Journal of Systems Architecture 135 (2023) 102805

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

TPPD: Targeted Pseudo Partitioning based Defence for cross-core covert channel attacks

Jaspinder Kaur [a,*], Shirshendu Das [b]

[a] Department of CSE, Indian Institute of Technology Ropar, Punjab, 140001, India
[b] Department of CSE, Indian Institute of Technology Hyderabad, Telangana, 502285, India

ARTICLE INFO

Keywords:
Cache security
Timing channel attacks
Cache partitioning
Covert Channel Attack (CCA)
Last level cache (LLC)

ABSTRACT

Contemporary computing employs cache hierarchy to fill the speed gap between processors and main memories. In order to optimise system performance, Last Level Caches (LLC) are shared among all the cores. Cache sharing has made them an attractive surface for cross-core timing channel attacks. In these attacks, an attacker running on another core can exploit the access timing of the victim process to infiltrate the secret information. One such attack is called a cross-core Covert Channel Attack (CCA). Timely detection and then prevention of cross-core CCA is critical for maintaining the integrity and security of users, especially in a shared computing environment. In this work, we have proposed an efficient cross-core CCA mitigation technique. We propose a way-wise cache partitioning on targeted sets, only for the processes suspected to be attackers. In this way, the performance impact on the entire LLC is minimised, and benign applications can utilise the LLC to its full capacity. We have used a cycle-accurate simulator (gem5) to analyse the performance of the proposed method and its security effectiveness. It has been successful in abolishing the cross-core covert timing channel attack with no significant performance impact on benign applications. It causes 23% less cache misses in comparison to existing partitioning based solutions while requiring ≈ 0.26% storage overhead.

1. Introduction

Modern multi-core processors have adopted a multi-level cache hierarchy to address the rising need for high-performance computing. Caches are smaller and faster memories deployed to bridge the speed gap between the processor and main memory. Most commercial processors available in the market are equipped with multiple levels of caches. The higher layers of caches are private to each core, whereas all CPU cores share the Last Level Cache (LLC).[1] Cache memories boost overall performance by allowing processors quicker access to data. While improving the system's overall performance, the LLCs have also become an attractive target surface for timing channel-based attacks [1] due to these reasons:

1. The significant time difference between a cache hit and miss provides an effective timing channel that is exploited to unveil the memory accesses of the targeted process [2,3].
2. Shared nature of LLC allows an attacker process to interfere in the cache occupancy of other processes. The attacker uses the timing channel to understand other processes' access patterns on the shared cache as described in Fig. 1. These accesses are further analysed to reveal the underlying secret [4].

Any remote process, sharing LLC with the victim process, can mount these attacks without requiring special privileges or shared address space [5]. The victim and attacker processes may run on separate cores, but they share the LLC space. Such attacks are called *cross-core attack*. These attacks become more threatening in shared computing environments such as the cloud, where multiple users from different security domains share the underlying LLC. Cache timing channel can be implemented in two forms: Side-Channel Attack (SCA) and Covert Channel Attack (CCA) [6,7]. The private keys of cryptography algorithms like AES [8], RSA [9], and ECDSA [10] have been the primary target of cache-based SCAs. In a CCA, two suspicious processes, spy and Trojan, communicate covertly by exploiting the cache timing channel [6]. The Trojan runs on a different core and knows some sensitive information that it needs to send to the spy. The attacker uses CCA when as no direct communication between Trojan and spy is possible because of the security restrictions.

* Corresponding author.
E-mail addresses: 2017csz0002@iitrpr.ac.in (J. Kaur), shirshendu@cse.iith.ac.in (S. Das).

[1] All the cache memories in this paper are considered as set-associative cache. We used the term "set" and "way" of a set-associative cache without a detailed explanation about them.

https://doi.org/10.1016/j.sysarc.2022.102805
Received 30 May 2022; Received in revised form 30 November 2022; Accepted 10 December 2022
Available online 19 December 2022

