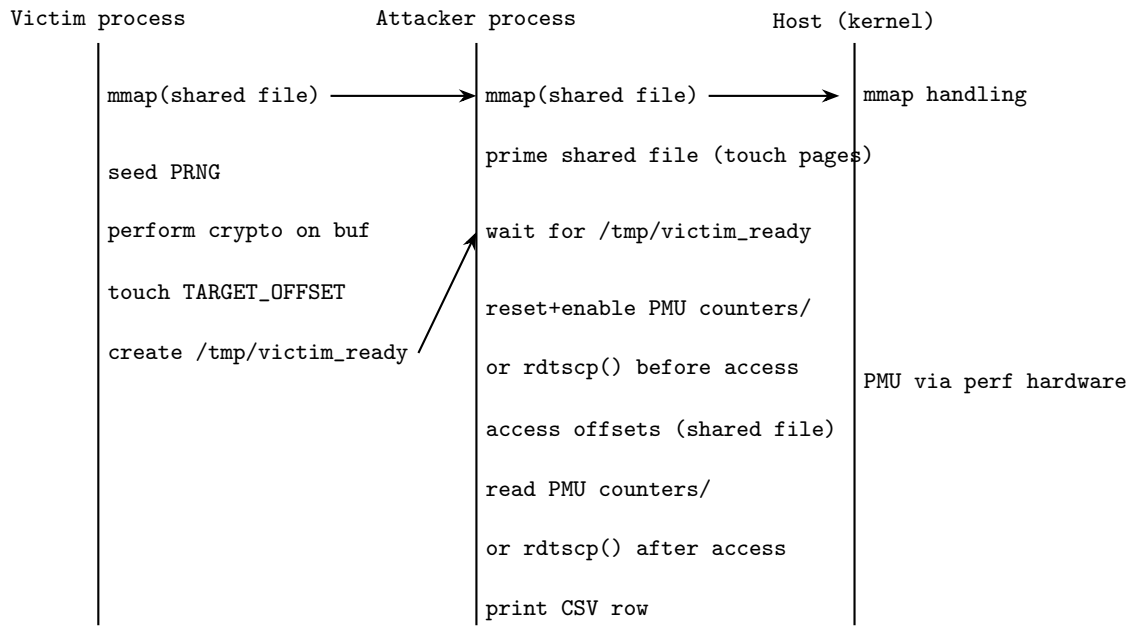

Figure 2. Sequence diagram of victim, attacker, and host (kernel) with priming and combined PMU/rdtscp measurements. Victim touches a single 4KB tile per run; attacker iterates all 512 offsets and records PMU counters/rdtscp().

4.5. Machine Learning Pipeline and Classification

To quantify information leakage from attacker-observable signals, we implemented an exhaustive key-classification framework in Python. For PMU-based experiments, each sample consists of the aggregated hardware-counter vector `ev00`–`ev07`, grouped by `(cryptographic mode, key_id, offset)` and averaged across repetitions, producing one eight-dimensional feature vector per (key, page-offset) example. For each cryptographic mode, we train and evaluate a small feed-forward neural network based on `scikit-learn`'s `MLPClassifier`. The model uses two hidden layers of sizes (64, 32) with ReLU activation and Adam optimization, and includes optional PCA dimensionality reduction for regularization [22–24]. PMU-based experiments therefore use the standardized pipeline:

$$\texttt{StandardScaler} \rightarrow \texttt{PCA} \rightarrow \texttt{MLP},$$

with hyperparameters selected via grid search over learning rate and L2 penalty.

In addition to the PMU modality, we also evaluate a *timing-based TLB leakage variant* using per-access cycle counts measured via `rdtscp()`. In this setting, each sample is represented by a single scalar feature—the averaged probe latency—i.e., $\mathbf{x} \in \mathbb{R}^1$. Because PCA is not meaningful in one dimension, the timing-based classifier uses a simplified pipeline

$$\texttt{StandardScaler} \rightarrow \texttt{MLP},$$

while keeping all other training, cross-validation, and evaluation procedures identical to the PMU-based setup. This variant tests whether fine-grained access latency alone encodes key-dependent information, serving as a direct analogue to Prime+Probe-style TLB timing attacks.

To evaluate discriminability, we adopt a one-vs-rest protocol where the classifier learns to distinguish each target key from all remaining keys (e.g., $\{0\}$ vs. $\{1, 2, 3\}$). For robustness, we remove the highest and lowest test accuracies across repetitions and report the trimmed-mean accuracy. Cross-validation uses a three-fold `StratifiedKFold` split to maintain class balance, with an independent 50% held-out test set for final evaluation.

