Differential Privacy for Secure Cache Partitioning: Promises, Limits, and Fixes

Prerna Priyadarshini, Abhijat Bharadwaj and Biswabandan Panda
Indian Institute of Technology Bombay

***Abstract*—The shared last-level cache (LLC) enables cross-core contention-based attacks. While LLC randomization mitigates conflict-based attacks, cache partitioning prevents all attack classes. Recent secure partitioning techniques dynamically allocate LLC space and claim security against cache occupancy attacks. We show that state-of-the-art LLC usage-based dynamic partitioning remains vulnerable. Focusing on Secure and Scalable Cache Partitioning (SCALE), which applies differential privacy–based noise to LLC way allocations, we demonstrate that allocations can be denoised over time to construct a reliable LLC occupancy channel. The key insight that drives the occupancy channel is that differential privacy protects individual allocations; it does not hide long-term statistics such as the mean, median, and standard deviation. We showcase a novel LLC occupancy attack that does application fingerprint exploiting statistical properties. Finally, we propose lightweight fixes that mitigate this leakage.**

***Index Terms*—Dynamic cache partitioning, differential privacy, application-fingerprinting**

I. Introduction

Modern processors employ multi-level caches, with the shared last-level cache (LLC) being vulnerable to cross-core side and covert channels. LLC attacks broadly fall into three categories: shared memory-based [3], conflict-based [4], and cache occupancy-based [5]. The first two rely on cache set–level activity, whereas occupancy attacks exploit variations in an application's total cache footprint over time for fingerprinting [5]–[7]. LLC randomization and partitioning are widely used defenses. Randomization complicates eviction-set construction and mitigates conflict-based attacks but does not effectively prevent occupancy attacks, except for heavy-weight designs such as Sasscache [8], which incur significant performance overhead [9]. In contrast, partitioning techniques isolate cache space across security domains and are regarded as a comprehensive defense against contention-based attacks.

Differential privacy (DP) has emerged as a widely trusted framework for limiting information leakage in security and cryptography-driven systems. Inspired by its strong theoretical guarantees, recent dynamic LLC partitioning techniques have incorporated privacy-driven randomness into their allocation mechanisms to provide principled security assurances while preserving performance. Secure and Scalable Cache Partitioning (SCALE) [2] is one such technique that integrates differential privacy–motivated design elements into dynamic cache allocation. SCALE provides dynamic and secure LLC partitioning, without performance loss. In this work, we select SCALE as a representative differential privacy–inspired dynamic partitioning policy to assess how strong and practical the resulting security guarantees are. SCALE is an LLC usage-driven, way-based partitioning scheme built on Utility-Based Cache Partitioning (UCP) [1], which monitors application utilities and dynamically assigns LLC ways to minimize total LLC misses at periodic reconfiguration intervals. Since deterministic UCP allocations can expose occupancy patterns, SCALE augments UCP with multiple layers of controlled randomness, such as (i) Laplace-distributed noise, (ii) randomizes the *allocation change rate* (the extent to which allocations can change), and (iii) randomizes the *reconfiguration interval length* (timing of allocation changes). It leverages ε-differential privacy to provide information-theoretic guarantees, where noise parameters, LLC capacity settings, and ε define the protected range. Figure 1 shows LLC allocation with SCALE.

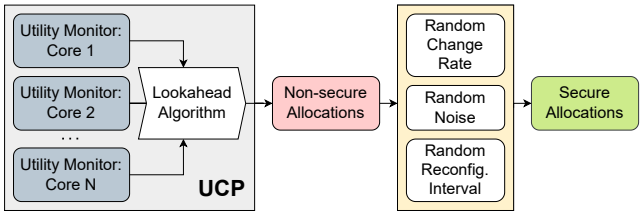

Fig. 1: Secure and dynamic LLC way allocation with SCALE

II. The Promises of Differential Privacy

ε-DP is a framework that aims to protect the individual privacy of dataset elements from an adversary using a differentially private mechanism, where ε is referred to as the privacy parameter or the privacy budget. For applications with strict privacy requirements, smaller values of ε are recommended. For a dataset x and a query on the dataset $q(x)$, a Laplace mechanism [11], [12] for ε-DP works by appending the query results with an additive zero-mean Laplacian noise. SCALE uses the Laplace mechanism of ε-DP to provide information-theoretic security guarantees. The UCP_{alloc} values are treated as a confidential dataset, while SCALE serves as the differentially private algorithm, using the Laplace mechanism to produce random and secure LLC allocations.