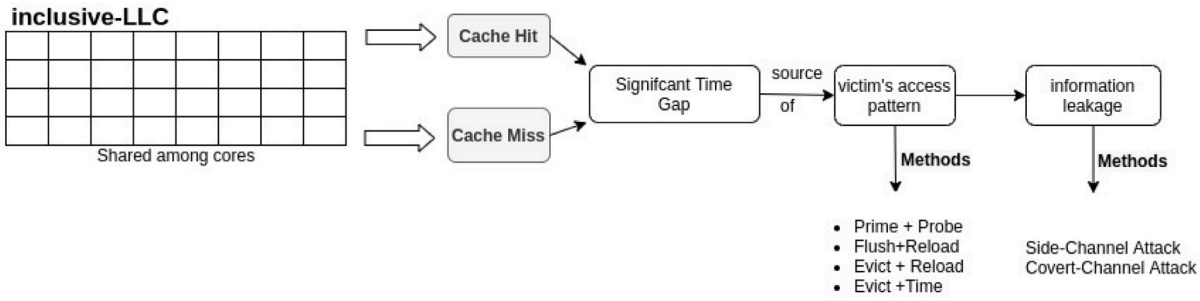


Fig. 1. The step-wise details of LLC based timing channel attacks.

From experiments, we have found that the attack works successfully on real-time systems with processors Intel Core i9-10885H, i7-9700, and i7-7700. It is crucial to detect and prevent these attacks effectively in order to preserve the confidentiality and integrity of the computer systems [11,12]. This paper is based on proposing countermeasures for preventing cross-core CCA attacks. Section 2.1 discusses CCA in more detail.

The CCA attack that is discussed in this paper is cross-core attack; based on the shared LLC used by the multi-core processors. Attack defence mechanisms based on not allowing cache sharing through LLC partitioning have been proposed previously [13,14]. Cache partitioning techniques were initially proposed for improving the performance of the system by fairly dividing the LLC among the cores (or applications) [15–17]. Most of them were not proposed keeping security in mind. These partitioning techniques divide the LLC either way-wise or set-wise. The way-wise partitioning techniques are more prevalent where the ways of a set-associative LLC are partitioned among the cores (or applications) [15]. The partition can be either static or dynamic. The static partitioning techniques cannot change the partition during the execution, while the dynamic partitioning techniques can change the partition based on the requirement of the running applications. Since cache partitioning can separate the LLC space used by Trojan and spy, the cache interference of two applications can be prevented. However, there are two major challenges in using cache partitioning for preventing CCA:

1. The static partition-based solutions suffer from severe performance degradation as cache cannot be utilised fairly based on the dynamic behaviour of the system [15].
2. Dynamic partitioning techniques utilise the cache more efficiently and hence improve performance. However, the dynamic partitioning technique itself can be exploited to mount CCA as proposed in [18]. In this work, the authors have shown that if the attacker knows the logic of partition change, it can misuse this and change the partition as per the attacker wish. Thus, a dynamic partitioning based countermeasure can restrict the attack but indirectly opens another option for CCA.

Hence for preventing CCA, a partitioning technique is required, in which the attacker cannot change partition size. Also, the technique should not degrade the performance of the overall system.

There are some existing countermeasures proposed based on the static partition where the technique applies to all the sets within the LLC. In this paper, we call this type of countermeasure as a non-targeted countermeasure. Such countermeasures cause higher performance degradation in the system. Our attack mitigation mechanism overcomes the drawbacks of the existing partitioning-based solutions. We do this by providing Targeted Pseudo Partitioning based Defence (TPPD). It is termed as "targeted" because the attack prevention mechanism is only applied to the cache sets suspected to be participating in CCA. The sets which participate in the timing channel attacks are termed as *targeted set*. The remaining cache sets can behave as before, thus minimising the effect of static partitioning on the performance. A practical attack detection technique is deployed that sends the information regarding suspicious processes (Trojan and spy) and cache sets involved. Attack detector based on conflict misses pattern [19] has been tested to identify the suspicious processes and sets involved.

The main contributions of this work are listed as follows:

1. We propose an effective attack defence mechanism, TPPD, that dismantles the cross-core CCA with an insignificant effect on system performance.
2. The proposed TPPD creates the pseudo partition only on the targeted set and is applied only for the Trojan and spy process. In this technique, not all blocks of the spy can be removed by Trojan and vice versa.
3. Effect of TPPD on the performance of PARSEC benchmark [20] is tested experimentally in gem5 simulator [21].
4. Experiments are conducted to test the effectiveness of TPPD in mitigating cross-core CCA.

The organisation of the paper is as follows. Section 2 describes the background and related work. Section 3 describes the threat model taken under consideration, and Section 4 discusses our proposed work in detail. The experimental analysis is shown in Section 5. Finally, Section 6 concludes the paper.

