Entropy defenses against side channels - a small literature survey

ved@rivosinc.com

Randomized Cache Architectures

Caches used as side channel - including with transient execution attacks such as spectre and meltdown.

Randomized cache architectures replace predictable address-to-index mappings with deterministic but random-looking mappings.

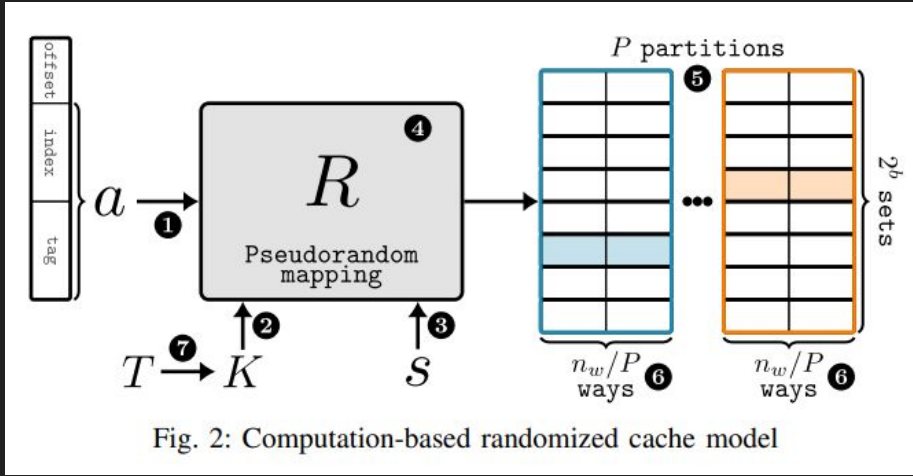


Fig. 2: Computation-based randomized cache model

1. Memory address `a` is the input
2. Key K is the designs entropy
3. S is the security domain separator
4. Rk(a,s) -> is a pseudo-random mapping
5. Cache divided into P partitions
6. One of the partitions selected randomly
7. T is the rekeying period

Examples:

- CEASAR/CEASAR-S
- SCATTER CACHE
- TIME-SECURE CACHE (TSC)

Randomized cache architectures - security analysis

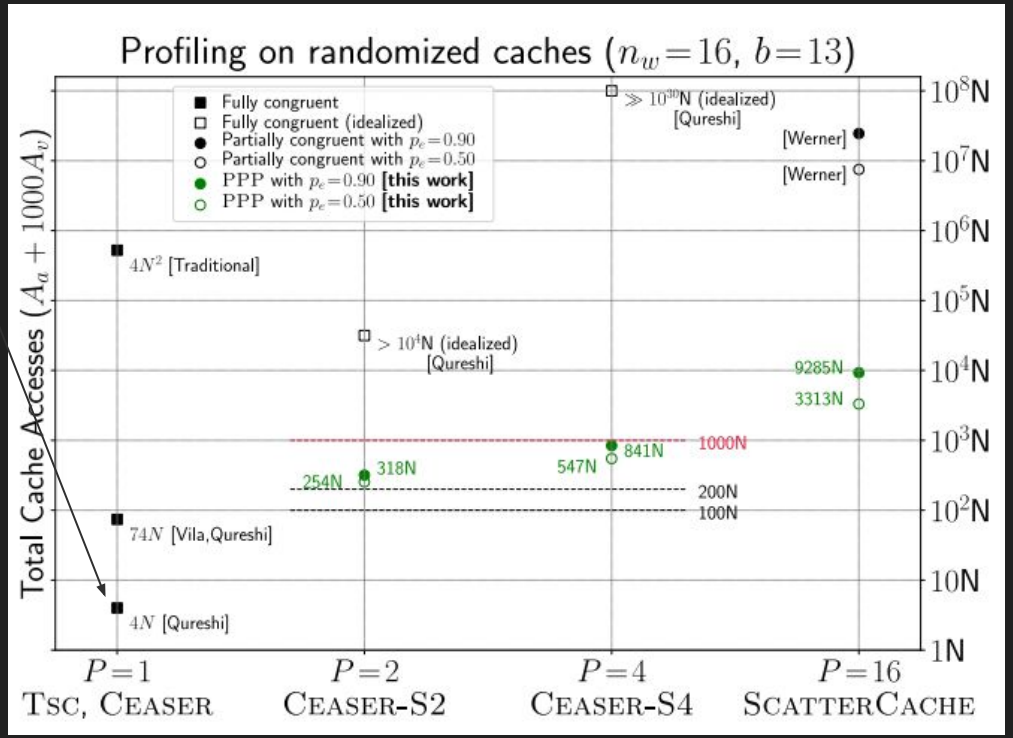
Systematic analysis of randomization-based protected cache architectures - Purnal et. al.