The key idea behind SCALE is using noise to secure the allocation policy while leveraging the benefits of UCP. It uses sampled Utility Monitors (UMONs) [1] to observe running applications over a time interval and estimate the utility of different cache sizes across applications. The LLC ways are then allocated to the applications to maximize net utility. Let us refer to this allocation as UCP_{alloc}. To secure the allocations, SCALE uses a multi-layered randomization

defense as per Equation 1, where $A(t)$ is the amount of LLC ways allocated to an application at t^{th} reconfiguration interval and $\Delta_{alloc}(t) = UCP_{alloc}(t+1) - A(t)$ is the difference between UCP's newly calculated way allocation and SCALE's current way allocation. Noise, change rate, random rounding, and a randomized length of reconfiguration intervals together serve as the defense measures. The noise is motivated by the ε-differential privacy.

$$\begin{aligned} A(t) = randRound(A(t-1)+ \\ randRound(\Delta_{alloc}(t-1) \times changeRate) + noise) \end{aligned} \tag{1}$$

We treat each LLC allocation, $A(t)$, as a discrete random variable because of the randomness in SCALE's design.

Random rounding. The $randRound$ function in Equation 1 randomly rounds a value up or down with equal probability. For all inputs $X \in \mathbb{Z}$, $randRound(X) = X$, otherwise, $\forall X \in \mathbb{R} \setminus \mathbb{Z}$, $randRound(X) = \lfloor X \rfloor + c$, where c is either 0 or 1 at random and $\lfloor . \rfloor$ is the floor or round-down function. Since the probability that $X \in \mathbb{Z}$ is negligible,

$$randRound(X) = \lfloor X \rfloor + c \quad \text{(with probability 1)} \tag{2}$$

effectively transforming Equation 1 to:

$$\begin{aligned} A(t) = A(t-1) + \lfloor \Delta_{alloc}(t-1) \cdot changeRate \rfloor + \\ \lfloor noise \rfloor + c_1 + c_2 \end{aligned} \tag{3}$$

where c_1 and c_2 are either 0 or 1 independently and uniformly at random. The expected value (denoted by $\mathbb{E}[.]$) of the sum $c_1 + c_2$ is 1. Moreover, random variable $\forall X \in \mathbb{R}$, the expected value of flooring is bounded as $\mathbb{E}[X] - 1 < \mathbb{E}[\lfloor X \rfloor] \leq \mathbb{E}[X]$.

Noise. A zero-mean Laplace distribution (refer Equation 4) is used for the noise. Even though the noise values are enforced to be non-zero, note that $\mathbb{E}[noise]$ remains zero.

$$Lap(x|0,b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right) \tag{4}$$

Change rate. The $changeRate$ parameter is sampled from a uniform distribution $\mathcal{U}(a_1, a_2)$, where $0 \leq a_1 < a_2 \leq 1$. Thus, $\mathbb{E}[changeRate] = (a_1 + a_2)/2$. If UCP_{alloc} remains unchanged across two consecutive intervals, $changeRate$ is set to 0. Notably, when SCALE's allocations approach the UCP_{alloc}, the effect of $changeRate$ diminishes.

Random reconfiguration interval. UCP reallocates partitions at a configurable yet fixed interval, which is suggested to be once every five million clock cycles [1]. SCALE, on the other hand, randomizes the duration between two re-allocations to increase the unpredictability. The extent of the randomness, however, is left to be configured by the user.