This MLP-based approach offers a nonlinear yet lightweight baseline for studying side-channel leakage. The model captures correlations among timing-, cache-, and control-flow–related events

 doi:10.20944/preprints202511.1567.v1

without requiring large datasets. Compared to more complex models such as CNNs or random forests, the MLP provides strong performance while preserving interpretability and computational efficiency. All classifiers operate purely on user-space–accessible attacker measurements, requiring no privileged access to raw hardware counters.

4.6. Page-Level Leakage Experiments

To evaluate whether the victim's memory accesses reveal *which page* was touched during cryptographic execution, we designed a pair of attacker–victim experiments targeting page-index leakage at both 4KB and 2MB granularities. These experiments were conducted using two attacker-side measurement modalities: (i) Performance Monitoring Unit (PMU) counters and (ii) precise timing instrumentation via `rdtscp()`.

In both experiments, the attacker:

1. **Primes** all pages in the shared region.
2. **Waits** for the victim to perform a page access.
3. **Probes** each page again and records its measured access latency using PMU or `rdtscp()`.

4.6.1. PMU-Based Leakage (Coarse-Grained Analysis)

For the PMU-based experiments, the attacker records the standard hardware events (`ev00`–`ev07`) described earlier, and these aggregated feature vectors form the input to the classification pipeline. The victim performs controlled accesses to pages within a shared memory-mapped region, and the attacker attempts to infer the victim's accessed page index from the PMU signatures alone.

Two PMU-based configurations are evaluated:

- **4KB mapping:** the victim operates on standard 4KB pages, issuing a single 4KB access for each example.
- **2MB mapping with 4KB offsets:** the victim maps a 2MB hugepage-backed region but still performs accesses at 4KB granularity inside this region.
- **2MB mapping with 2MB offsets:** the victim performs accesses at 2MB granularity across multiple 2MB regions. The attacker maps and measures one 2MB region at a time, then advances to the next.

4.6.2. Timing-Based Leakage (Fine-Grained Prime+Probe Analogue)

To approximate a classic Prime+Probe-style TLB experiment, we also conduct a timing-based variant using the `rdtscp()` instruction, which provides cycle-accurate per-access latency. The same three granularities evaluated in the PMU-based setting (4KB pages, 4KB offsets inside a 2MB region, and 2MB region-level accesses) are also used here.

4.6.3. Objective

Across both modalities, the goal of the page-level leakage experiments is to determine whether attacker-observable microarchitectural signals—either coarse-grained PMU activity or fine-grained per-access timing—encode enough information to identify which page the victim accessed during its cryptographic workload.

5. Results

5.1. Key-Class Classification (PMU-Based)

Table 1 reports trimmed-mean test accuracy for one-vs-rest key classification per cryptographic mode under the two mapping configurations. Each entry reflects models trained on the aggregated PMU features (`ev00`–`ev07`) averaged over 25 repetitions for each (key, page-offset) example, with a 50% held-out test split and three-fold stratified CV for hyperparameter selection.

 doi:10.20944/preprints202511.1567.v1

Table 1. Trimmed-mean accuracy (test) for one-vs-rest key classification by mode and page size with PMU features.

Mode	Key Size	2MB Avg	4KB Avg
AES-CBC	128 bit	0.809	0.758
AES-ECB	128 bit	0.751	0.756
AES-GCM	128 bit	0.826	0.757
ChaCha20	256 bit	0.762	0.766
ECC-ECIES	256 bit	0.744	0.751
ECC-P256	256 bit	0.812	0.813
RSA-2048	2048 bit	0.802	0.791
RSA-RAW	256 bit	0.821	0.766

Overall, PMU-based key classification sits well above random guessing (0.5), with accuracies typically in the 0.74–0.83 range across modes. Comparing 2MB huge pages to 4KB mappings, the differences are modest: AES-GCM and RSA-RAW show a consistent but moderate advantage for 2MB, AES-EBC and ECC modes exhibit higher accuracy in 4KB mappings. This suggests that, at least for our setup, using 2MB huge pages does not qualitatively change the attacker's PMU-based key distinguishability.

5.2. Key-Class Classification (Time-Based)

We next repeat the exhaustive one-vs-rest key classification, but using only the attacker's execution time measured by rdtscp() as the only one feature. Table 2 reports the mean test accuracy across these four one-vs-rest splits for 2MB and 4KB mappings.

Table 2. Trimmed-mean accuracy (test) for one-vs-rest key classification using only execution-time cycles (`rdtscp() measurement`) as the feature.