Single partition caches are not much better than conventional caches - to be effective need extremely frequency re-keying

Prime-prune-probe technique on CEASAR-S2 has average complexity of ~320N => rekeying needed more frequently 100/200N

CEASAR low latency block cipher (LLBC) lack a non-linear function leading to shortcut attacks - collision in one partition implies collision in all

SCATTERCACHE using Qarma resilient to shortcuts but the PPP reduced the complexity significantly.



Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It

Presented at IEEE 21 symposium on security and privacy

Key contributions:

- Measure the remap period by LLC evictions rather than accesses because the probability of successfully finding an eviction set is closely related to the number of evictions allowed between remaps.
- Further reduce the period to stop attackers from finding even small partially congruent eviction sets.
- Adopte ZCache-like [24] multi-step relocation to minimize the number of cache blocks evicted during the remap process.
- Promote the use of CEASER (randomized set-associative cache) rather than skewed caches because CEASER introduces less overhual to the existing cache structure than skewed caches and it can be made secure enough.
- A simple attack detection mechanism to further strengthen CEASER.

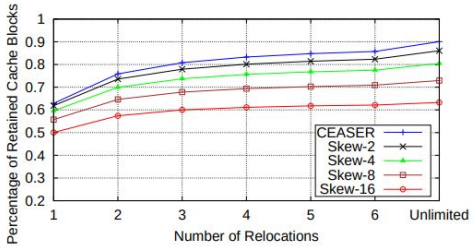
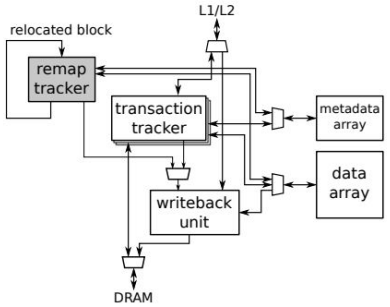


Fig. 12. The percentage of cache blocks retained during remapping by applying limited number of relocations. The maximum retaining percentage is achieved when 'infinite' trials of relocation are applied until a remapped cache block is found as the replacement (and evicted). Each result is averaged from 100 independent experiments.



Phantom Cache

Paper proposes Securing an LLC with remapping-free randomized mapping.

Propose a localized randomization technique to bound randomized mapping of a memory address within only a limited number of cache sets (propose 8 - for a 16M/16-way cache)

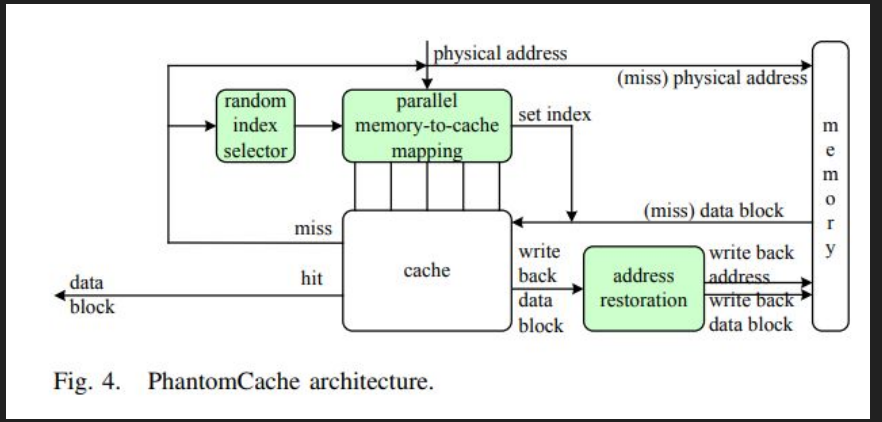


Fig. 4. PhantomCache architecture.

