Entropy-Coupled CLFLUSH for Secure Cache Flushing Against Timing-Based Attacks

Samiksha Verma
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
samiksha@cse.iitb.ac.in

Virendra Singh
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
singhv@iitb.ac.in

I. Introduction

Modern processors expose explicit cache control instructions such as `CLFLUSH`, `CLFLUSHOPT`, and `CLWB` to enable software-driven cache management. These instructions are essential in legitimate domains such as direct memory access (DMA), memory-mapped I/O, and persistent memory, where explicit coherence and ordering control are required.

However, these same instructions have been exploited to mount cache-based side-channel attacks. In particular, the *Flush+Flush* attack [1] (illustrated in Fig. 1) leverages the deterministic latency of `CLFLUSH` to infer whether a cache line was recently accessed by a victim. The attacker flushes a shared cache line, waits, and then flushes it again while measuring the latency of the second flush. A lower latency indicates that the victim accessed the line and brought it into the cache, revealing fine-grained memory activity.

Such deterministic timing behavior also enables attacks such as Flush+Reload and RowHammer-style fault injections. By observing flush completion latency, attackers can infer sensitive access patterns even across cores or virtual machines, making `CLFLUSH` a powerful primitive for timing-based information leakage.

A. Motivation

Despite the known risks, disabling `CLFLUSH` entirely is impractical. It remains essential to a wide range of system software stacks, including kernel subsystems, persistent memory management, and device drivers. Therefore, a practical defense must preserve the correctness and ordering guarantees of `CLFLUSH` while removing its exploitable determinism.

B. Limitations of State-of-the-Art

Existing mitigation strategies exhibit significant drawbacks:

- **Instruction Restriction:** Microcode-level filtering or privilege elevation for `CLFLUSH` (as proposed in some OS kernels) blocks side channels but breaks backward compatibility for user-space libraries that depend on direct cache management.
- **Timer Obfuscation:** Methods such as TimeCache [3] and CacheNoise introduce delay jitter to reduce timing precision. However, since these act externally to the flush instruction, the instruction's latency remains statistically distinguishable under repeated probing.
- **Hardware Randomization:** Defenses like InvisiSpec [4] and SafeFlush [5] integrate internal cache randomization or speculative shielding. While effective, they incur high area and latency overheads, often exceeding 10%.

The key limitation across these approaches is that they treat timing noise as a uniform or global source of randomness. In contrast, an attacker only needs local repeatability to exploit `CLFLUSH`—thus global jitter is insufficient. A defense must instead bind unpredictability to each invocation of the instruction itself.

C. Key Insight

We identify that the vulnerability of `CLFLUSH` arises from its deterministic microarchitectural completion path rather than from its eviction semantics. Our key insight is that coupling each flush with a unique, local, and transient entropy value can retain its correctness while eliminating predictable timing signatures. By introducing bounded randomness within the flush pipeline—without affecting coherence guarantees—we can effectively decouple the attacker's observation window from the cache's internal behavior.

D. Contributions

This work introduces **Entropy-Coupled CLFLUSH**, a novel redesign of the cache flush mechanism with the following contributions:

- A new entropy-coupled execution model for `CLFLUSH` that introduces microarchitectural randomness directly at the cache controller level.
- A formal timing-independent flush completion model ensuring that the entropy injection does not affect cache coherence or ordering semantics.

II. Threat Model and Overview

We consider an attacker with unprivileged user-space access who can execute `CLFLUSH` on shared physical pages and measure latency using high-resolution timers such as `rdtsc`. The attacker's objective is to differentiate cache-resident and non-resident data through timing analysis or to repeatedly hammer a DRAM row via flush-induced activation patterns.

The defender (ECF) seeks to ensure that the flush latency and cache traversal behavior observed by any software agent are statistically indistinguishable regardless of cache state.

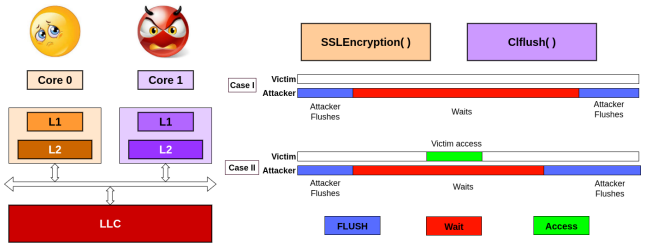

Fig. 1. Flush+Flush attack. In Case II, the second `FLUSH` takes longer due to a cache hit.