The Combined Effect. Given $A(t-1)$ and $\Delta_{alloc}(t-1)$, from the above discussion, the expected allocation at time t, where t is a valid reconfiguration instance, is:

$$\mathbb{E}[A(t)] = A(t-1) + \left(\frac{a_1 + a_2}{2}\right) \Delta_{alloc}(t-1) + \delta \tag{5}$$

Here, δ is a small error term such that $-1 < \delta \leq 1$. *Equation 5 highlights an important property of SCALE's allocation policy. Despite multiple layers of randomness, the expected allocations still follow a predictable behavior.*

III. The Limits of Differential Privacy

Claim-I of SCALE. For allocation changes within the *protected range*, SCALE provides strong security guarantees. A strong security guarantee specifically refers to a small ε in the differential privacy mechanism, meaning that individual allocations changes within the protected range (Δf) are indistinguishable.

Claim-II of SCALE. The probability of observing the same allocation fingerprint across repeated executions approaches zero as the number of reconfiguration periods increases.

We argue that the randomness introduced by differential privacy principles indeed makes it difficult to deduce LLC allocation within a given interval. However, if the occupancy attacker observes LLC allocations over a long interval, it can still make fingerprinting feasible through LLC occupancy, even in the presence of SCALE operating within its *protected* range. Figure 2 validates our argument across different SCALE configurations for a SPEC CPU2017 benchmark `mcf_1554B` co-running with `xalancbmk_10B` on a 2-core, 16-way, 4MB LLC system. The cumulative average of SCALE's allocation converges close to UCP's across all noise levels and change-rate configurations. While higher noise increases short-term fluctuations in LLC allocations and lower change rates slow down convergence, neither prevents long-term convergence, and the average behavior remains recoverable.

Our observations. With ε-DP, a single instance of noise drawn from the Laplace distribution for a single LLC space allocation is unpredictable. However, the statistics like *mean* of a sufficiently large number of noise samples converge to the *mean* of the distribution (the law of large numbers), revealing the average LLC allocation. Although DP preserves the privacy of individual LLC allocations in isolation, it does not guarantee privacy for the statistical summaries, such as mean [11].

A. Application fingerprinting attack on an LLC with SCALE

We use the ChampSim [10] simulator to simulate SCALE. We simulate a two-core system with Sunny Cove–like memory hierarchy with 2MB non-inclusive LLC bank per core, with 16 ways. L1D and L2 are 48KB and 512KB, with 8 ways and 12 ways, respectively. Our processor is an out-of-order 4GHz core with a 352-entry ROB with a retire width of four.

Threat model. Our threat model follows Cook et al. [7] and Shusterman et al. [5]. An attacker is co-located with a victim and executes on the same physical host, and leverages shared microarchitectural resources like LLC, and the attacker can monitor shared resources to infer the victim's activity. In our case, the attacker and victim run on separate cores sharing a SCALE-partitioned LLC. We evaluate the attack in a closed-world setup, where the attacker knows all the applications and has already profiled them. The goal of the attacker is to distinguish among applications that are co-located with the attacker by observing the LLC that is partitioned using SCALE. The practical implication of this attack is similar to a website fingerprinting attack, where the attacker distinguishes the websites visited by the victim.

Occupancy attack on LLC. The attacker performs a sweep-counting occupancy attack [5], [6] by repeatedly accessing a buffer equal to the LLC size and counting completed