TABLE II. TIME FOR EVICTION SET MINIMIZATION UNDER PHANTOMCACHE USING r CANDIDATE CACHE SETS.

r	$\mathcal{O}(\|E\|^2)$ algorithm	$\mathcal{O}(\|E\|)$ algorithm
2	13 days	0.7 seconds
4	584 years	6.5 days
6	170,682 years	7.8 years
8	9,583,986 years	584 years

Securing branch predictors with two-levels of encryption

[Paper]

- PC encrypted with per-context key for index randomization
- Data encrypted for further security
- Rekeying
- Inspired by Caesar

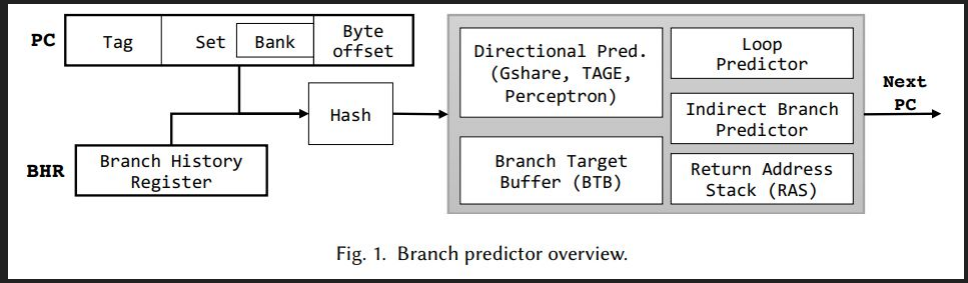


Fig. 1. Branch predictor overview.

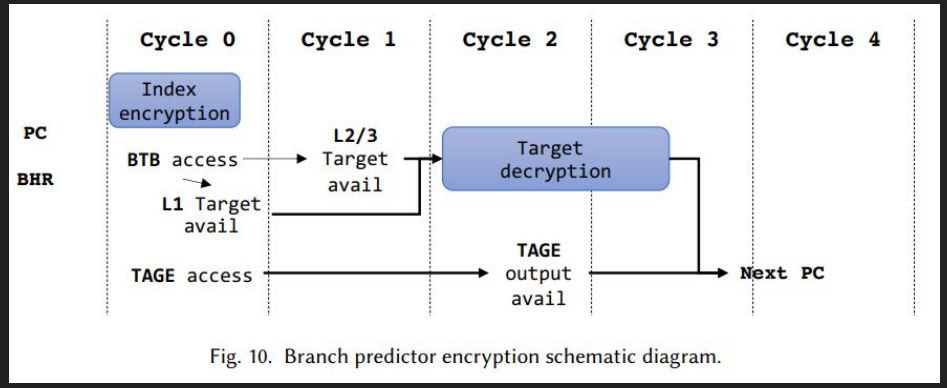


Fig. 10. Branch predictor encryption schematic diagram.

Security Analysis of two-level branch predictor encryption

Table 2. Security analysis (✓: protected, ✗: not protected).

Scheme	Category 1	Category 2	Category 3 (Spectre v1)	Category 3 (Spectre v2)
No encryption	✗	✗	✗	✗
Shared-key encryption [31]	✓	✗	✗	✗
Index encryption (different keys)	✓	✓	1024	4096
BSUP	✓	✓	16384	✓

Category 1 To infer control low graph. Category 2 similar to 1 but targets outcome of a specific entry. Category 3 specter v1/v2 style attacks.

With target encryption v2 can be mitigated per author

V1 - even with frequent rekeying - prone to brute force attacks (e.g. encrypting 3-bit saturating counter still has a small number of outcomes to force)

Randomly scaling frequency and prefetchers

Mitigating Cache-Based Side-Channel Attacks through Randomization: A Comprehensive System and Architecture Level Analysis

Carefully adapting the processor frequency and prefetchers operation and adding proper level of noise to the attackers' cache observations to protect the critical information from being leaked.

Results indicate that the concurrent randomization of frequency and prefetchers can significantly prevent cache based side-channel attacks with no need for a new cache design.

Proposed randomization and adaptation methodology outperforms the state-of-the-art solutions in terms of the performance and execution time by reducing the performance overhead from 32.66% to nearly 20%.