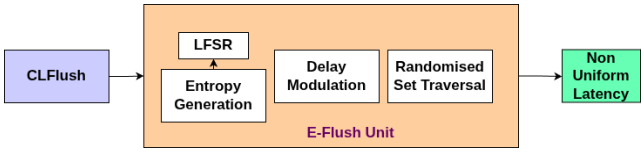

Fig. 2. Flowchart of the proposed Entropy-Coupled CLFLUSH (ECF) mechanism. The EGU generates entropy per invocation, which modulates both delay and cache traversal order to eliminate deterministic timing channels.

ECF guarantees that no measurable correlation exists between flush completion time and target cacheline status.

III. Design of Entropy-Coupled CLFLUSH (ECF)

ECF extends the cache controller logic with a lightweight entropy generation unit (EGU) based on a linear feedback shift register (LFSR) seeded by dynamic microarchitectural signals such as bus contention, prefetch queue occupancy, and thermal counters. This entropy is consumed per invocation of the flush instruction (refer Fig. 2).

The ECF pipeline operates in three stages:

1) **Entropy Generation:** When a `CLFLUSH` is decoded, the EGU produces an entropy token E_t unique to that instruction instance.
2) **Delay Modulation:** A small, bounded delay $\delta(E_t)$ (typically 2–15 cycles) is added before issuing the flush acknowledgment, ensuring the instruction's completion time varies pseudo-randomly.
3) **Randomized Set Traversal:** For each targeted set, the cache controller traverses ways in a permuted order defined by E_t, breaking deterministic access patterns.

The delay is tightly bounded to preserve memory ordering guarantees. All coherence and write-back semantics remain identical to the baseline design.

IV. Security Analysis

We model the adversary's observation of a flush invocation as a random variable A. Let $S \in \{0,1\}$ represent whether a target cache line is resident ($S = 1$) or not ($S = 0$). In our scheme each flush produces a randomized latency Δ and possibly randomized scope U, both drawn from distributions independent of S. The observed time is then

$$A = T_{\text{base}} + \Delta + N,$$

where T_{base} is the deterministic flush component and N is measurement noise.

Assumption 1. Δ and U are independent of S.

Assumption 2. The adversary only obtains A; internal entropy state is hidden.

Under these assumptions the flush observation satisfies the Markov chain

$$S \;\to\; (\Delta, U) \;\to\; A,$$

and by the data-processing inequality we get

$$I(S;A) \;\le\; I(S;\Delta,U) = 0.$$

Thus the mutual information between the secret state and the adversary's observation is zero: the adversary gains no information about S from a single flush.

From a hypothesis-testing perspective, let $P_{A|S=0}$ and $P_{A|S=1}$ denote the distributions of A conditioned on $S = 0$ or 1. Since Δ and U are identically distributed and independent of S, it follows $P_{A|S=0} = P_{A|S=1}$. The total variation distance between the two distributions is zero, implying the adversary's distinguishing advantage is also zero.

Hence even after repeated sampling the adversary cannot distinguish the two states better than random guessing. In other words, the flush instruction no longer provides a usable timing oracle.

Discussion. The analysis depends crucially on hiding the entropy generator state and ensuring that Δ is genuinely independent of cache residency. If either is violated (for example through leakage of RNG state or side channels correlating $\Delta \to S$), then the guarantee degrades. Our design mitigates these risk vectors by drawing entropy from microarchitectural sources not influenced by single-line residency, and by isolating the entropy path from user-accessible state.

V. Conclusion and Ongoing Work

Entropy-Coupled CLFLUSH addresses a long-standing microarchitectural vulnerability by embedding entropy directly into cache flush execution. Unlike prior noise-based or restrictive approaches, ECF preserves the full semantics and compatibility of the instruction while eliminating deterministic timing behavior. Our ongoing work focuses on hardware prototyping, adaptive entropy calibration based on workload sensitivity, and integration into emerging NVRAM-aware systems. This study underscores the potential of entropy-driven hardware design as a general security primitive for microarchitectural hardening.

References

[1] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack," in *Proc. DIMVA*, 2016.
[2] Y. Yarom and K. Falkner, "Flush+Reload: A High Resolution, Low Noise Cache Side-Channel Attack," in *Proc. USENIX Security*, 2014.
[3] K. Falkner et al, "TimeCache: Temporal Randomization of Cache Access Latency for Side-Channel Mitigation," in *Proc. ICCD*, 2018.
[4] K. Weisse *et al.*, "InvisiSpec: Making speculative execution invisible in the cache hierarchy," in *Proc. MICRO*, 2019.
[5] J. Wang and Y. Li, "SafeFlush: Mitigating flush-based side channels through randomized cache line invalidation," in *Proc. HPCA*, 2020.