2. Background and related works

This section describes the background information required to understand the proposed attack defence mechanism. In a cache timing channel attack, the attacker process uses different attack methods to unveil the cache access pattern of the target process. In side-channel attacks, the target is an innocent victim process performing cache accesses as part of its underlying operation without any malign intentions. In a covert channel attack, there are two malign processes performing cache accesses with the intent of leaking secrets. An abstract overview of cache timing channel attacks is shown in Fig. 1.

2.1. Cross-core covert channel attacks

In a covert timing channel attack, two suspicious processes: spy and Trojan, are involved. Trojan has access to some critical information that it wants to transmit to spy, but this transmission is not allowed under the system security policy. Trojan uses a cache timing channel to leak this information to spy without being noticed. This type of attack is called Covert Channel Attack (CCA). The CCA is called cross-core when the spy and Trojan execute on two different cores. In this case, both Trojan and spy use the shared LLC to create a timing channel. Cache timing channel attack can be constructed using various attack techniques like Prime+Probe (P+P), Evict+Reload (E+R), Flush+Reload (F+R), and Evict+Time (E+T) [6]. These attack techniques, even though different, follow these basic three steps:

- Step 1: In the first step, the aim is to bring the shared LLC in a predictable state known to the attackers. It is done by either bringing some data in LLC or evicting it out by the spy.

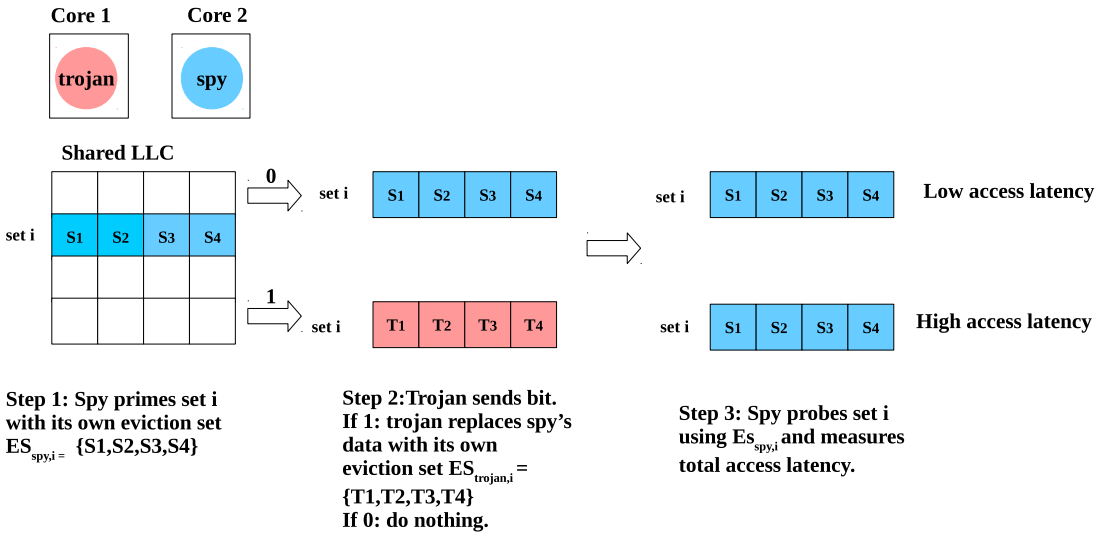


Fig. 2. Cross-Core covert channel attack using Prime+Probe technique.

- Step 2: In this step, the spy process sits idle, and the Trojan process performs conditional memory accesses based on the bit to transmit (bit 1 or 0).
- Step 3: In this step, the spy process observes the changes made in the earlier known state of the cache by the Trojan to detect the bit as 1 or 0.

Repeating the above mentioned steps, the Trojan can send multiple bits to the spy.

CCA using Prime+Probe (P+P) technique is described in Fig. 2. In this method, spy and Trojan do not require any shared address space. Both these processes have their own Eviction Set (ES) [22]. ES is the group of unique block addresses mapping to the same set. The number of addresses in each ES is either equal to or more than the associativity of the underlying cache. In the prime phase, the spy uses the addresses present in its ES to fill the targeted set with its own blocks. After priming the cache, spy waits for a Trojan to send the bit. If Trojan wants to transmit bit 1, it replaces all of spy's blocks from the targeted set with its own blocks (from Trojan's ES). For bit 0, it does not do anything; thus, spy's blocks stay in the set. After this, in the probe phase, spy accesses the addresses from its ES again. In this phase, spy faces high latency in case Trojan has removed its blocks from the set; otherwise, less latency. Based on this latency, the spy unveils the cache access pattern of Trojan and the secret bit transmitted. The rest of the discussions of this paper assume Prime+Probe (P+P) based CCA.