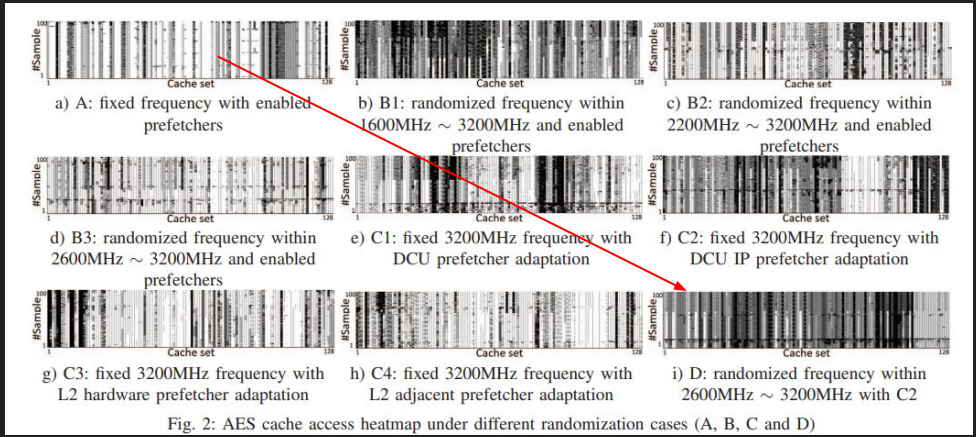


Fig. 2: AES cache access heatmap under different randomization cases (A, B, C and D)

Morpheus: A Vulnerability-Tolerant Secure Architecture Based on Ensembles of Moving Target Defenses with Churn

[Paper]

Morpheus, combines two powerful protections: ensembles of moving target defenses and churn.

Ensembles of moving target defenses randomize key program values (e.g., relocating pointers and encrypting code and pointers) which forces attackers to extensively probe the system prior to an attack.

To ensure attack probes fail, the architecture incorporates churn to transparently re-randomize program values underneath the running system.

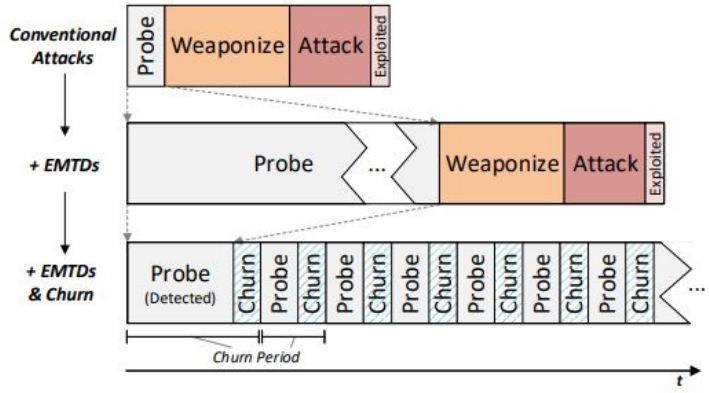
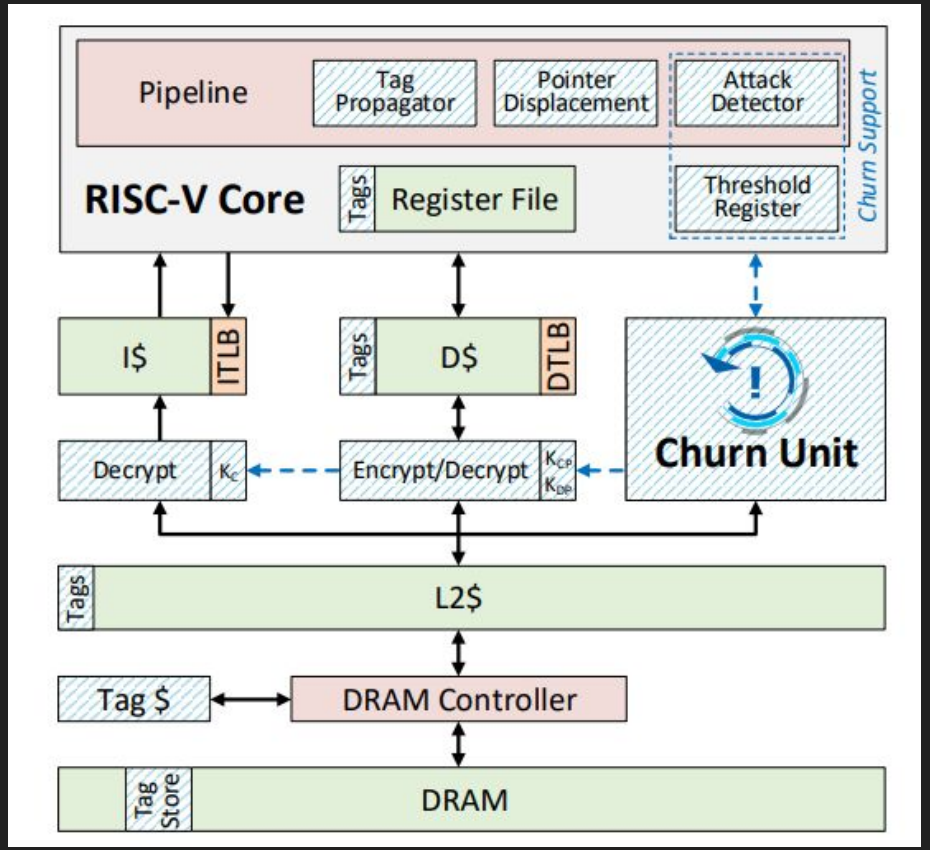