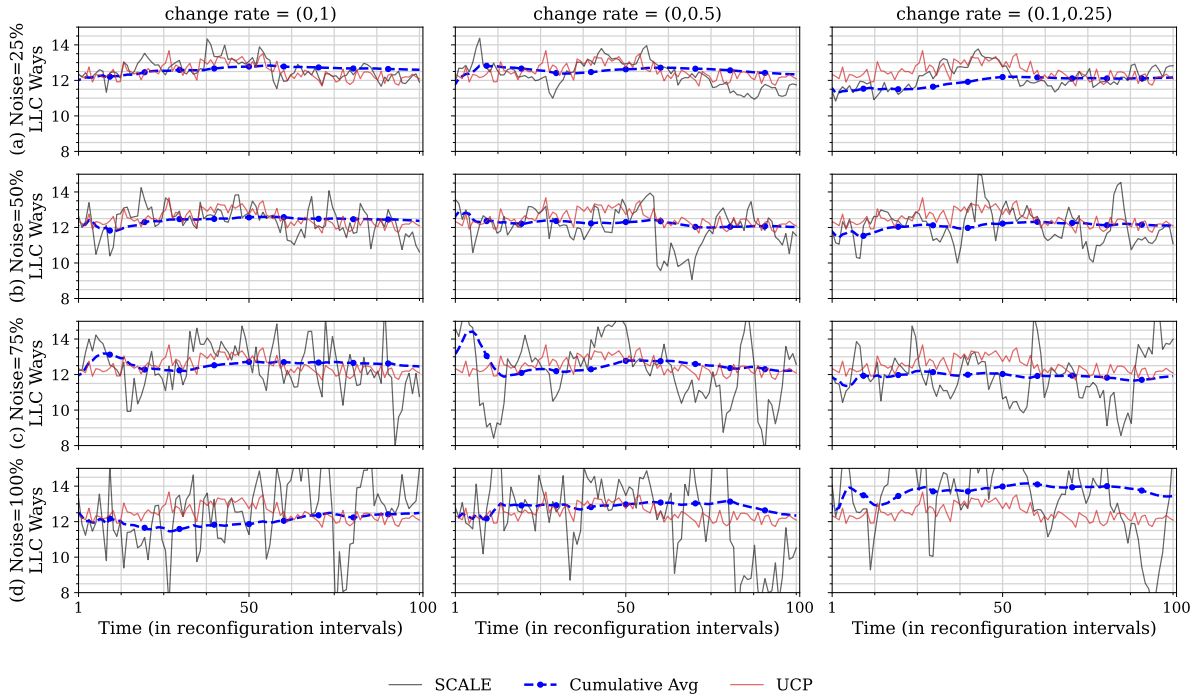

Fig. 2: LLC ways allocation with SCALE and UCP over 100 reconfiguration intervals on a 4MB 16-way LLC for `mcf_1554B` running with `xalancbmk_10B` under different SCALE noise levels: (a) 25%, (b) 50%, (c) 75%, and (d) 100%.

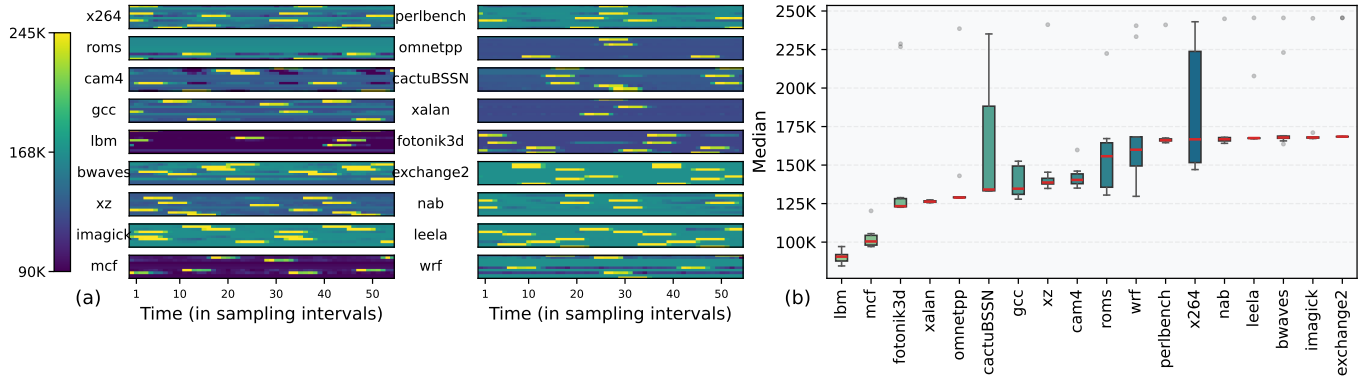

Fig. 3: Attacker's occupancy when co-running with SPEC CPU2017 benchmarks across runs over 55 sampling intervals. (a) Memorygram with each row as a run (y-axis omitted). (b) Boxplot of median counter values after median filtering; boxes show the interquartile range, the center line marks the median, and whiskers denote the non-outlier range.