Mode	Key Size	2MB Avg	4KB Avg
AES-CBC	128 bit	0.750	0.750
AES-ECB	128 bit	0.737	0.783
AES-GCM	128 bit	0.790	0.752
ChaCha20	256 bit	0.763	0.755
ECC-ECIES	256 bit	0.777	0.770
ECC-P256	256 bit	0.772	0.730
RSA-2048	2048 bit	0.749	0.750
RSA-RAW	256 bit	0.750	0.750

Time-based key classification also remains well above random guessing for all modes, with accuracies in the 0.73–0.79 range. Comparing 2MB huge pages to 4KB mappings, the differences are modest: AES-GCM and ECC-P256 exhibit a slight advantage for 2MB, whereas AES-ECB shows somewhat higher accuracy under 4KB, and the remaining modes are essentially comparable across page sizes. This mirrors the PMU-based results in Table 1, suggesting that the choice between 2MB and 4KB mappings does not fundamentally change the attacker's ability to distinguish keys using either PMU counters or timing alone.

5.3. Page-Level Leakage

We next evaluate whether cryptographic executions leak *the page number* accessed by the victim. Page-index identification is tested at both 4KB and 2MB granularity using two attacker-side modalities: (i) PMU-based measurements and (ii) timing measurements via `rdtscp()`.

PMU-based leakage (coarse-grained analysis).

Using the aggregated feature vectors (`ev00`–`ev07`) and the same neural classifier, we measure how well the attacker can infer which page was accessed.

Results:

 doi:10.20944/preprints202511.1567.v1

- **4KB configuration (predict 4KB page index):** accuracy = **3.6%**.
- **2MB mapping, 4KB offset (predict 4KB page index):** accuracy = **3.7%**.
- **2MB mapping (predict 2MB page number across 20 regions):** accuracy = **3.6%**.

These values are all near chance for the corresponding label spaces, indicating weak or negligible PMU-based page-index leakage.

Timing-based leakage (fine-grained Prime+Probe analogue).

To approximate a traditional Prime+Probe TLB attack, we repeat the experiment using cycle-accurate timing via `rdtscp()`. After priming the region, the attacker measures per-page access latency following the victim's access.

Measured accuracies:

- **4KB configuration (predict 4KB page index):** trimmed-mean accuracy = **2.1%**.
- **2MB mapping, 4KB offset (predict 4KB page index):** trimmed-mean accuracy = **1.6%**.
- **2MB mapping (predict 2MB page number across regions):** estimated accuracy = **1.5%**, consistent with 1-in-20 chance.

Again, all results are effectively at random-chance level, despite using high-resolution timing.

Interpretation.

Across both PMU-based (coarse-grained) and timing-based (fine-grained) experiments, page-index prediction remains extremely weak. PMU counters capture only aggregated microarchitectural activity and therefore cannot isolate TLB hit/miss behavior. Meanwhile, `rdtscp()` exposes per-access latency but still yields a noisy signal that fails to correlate with the victim's accessed page.

These findings indicate that—under our controlled experimental design—neither PMU counters nor timing measurements reveal reliable page-index information at either 4KB or 2MB granularity. Accordingly, we treat page-level identification as a low-signal task in this work and direct the main analysis toward key-class leakage, where measurable signal exists.

5.4. Runtime/Overhead Snapshot

We measured per-run execution cost using the `cpu-cycles` PMU counter (recorded as `ev00` in the CSV traces). For each (mode, key) run we aggregated the `ev00` readings across the 25 repetitions and then averaged across all evaluated runs to produce a single representative cycle count per configuration (2MB vs. 4KB). Relative difference was computed as

$$\Delta_{\%} = \frac{\overline{\text{cycles}}_{\text{4KB}} - \overline{\text{cycles}}_{\text{2MB}}}{\overline{\text{cycles}}_{\text{4KB}}} \times 100\%.$$

Using this metric the 2MB huge-page configuration required on average $\approx 11\%$ fewer CPU cycles per victim run than the 4KB baseline (i.e., $\Delta_{\%} \approx 11\%$ in favor of 2MB).

Two practical notes: (1) these numbers come from the standardized PMU counter (`cpu-cycles`) and therefore reflect processor-cycle work rather than high-level wall-clock time; (2) we report trimmed/averaged statistics (removing extreme outliers) to reduce influence from rare system perturbations. In informal checks the cycle-based speedup was consistent with small improvements in wall-clock runtimes, but we conservatively present the PMU-derived figure here because it has the highest-resolution and least affected by OS scheduling jitter.