Figure 2. Thwarting Attacks via EMTDs with Churn. Exploits progress with the following phases: *probe*, *weaponize*, and *attack*. Ensembles of moving target defenses (EMTDs) increase uncertainty in key values needed to weaponize attacks, thus, probe times increase significantly. To ensure the patient attacker does not complete their probes, churn re-randomizes key program values at regular intervals. An attack detector senses when probe-like activity is occurring and reduces the churn period to strengthen defenses.

Morpheus

Key defenses:

- Domain Tagging
- Pointer displacement
- Domain Encryption



Morpheus

Key defenses:

- Domain Tagging
  - Tracks Code, Code pointer, data pointer, and data
  - Use LLVM extensions to identify code and pointers
  - Churn rules trigger churn
- Pointer displacement
- Domain Encryption
- Churning

Table 2. Attack Detector Logic. ABORT rules monitor particularly grievous operations and trigger an exception that terminates the program. CHURN rules detect undefined behavior that may be indicative of an ongoing attack. In response to these violations, the attack detector initiates a churn cycle. Tags are defined as follows: C = Code, CP = Code Ptr, D = Data, DP = Data Ptr.

	<OP>	Check Condition	Rule
ABORT	Execute	Insn.tag != C	Only execute C
	ANY	R1/R2.tag == C	No C in the pipeline
	JAL(R)	R1.tag != CP	Jump target must be CP
	LD/ST	R1.tag != DP	Address must be a DP
CHURN	COMPARE	R1.tag != R2.tag	No inter-domain compares
	ANY (not JAL(R))	R1.tag == CP	CP arithmetic suspicious
	ANY (not LD/ST)	R2.tag == DP	DP arithmetic suspicious, except add/sub D
	ANY	Overflow Occurs	Overflows are undefined
	SHIFT	Shift > RegWidth	Invalid shift is undefined

Morpheus

Key defenses:

- Domain Tagging
- Pointer displacement
  - create two randomly displaced address spaces (DASC and DASD) above the virtual address space (VAS).
  - All pointers as displaced for its lifetime, including pointers in the registers, caches, and memory.
  - Pointer computations (e.g., p + const) proceed on displaced pointers as normal without being impacted by the displacement.
  - 60-bit random displacement,
- Domain Encryption
- Churning

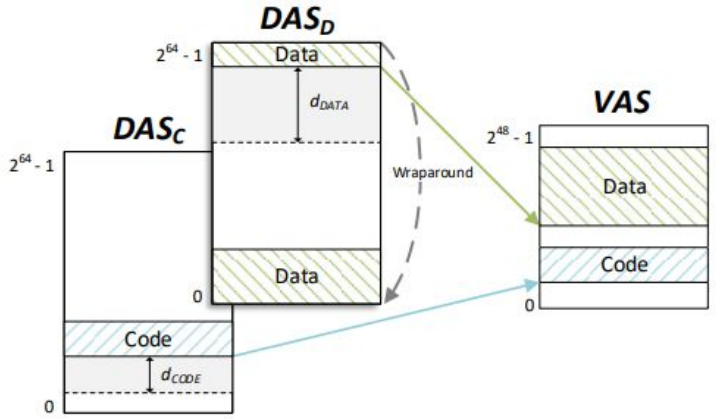


Figure 4. Pointer Displacement. Pointer displacement creates two displaced address spaces: DAS_D and DAS_C, each shifted by a random displacement of d_{DATA} and d_{CODE}, respectively. By having two displaced address spaces, it is possible for each address space to utilize any displacement without colliding with other objects in the other domain, resulting in higher entropy for the code and data segments.