The Prime+Probe attack discussed in Fig. 2 can be slightly modified to make the attack possible in non-inclusive caches also [23]. The idea here is to make sure that the blocks requested in the prime phase reside only in the LLC after the prime phase. This can be achieved by using some additional block requests at the end of the prime phase. Such actions remove all the actual blocks of eviction set from L1 and place them in LLC. The same policy will be followed by the Trojan. Hence the timing difference will be created only based on the hits and misses of the LLC. Since the timing difference, in this case, is not expecting a hit in L1, the attack works perfectly for non-inclusive caches.

2.2. Existing attack mitigation techniques

As discussed in Section 2.1, cache sharing is the critical pre-requirement of cross-core cache timing channel attacks. Prohibiting cache sharing across processes or different security domains is an effective solution. However, it is not feasible as the impact of a non-shared cache will lead to under-utilisation of cache capacity thus, impacting system performance significantly. Various works with different cache partitioning techniques that try to maintain cache utilisation have been proposed in recent years. Partitioning Locked cache (PLcache) [24] allows compiler and programmer to mark security critical lines, and these will be locked in the cache. In this architecture, each cache line is augmented with a process id and a bit dictating whether that line is locked or not. A locked line of a process cannot evict a locked line of a different process, while no unlocked line is allowed to replace a locked line. Thus, disabling inter-process and intra-process cache interference has been identified as the root cause of cache access-based attacks. NonMonopolizable (NoMo) [25] cache allocated v number of ways of each set to an active thread. The allowed range for v is 1 to A/M, where A is the associativity of the cache, and M is the maximum number of threads allowed per processing core. Data of one thread cannot evict reserved lines of another thread; thus, an attacker cannot know the victim's cache access.

NoMo and PLCache offer easy-to-implement static partition solutions, but the static partitioning can lead to not utilising the cache to its full capacity. Security Dynamic Cache Partitioning(SecDCP) cache [26] addresses this issue by proposing a dynamic cache partition. SecDCP allocates a number of ways to different security classes based on their requirement. However, dynamic partitioning in itself can be exploited to create cache covert timing channel attack as described in [18]. A secure dynamic partitioning in terms of SCA is proposed in [13]. It proposes a secure dynamic cache partitioning cache called FairSDP, where partitioning size is determined by the cache usage of non-critical processes, thus not revealing the requirement of security-critical processes. However, this does not work against CCA as it involves two suspicious processes, and there is no security-critical process.

All the mitigation techniques discussed above apply some strict cache partitioning policy across all sets for all processes. Hence these techniques reduce the performance of the LLC as well as the entire system. Also, most of the existing countermeasures are non-targeted, as the mechanism is applied to all the cache sets and processes regardless of their participation in the attack. A targeted countermeasure can be less expensive as the necessary actions can be applied only in the sets currently under attack. An alternative solution to prevent cross-core CCA is the recently proposed randomised LLCs [27,28]. In these works, the block-to-set mapping of the set-associative cache changes after an epoch. However, these techniques suffer significant performance degradation because of the remapping [29]. The number of write-backs required to remap is high. Hence in this work, we have been motivated to propose a targeted countermeasure for the cross-core CCA.