accesses within each interval (5M cycles). Note that moving ahead, we call this interval the *sampling interval*. Larger victim cache occupancy increases attacker misses, reducing the counter; smaller occupancy increases hits and the counter. The attacker records these values over time as a *memorygram* and classifies SPEC CPU2017 benchmarks using the resulting traces. We demonstrate that an attacker can fingerprint co-running applications by observing only their cache occupancy. The attacker constructs memorygrams, as shown for SPEC CPU2017 benchmarks (Figure 3(a)). Due to the randomness introduced by SCALE, instantaneous occupancy patterns of many applications overlap in the memorygram. However, their long-term statistical characteristics form distinct signatures, enabling reliable differentiation, as shown in Figure 3(b).

Classification accuracy. We use a decision tree to classify applications, which learns interpretable threshold-based rules. Before extracting features, we first apply a median filter, calculating the median after five reconfiguration intervals for each memorygram. This step helps remove short-term noise and fluctuations caused by sharp noise effects, while preserving the underlying trend of cache occupancy over time. After filtering, each run is summarized using a compact set of statistical features that we feed into the classifier. The mean captures the average cache occupancy, giving a general sense of how much cache space an application typically uses. The median provides a more robust central value that is less affected by any remaining noise or occasional spikes. The standard deviation measures how much the occupancy varies over time, helping differentiate between stable workloads and those with more dynamic behavior. In addition, the minimum and maximum

values define the range of occupancy, indicating the low and high levels of allocation during execution. The 25th and 75th percentiles further capture intervals where most of the occupancy values lie.

Figure 4 shows the effect of noise, reconfiguration interval range, and sampling intervals used by the attacker on the accuracy. The accuracy increases with sampling intervals and saturates, with the saturation point shifting to larger values as the reconfiguration range increases, indicating higher sample requirements. Higher noise levels reduce achievable accuracy and further delay saturation. The highlighted plane shows the SCALE with a reconfiguration interval varying between 4M and 40M cycles, providing a zoomed behavior of the required sampling intervals. Increasing reconfiguration randomness and allocation randomness does not prevent the attack, but only increases the required number of samples. Large reconfiguration ranges are undesirable from a performance perspective, as they drive the system toward near-static partitioning. The vertical dashed lines denote the saturation points corresponding to each noise level, while the circled markers indicate the accuracy attained at those points.

Sensitivity studies. We perform the attack with different sizes of L1D, L2, LLC, and in the presence of two recent hardware prefetchers, IPCP [14] and Berti [15]. In all the cases, we see an average classification accuracy of more than 90%. As a pathological case, the classification accuracy drops significantly if we use an impractical 100 MB LLC per core.

Takeaway. SCALE's security argument relies on the low probability of reproducing identical allocation sequences across executions due to multiple sources of randomness. However, this reasoning implicitly assumes that successful fingerprinting requires exact sequence matching. In contrast, our attack leverages statistical properties of allocations, which remain stable across executions. Even though the probability of observing identical allocation patterns approaches zero, the probability of inferring the same statistical fingerprint remains high. SCALE overlooks this fact.

IV. The Fixes

Memoized per-interval secret bias. At the start of each reconfiguration interval, a secret offset $\mu_e \sim \mathcal{U}(-\delta, \delta)$ is sampled and kept fixed within that interval. The allocation is computed as:

$$A(t) = U(t) + \eta(t), \quad \eta(t) \sim \text{Lap}(\mu_e, b).$$

This introduces a shift in the observed allocations, so that the same application exhibits different allocations across runs. As a result, simple averaging-based inference converges to $\bar{U} + \mu_e$ instead of $\bar{U}$. While the per-interval shift disrupts averaging-based inference, it does not affect features that are invariant to such shifts, such as distributional spread or relative behavior.