6. Discussion & Conclusion

6.1. Performance Gain Without Additional Leakage

Our results demonstrate that using 2MB huge pages can improve runtime efficiency without amplifying measurable side-channel leakage. Across all cryptographic workloads, the 2MB configuration consistently achieved a mean **11% reduction in CPU cycles** (measured by the `cpu-cycles`

 doi:10.20944/preprints202511.1567.v1

PMU counter) relative to the 4KB baseline. This speedup is consistent with the expected effect of reducing translation overhead—fewer TLB misses and page-table walks when the entire 2MB region is covered by a single page translation entry. Importantly, this performance improvement does *not* come at the cost of increased leakage: key-classification accuracies between 4KB and 2MB mappings differ only slightly (typically within a few percentage points), and page-index identification remains at random-chance levels in both cases.

6.2. Why Huge Pages Do Not Increase Leakage

Although huge pages alter the hardware's memory-translation behavior, they primarily reduce noise rather than exposing new microarchitectural side channels. In our setup, the victim always accesses a single 4KB page per run inside the 2MB mapping, so address-translation structures (e.g., TLBs) remain stable between repetitions. Since our PMU features focus on general microarchitectural events—cycles, cache references/misses, and branch activity—rather than raw TLB-specific events, the overall leakage surface does not expand under huge-page mappings. The resulting classifier performance indicates that huge pages neither obscure nor amplify the extractable information about secret-dependent operations.

6.3. Rationale and Motivation of the Cryptographic Access Pattern Model

Real cryptographic routines—such as AES, RSA, or ECC—rarely operate on a single contiguous buffer. Instead, they perform sequences of reads from multiple memory pages to fetch keys, lookup tables, or plaintext blocks, followed by computations on intermediate states, and finally writes of ciphertexts or authentication tags. These read–compute–write cycles constitute the dominant sources of side-channel leakage in deployed cryptographic libraries, as each stage interacts differently with caches, TLBs, and execution pipelines.

In our experimental framework, we abstract the full cryptographic process into a controlled and reproducible model that concentrates on the translation-sensitive components of execution within a shared memory-mapped region. Specifically, we emulate only the *plaintext read* and *ciphertext write* phases, which directly invoke page translations and memory traffic—making them the primary contributors to TLB-driven leakage under different page sizes (4KB vs. 2MB). The PMU event counters (e.g., `cpu-cycles`, `cache-misses`, `branch-misses`) simultaneously capture aggregate microarchitectural effects from the entire execution, including arithmetic and control-flow variations during the intermediate computation phase. Consequently, the PMU-based measurements captures additional, computation-driven leakage beyond translation effects. In contrast, the timing-based TLB leakage is primarily influenced by page size and is limited to shared-file read and write activities, and thus represents a conservative *lower bound* on the total side-channel leakage of a full cryptographic system.

While our model abstracts the full cryptographic process, it provides an architecture-agnostic and analytically tractable benchmark that faithfully mirrors the key structural phases of real cryptographic operations—reads, computation, and writes. This controlled design enables fine-grained analysis of how page granularity influences both performance and leakage, while maintaining full experimental reproducibility. It thus serves as a clean framework for isolating and studying page-size–induced side-channel effects in modern cryptographic systems.

6.4. Security–Performance Balance

These findings suggest that, at least for user-space software performing repeated cryptographic or inference-like workloads, enabling 2MB huge pages can yield a tangible runtime gain *without* sacrificing observable side-channel resilience at the PMU level. Intuitively, fewer PTEs (Page Table Entries) with 2MB page size increase the probability for an attacker page to collide with a victim page, thereby leaking victim memory usage information. However, since the number of PTEs for 2MB page size reduces significantly compared to a 4KB page size, the number of bits of information leaked is also reduced manifold. While we do not claim that huge pages universally eliminate leakage (especially for attacks explicitly targeting translation structures), in our controlled attacker–victim

 doi:10.20944/preprints202511.1567.v1

model the huge-page configuration preserved comparable security while offering measurable efficiency improvements.

6.5. Broader Implications

From a systems perspective, this highlights a favorable trade-off: huge pages enhance throughput and determinism—benefiting both high-performance and privacy-sensitive workloads—without creating new leakage vectors (detectable via standard unprivileged PMU events). Future work could explore whether similar conclusions hold under temporal (sequence-based) analysis or on newer microarchitectures with extended PMU/TLB event coverage.

Author Contributions: Conceptualization: X.L.; methodology: X.L.; formal analysis: X.L.; writing—original draft: X.L.; writing—review and editing: A.T.; project administration: A.T.; funding acquisition: A.T. All of the authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author(s).