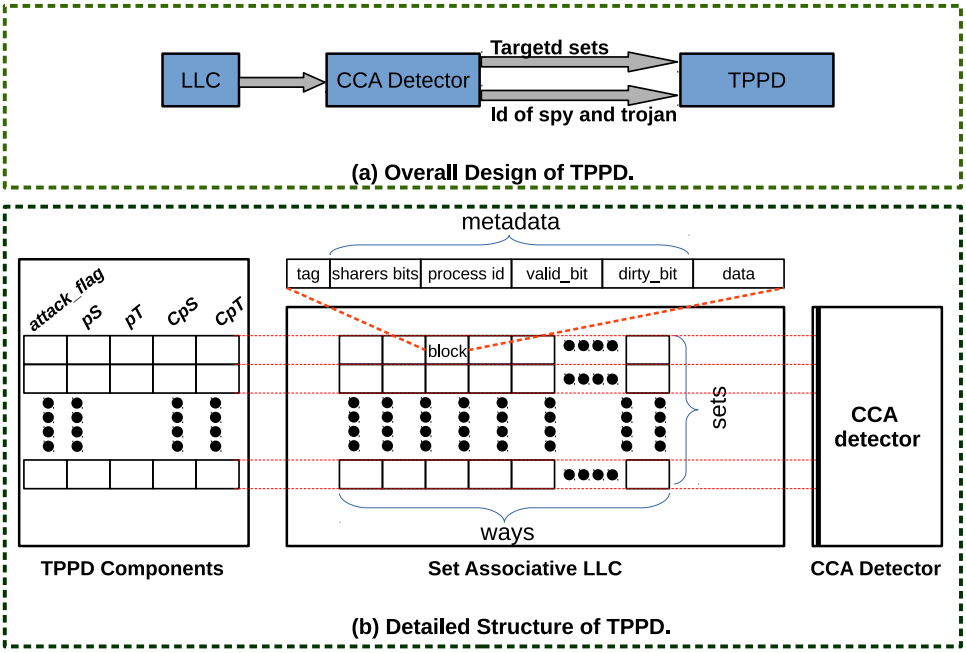


Fig. 3. Overall design of TPPD and its structure.

3. Threat model

In this paper, we have assumed a chip multiprocessor (CMP) with two levels of cache memories. Each core in the CMP has its own private L1 cache, and all the cores share a common L2 cache as LLC. The LLC is inclusive, and all the cache memories are set-associative. The cores, L1 caches, and LLC are connected with some on-chip interconnects. The cross-core CCA attack discussed in this paper runs spy and Trojan on two different cores. These two attacking processes (spy and Trojan) are not using any shared address space. Hence no cache blocks can be shared among these two processes, and the eviction sets (c.f. Section 2.1) of both processes are also different. However, both spy and Trojan share the LLC, and they can evict each other blocks from the LLC. The eviction set of both spy and Trojan maps to the same targeted set. We propose TPPD to mitigate cross-core CCA. This attack can be based on Prime+Probe [6] or Evict+Time [6] methods, where spy and Trojan processes rely on replacing each other's block to transmit bits. Trojan has a piece of secret information that spy cannot access directly due to underlying system security policies. Trojan transmits this information to spy through LLC based covert communication channel (details of CCA is already discussed in Section 2.1). Both Trojan and spy can successfully create their eviction sets as they are aware of the virtual address to LLC set mapping. An undisclosed set mapping of set-associative cache can also be calculated as discussed in [30]. Both attacker processes are unprivileged processes that do not have any kind of special privilege.

4. Our proposal

The most straightforward approach to preventing attacks is killing the suspected attacker process. However, no attack detection mechanisms are free from false detection. The killing of the innocent process can cause severe performance as these processes needs to be restarted from the beginning. An ideal attack prevention mechanism should effectively obfuscate the covert communication between spy and Trojan without compromising system performance. We propose a localised attack prevention mechanism with minimal impact on benign processes to fulfil this requirement. Our proposed targeted defence mechanism, TPPD, works against LLC based covert channel attacks (cross-core CCA) by creating way-wise pseudo partitioning between the suspicious process pair (spy and Trojan). This partition technique used in TPPD cannot be changed by the attacker process as per its requirement. Also, the partition is only applied on the sets suspected to participate in covert channel communication. The purpose of using the term "pseudo" is discussed later in this section. TPPD decreases the cache interference between the Trojan and spy on the targeted sets, thus disturbing the access latency pattern observed by the spy to interpret the bit sent. TPPD receives information regarding suspicious process ids (spy and Trojan) and the targeted sets from a CCA detector module; then applies the proposed pseudo-partitioning on these sets for the suspicious processes. The CCA detection module is discussed in Section 4.1. The overall design of TPPD is shown in Fig. 3(a). In this section, we describe TPPD in detail, along with how it can be implemented with minimal modification in architectural design.

4.1. Cross-core covert channel attack detector

TPPD works efficiently by applying attack mitigation mechanisms only for suspicious process pairs on the LLC sets suspected to be used for cache covert channel attacks. The countermeasure proposed in TPPD depends on an efficient CCA detection mechanism. A cross-core CCA detection mechanism based on conflict misses pattern is proposed in [19]. It is based on the observation that set under attack have a higher number of conflict misses than other sets during the attack. These misses form a ping-pong pattern because spy and Trojan processes evict each other's data regularly to communicate covertly using cache timing channel attacks. In [19] a two-step detection mechanism is proposed which, firstly, filters out the sets with low conflict. In the second step, sets observing ping pong pattern of conflict misses between two processes are declared suspicious along with processes participating in this pattern. However, we observed that the behaviour of total cross-process misses per unit time is sufficient in order to identify sets and processes used for mounting CCA. The experimental observation for the same is shown in Fig. 4. The figure shows the cross-process conflict misses on an LLC set. The experiment is performed on different mixes of benign processes (Mix-x) from Parsec benchmarks (described in Table 1) and also on an attacker process pair (only-attack). The details about the experiment setup, benchmarks, and creating attack process pair are discussed in Section 5. From the figure, it can be observed that the attacker process pair suffers high cross-process conflict misses when