Allocation quantization. To further reduce fingerprint distinguishability, allocations are quantized into coarse buckets. This limits the granularity of observable values and maps nearby allocations to the same level, thereby reducing the resolution of the signal. Together, the per-interval shift reduces consistency across runs, while quantization reduces the precision of observations. This combination reduces classification accuracy to below 20% with four buckets and a per-interval offset sampled as $\mu_e \sim \mathcal{U}(-3, 3)$ for a 16-way LLC.

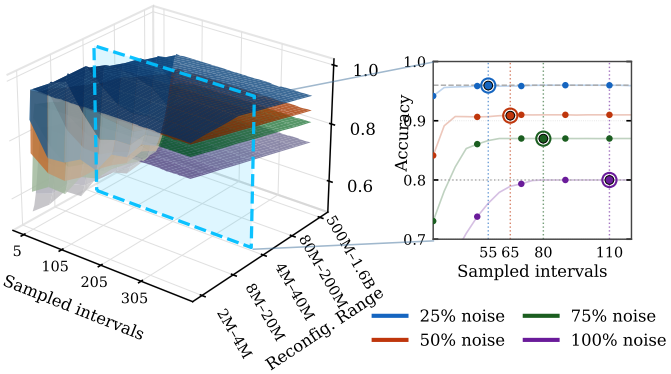

Fig. 4: Classification accuracy with different reconfiguration interval ranges (in cycles), noise, and sampled intervals.

V. Conclusion

In this paper, we demonstrated a cache–occupancy–based attack against SCALE that uses differential privacy. We provide both empirical evidence and a theoretical analysis of SCALE's design in examining how each randomization component affects security and how residual information leakage persists. Our results show that differential privacy-based cache partitioning is insufficient to prevent long-term occupancy inference. We also proposed a few fixes that can ensure the security guarantees of SCALE, even for an occupancy attack that looks at long-term statistics for fingerprinting attacks.

Acknowledgement

We would like to thank members of the CASPER research group, especially Shubham Roy. We also thank the authors of SCALE for clarifying our queries and providing feedback on our work. This work was supported by the Trust Lab grant 2023 at IIT Bombay.

References

[1] Qureshi *et al.*, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance Runtime Mechanism to Partition Shared Caches," in *MICRO*, 2006.
[2] Holtryd *et al.*, "SCALE: Secure and Scalable Cache Partitioning," in *HOST*, 2023.
[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security*, 2014.
[4] Y. A. Younis *et al.*, "A New Prime and Probe Cache Side-Channel Attack for Cloud Computing," in *IEEE CIT*, 2015.
[5] A. Shusterman *et al.*, "Website Fingerprinting Through the Cache Occupancy Channel and Its Real-World Practicality," in *IEEE TDSC*, 2020.
[6] A. Shusterman *et al.*, "Robust Website Fingerprinting Through the Cache Occupancy Channel," in *USENIX Security*, 2019.
[7] J. Cook *et al.*, "There's Always a Bigger Fish: A Clarifying Analysis of a Machine-Learning-Assisted Side-Channel Attack," *ISCA*, 2022.
[8] J. Giner *et al.*, "Scatter and Split Securely: Defeating Cache Contention and Occupancy Attacks," in *IEEE S&P*, 2023.
[9] A. Bhatla *et al.*, "SoK: So, You Think You Know All About Secure Randomized Caches?," in *USENIX Security*, 2025.
[10] ChampSim, "ChampSim Simulator," 2020. [Online]. Available: http://github.com/ChampSim/ChampSim
[11] S. Vadhan, "The Complexity of Differential Privacy," Springer, 2017.
[12] C. Dwork *et al.*, "Calibrating Noise to Sensitivity in Private Data Analysis," in *TCC*, 2006.
[13] A. Bhatla *et al.*, "The Maya Cache: A Storage-Efficient and Secure Fully-Associative LLC," in *ISCA*, 2024.
[14] S. Pakalapati and B. Panda, "Bouquet of Instruction Pointers: Instruction Pointer Classifier-Based Spatial Hardware Prefetching," in *ISCA*, 2020.
[15] A. Navarro-Torres *et al.*, "Berti: An Accurate Local-Delta Data Prefetcher," in *MICRO*, 2022.