Conflicts of Interest: The funders had no role in the design of this study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Li, X.; Tyagi, A. Cross-world covert channel on arm trustzone through pmu. *Sensors* **2022**, *22*, 7354.
2. Xu, Y.; Qi, W.; Lin, Z. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In Proceedings of the IEEE Symposium on Security and Privacy, 2015, pp. 640–656.
3. Zhu, W. Exploring superpage promotion policies for efficient address translation. Master's thesis, Rice University, 2019.
4. Yarom, Y.; Falkner, K. Flush+ Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 719–732.
5. Gruss, D.; Maurice, C.; Mangard, S. Flush+ Flush: A Fast and Stealthy Cache Attack. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2016, pp. 279–299.
6. 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* **2020**, *63*, 93–101.
7. 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* **2020**, *63*, 46–56.
8. Xiao, Y.; Zhang, X.; Zhang, Y.; Teodorescu, R. One bit flips, one cloud flops:{Cross-VM} row hammer attacks and privilege escalation. In Proceedings of the 25th USENIX security symposium (USENIX Security 16), 2016, pp. 19–35.
9. Bepary, M.K.; Basu, A.; Mohammad, S.; Hassan, R.; Farahmandi, F.; Tehranipoor, M. SPY-PMU: Side-Channel Profiling of Your Performance Monitoring Unit to Leak Remote User Activity. *Cryptology ePrint Archive* **2025**.
10. Whyman, E.K.; Somers, H.L. Evaluation metrics for a translation memory system. *Software: Practice and Experience* **1999**, *29*, 1265–1284.
11. Wang, W.; Chen, G.; Pan, X.; Zhang, Y.; Wang, X.; Bindschaedler, V.; Tang, H.; Gunter, C.A. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2421–2434.
12. Wei, J.; Zhang, Y.; Zhou, Z.; Li, Z.; Al Faruque, M.A. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2020, pp. 125–137.
13. Hua, W.; Zhang, Z.; Suh, G.E. Reverse engineering convolutional neural networks through side-channel information leaks. In Proceedings of the Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

 doi:10.20944/preprints202511.1567.v1

14. Shokri, R.; Stronati, M.; Song, C.; Shmatikov, V. Membership inference attacks against machine learning models. In Proceedings of the 2017 IEEE symposium on security and privacy (SP). IEEE, 2017, pp. 3–18.
15. Zhang, Y.; Jia, R.; Pei, H.; Wang, W.; Li, B.; Song, D. The secret revealer: Generative model-inversion attacks against deep neural networks. In Proceedings of the Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 253–261.
16. Engler, D.R.; Kaashoek, M.F.; O'Toole Jr, J. Exokernel: An operating system architecture for application-level resource management. *ACM SIGOPS Operating Systems Review* **1995**, *29*, 251–266.
17. Gupta, A.; Faraboschi, P.; Gioachin, F.; Kale, L.V.; Kaufmann, R.; Lee, B.S.; March, V.; Milojicic, D.; Suen, C.H. Evaluating and improving the performance and scheduling of HPC applications in cloud. *IEEE Transactions on Cloud Computing* **2014**, *4*, 307–321.
18. Michailidis, T.; Delis, A.; Roussopoulos, M. Mega: Overcoming traditional problems with os huge page management. In Proceedings of the Proceedings of the 12th ACM International Conference on Systems and Storage, 2019, pp. 121–131.
19. Osvik, D.A.; Shamir, A.; Tromer, E. Cache attacks and countermeasures: the case of AES. In Proceedings of the Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology, Berlin, Heidelberg, 2006; CT-RSA'06, p. 1–20. https://doi.org/10.1007/11605805_1.
20. Wang, W.; Chen, G.; Pan, X.; Zhang, Y.; Wang, X.; Bindschaedler, V.; Tang, H.; Gunter, C.A. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2017; CCS '17, p. 2421–2434. https://doi.org/10.1145/3133956.3134038.
21. Wang, A.; Chen, B.; Wang, Y.; Fletcher, C.W.; Genkin, D.; Kohlbrenner, D.; Paccagnella, R. Peek-a-Walk: Leaking Secrets via Page Walk Side Channels. In Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP). IEEE, 2025, pp. 3534–3548.
22. Gao, T.; Huo, X.; Liu, H.; Gao, H. Wide neural networks as gaussian processes: Lessons from deep equilibrium models. *Advances in Neural Information Processing Systems* **2023**, *36*, 54918–54951.
23. Javed, Y.; Rajabi, N. Multi-layer perceptron artificial neural network based IoT botnet traffic classification. In Proceedings of the Proceedings of the Future Technologies Conference. Springer, 2019, pp. 973–984.
24. Gao, T.; Sun, S.; Liu, H.; Gao, H. Exploring the Impact of Activation Functions in Training Neural ODEs. In Proceedings of the The Thirteenth International Conference on Learning Representations, 2025.