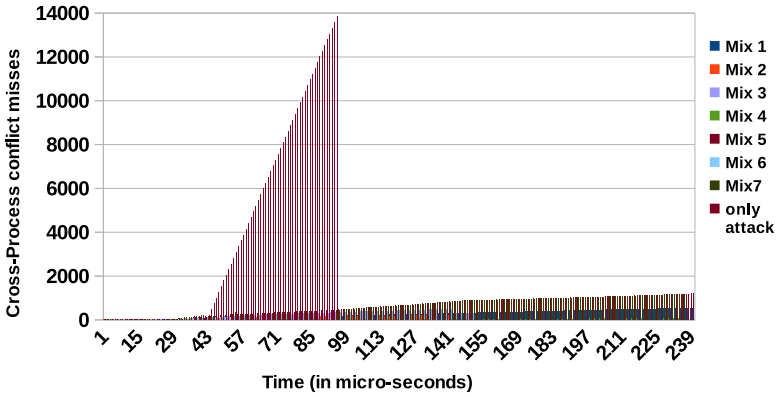


Fig. 4. Cross-process conflict misses per unit of time (.10 s).

Table 1
PARSEC benchmark mixes used.

Mixes	Benchmarks Involved
Mix 1	blacksholes + canneal
Mix 2	blacksholes + dedup
Mix 3	blacksholes + fluidanimate
Mix 4	blacksholes + freqmine
Mix 5	canneal + dedup
Mix 6	canneal + fluidanimate
Mix 7	freqmine + fluidanimate

compared to any mixes of benign benchmarks for the duration of the attack.

Initially spy and Trojan collude to decide on a target set and bit pattern for synchronisation. During synchronisation, Trojan sends this small bit pattern to spy for sync purposes using a covert channel attack on the targeted set. This pattern transmission generates more cross-process conflict misses in a smaller duration than any other benign benchmark mix. Thus, attacks are detected before actual suspicious transmission can take place. The cross-process conflict misses and rate of increase is significantly higher than any Innocent process pair as described in Fig. 4. Thus, setting an appropriate threshold was able to detect suspicious pairs without any false-positive in these experiments.

In this work, we have used the concept of a conflict-miss pattern-based CCA detector to detect the attack. Since our work is on detection based countermeasures, we have used the existing detection techniques. Fig. 3(a) shows that the outcome of the CCA detector is a pair of attacking processes (Trojan and spy) and the targeted sets. During our experimental analysis, the detector detects CCA successfully every time. The mechanism has some hardware overhead to record the cross-process conflict misses in LLC. However, the operations of the detector perform in the background and do not affect the critical path of execution.

4.2. Targeted pseudo partitioning based defence (TPPD)

When the CCA detector detects an attack, it sends information about the processes and sets involved in the CCA. TPPD enables pseudo partitioning on the suspected sets for the suspected process pair (Trojan and spy). Two non-overlapping way-wise partitions are created and assigned to Trojan and spy. The spy and Trojan are unable to replace each other's data if the block is present in this allocated partition. This restriction, however, exists solely for identified malicious pairs; other benign processes can access this cache set as before without any constraint. Because of this reason, the proposed partitioning is called pseudo as it is invisible to all processes except attackers. As our defence mechanism is targeted, it only affects the cache sets and the processes involved in CCA, thus not impacting the system's performance significantly.

The proposed TPPD can handle multiple CCA attacks on different cache sets simultaneously. However, in this section, we have assumed only one pair of attacker processes (spy and Trojan) and only one targeted set. For the rest of this section, we assume that the detector detects pS as a spy and pT as a Trojan. Here pS and pT are the process id of spy and Trojan, respectively. Also, we assume that the detected targeted set is s'. TPPD can be implemented on top of any replacement policy with minor modifications. A replacement policy has three major modules: (a) insertion, (b) promotion, and (c) eviction [16]. To implement TPPD, minimal changes are required in the eviction module of a replacement policy. TPPD expects two victim selection approaches on the targeted set s'. We call it as "dual victim policy".