Morpheus

Key defenses:

- Domain Tagging
- Pointer displacement
- Domain Encryption
  - randomizes the representation of code, code pointers, and data pointers in memory using a strong cipher.
  - QARMA7-64-σ1, which encrypts 64-bit blocks with a 128-bit key in 16 rounds
  - Separate key set per process/VM
- Churning

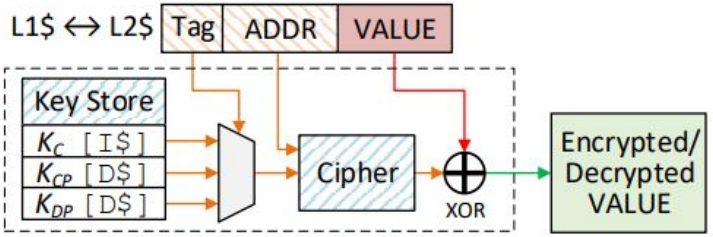


Figure 5. Domain Encryption Defense. Morpheus' encryption makes use of per-domain keys, denoted as K_C, K_{CP}, and K_{DP} for *code*, *code pointer*, and *data pointer*, respectively. These keys are selected by a value's domain tag and used to protect it in memory. Non-pointer data is unencrypted.

Morpheus

Key defenses:

- Domain Tagging
- Pointer displacement
- Domain Encryption
- Churning
  - Under live execution
  - Churn unit uses the tags in tag store
  - 50ms churn period
  - Two keys active simultaneously

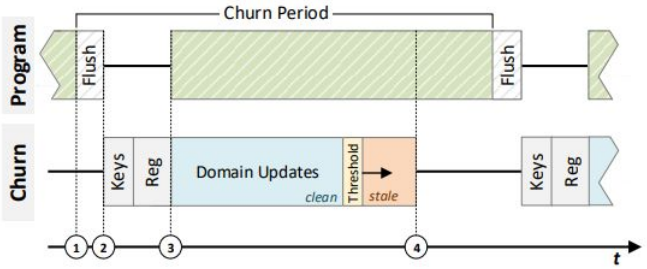
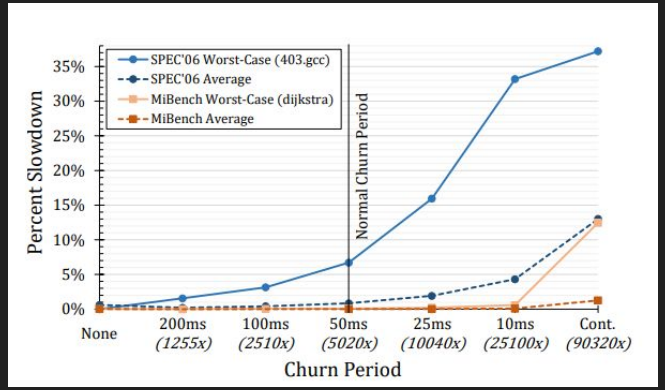


Figure 6. Churning During Execution. The churn cycle starts with ① a pipeline flush and ② new key generation and register updates. Next, ③ a threshold register is used to coordinate updating values in memory under the new displacements and encryption keys. When ④ churn completes, all domains have been updated, effectively disposing of any information the attacker may have acquired previously.



Ghost Thread: Effective User-Space Cache Side Channel Protection

Ghost Thread serves as a flexible library that protects accesses to the sensitive memory region by injecting random accesses to the same region. By doing this, an adversary now has to distinguish random behavior added by Ghost Thread from actual program behavior to launch a successful attack

Algorithm	Time w/o GT	Time w/ RT GT
AES [36]	0.064s	1E17 years
AES (Cross-VM) [18]	4.5s	1E19 years
RSA [47]	30 min	3170 years

Table 5: A summary of how long it takes to collect sufficient number of samples on our test machine to launch the corresponding cache attacks without and with Ghost Thread.