(a) The default victim selection of a replacement policy. Let us represent the victim block selected through this method as $V(s')$, where s' is the targeted set.
(b) Victim block selection excluding a particular process p. Let us represent the victim block selected through this method as $V_x(s', p)$. Here the victim block is selected from the targeted set s', excluding the blocks owned by process p.

This dual victim policy can be implemented on most of the existing replacement policies [31,32] with minor modifications. The concept of TPPD, generic to any replacement policy (those can be modified for dual victim selection), is discussed next.

Consider that for a cache block b, $O(b)$ represents the owner process to which the block belongs. The dual eviction policy is only required for the targeted set s', detected by the CCA detector as discussed in Section 4.1. For other sets, $s; s \neq s'$, the victim block is always $V(s)$. Since the attacker can target any of the cache sets, the facility of dual victim selection must be present in all the sets. However, it is used only for the targeted set. To implement the dual eviction policy, we need to maintain two counters for each set $CsP(i)$ and $CsT(i)$ where i is the set number. $CsP(s')$ and $CsT(s')$ are used to count the number of spy and Trojan blocks respectively present in s'. When pT (Trojan) block needs to replace an existing block of pS (spy) from s', TPPD sets some additional conditions. If $O(V(s'))$ is pS (spy) and $CsP(s')$ is less than a threshold, pT cannot replace $V(s')$. In that case, the victim block is selected by $V_x(s', pS)$. This policy restricts the Trojan from removing all of the spy blocks from s'. A similar policy is also applied when a spy tries to evict a block of Trojan from s'. The ping-ponging pattern of conflict misses between the spy and Trojan is interrupted because the continuous eviction of each other's data is restricted. In this way, a spy cannot observe the cache access latency pattern to distinguish bit 0 or 1, resulting in noisy covert channel communication. The value of the threshold depends on the underlying replacement policy. In the rest of this paper, we use LRU to implement TPPD, and an appropriate threshold value to inhibit attack is also discussed.

4.2.1. Structure of LLC using TPPD

Fig. 3(b) shows the structure of the LLC using TPPD. The additional components required for TPPD are shown on the two sides of the set-associative LLC. These additional components are divided into two major parts: the CCA Detector and TPPD Components. For TPPD, each set maintains a tuple, $(attack\_flag, pS, pT, CsP, CsT)$. Here the attribute, $attack\_flag$ is a single bit of information that indicates whether the set is under attack or not. The other attributes are already defined in this section. The attributes excluding $attack\_flag$, are only required for a suspicious set. When the CCA detector detects a CCA attack, the corresponding attributes of the attacker set are updated in the TPPD component. As mentioned in Section 4.1 the CCA detection module is a cross-process conflict-miss pattern-based detector as proposed in [19].

4.2.2. Engagement and disengagement with dual victim policy

Since the CCA attack happens during execution, in the beginning, all the sets are non-suspicious. A set engages with dual victim policy only when it has been detected as a targeted set by the CCA detector. Once a process pair and a set s' have been detected as suspicious, one option is to continue with the proposed dual victim policy (in s') till the termination of these processes. However, the option may reduce the performance of these processes in false-positive cases. The second option is to periodically check the set status in the CCA detector. If any time the cross-process misses between spy and Trojan reduces in s', the set may again reset as non-suspicious. Though the chances of such false-positive cases in the CCA detector are very few, we have only used the first option in this paper.

4.2.3. Maintaining process ID

The replacement policy of TPPD needs the process id of a block stored in the LLC. Fig. 3(b) shows it as the metadata of each LLC block. The existing partitioning techniques [15,16] as well as some replacement policies [31,33,34] need process id for each block. Hence the overhead of maintaining process id in LLC as metadata cannot be considered as the overhead of TPPD. In case we assume only one process can run in a core, then instead of a process id, we can also use the core id. Storing core id takes fewer bits than storing process id. This policy is used in some existing works [15,16]. However, in a practical design, each block must store the process id. The additional components required for TPPD (as shown in Fig. 3(b)) also need to store process id. We have considered all the additional components as an overhead of TPPD. In Section 5.6, we have calculated the hardware overhead of TPPD, both assuming core id and process id as TPPD components.

4.3. TPPD for LRU replacement policy

Algorithm 1 shows an implementation of $V_x(s', p)$ for LRU replacement policy. The alternative victim ($V_x(s', p)$) in LRU can be selected in two ways. The first method is selecting a random block not belonging to p, and the second method is to choose the next LRU block that does not belong to p. The algorithm describes the second method. All the experimental analyses in this paper use the second method. The complete mechanism of TPPD in terms of LRU replacement policy is discussed in Algorithm 2. The terminology used in this algorithm is described in the box. As mentioned in Section 4.2, we have used s' to represent the targeted set, but to write an algorithm, we have used i to represent any LLC set. The set can be either suspicious or non-suspicious. Here $0 \leq i < N$, and N is the total sets of LLC.

The main function in this algorithm is $EvictionVictim(i, p)$, which selects the appropriate victim block from the LLC as per the requirement of TPPD. The function is called every time a process p requests for a block and has a conflict miss. Process p is called an incoming process, and the block it requested as an incoming block. Initially, an original eviction victim w is selected per the LRU replacement policy (Line 2). Then it checks if the requested set is under attack or not (Line 3). If not, the original victim block w is returned (Line 4). However, additional checks are required if the request is for the suspicious set. Everything mentioned from Line 5 onward in this function is for a suspicious set. First, the owner process p_w of the victim block w is determined (Line 6 to Line 13). The p_w can be either spy (pS), Trojan (pT), or an innocent process. If the incoming process p is innocent, it can replace any block from the set. Hence, in that case, w is evicted irrespective of p_w, and the corresponding counter ($CsT[i]$ or $CsT[i]$ based on p_w) needs to be updated. Lines 14 and 15 do the same. The function $updateCounter(i, p_{in}, p_{out})$ is used for this which is discussed later. Similarly, the original victim w is evicted when the incoming block and victim block belong to the same process but without an update in the counters.

The condition mentioned in Line 16 is true only when these three conditions are true: (a) p and p_w are not the same, (b) the incoming process p is either Trojan or spy, and (c) the value of counter corresponding to p_w is less than the threshold. In this case, the alternative victim blocks w'' are selected by calling function $findVictimExcept(i, p)$ (Line 17). This function is defined in Algorithm 1. New victim block w'' can be from the incoming process p or from an innocent process. The counter update is required only in the latter case (Line 18–20). In case the condition written in Line 16 is false then the original LRU victim w is selected as victim block (Line 23), and counters are updated.

Terminology used in Algorithms:

- $EvictionVictim(i, p)$**:** Finds the victim block to be replaced from set i.
  - i: Set number of the victim block.
  - p: Owner process of the incoming block. Or the process responsible for triggering the cache replacement.
- $isUnderAttack(i)$**:** Returns true if the set i is under CCA attack, otherwise false. The value is provided by the status bit maintained in additional TPPD structure.
- $getTrojan(i), getSpy(i)$**:** Returns the suspicious process pair(spy and Trojan) on set i. The values are provided by the two suspicious process identifier maintained in TPPD structure.
- $getLRUVictim(i)$**:** Returns $V(i)$ from the set i. Or returning the victim block as per LRU replacement policy.
- $checkOwner(i, w, p)$**:** Returns true if process p is the owner of the block w in set i, otherwise false.
- $Owner(w)$**:** Returns owner process of block w. Also represented as $O(w)$.
- $pS[i]$ **and** $pT[i]$**:** Process identifiers of spy and Trojan respectively.
- $CsP[i]$ **and** $CsT[i]$**:** The counters to keep track of Spy and Trojan's blocks respectively in a suspicious set i. This is also defined in Section 4.2.
- $updateCounter(i, p_{in}, p_{out})$**:** Updates $CsP(i)$ or $CsT(i)$ for the set i. Here w is the incoming block. Here p_{in} and p_{out} are process id of incoming block and victim block respectively.
- $findVictimExcept(i, p)$**):** Returns $V_x(i, p)$, as discussed in Algorithm 1.
- th_s **and** th_t**:** Threshold partition size of spy and Trojan respectively.

Function $updateCounter(i, p_{in}, p_{out})$ keeps spy and Trojan updated based on the id of incoming process (p_{in}) and process of victim block (p_{out}). Note that no counter updates are required if p_{in} and p_{out} are the same. Also, counter updates are not required in non-suspicious sets. Since the counter maintains the total number of blocks available in the set for Trojan and spy, they may need to be incremented or decremented during an eviction. From the algorithm, it can be observed

Input: sId: Cache set index. $omitP$: Process whose block not to evict.
Output: A victim block not belonging to process $omitP$.

```
Function findVictimExcept(sId, omitP):
    /*Assuming cache[row, assoc] is the set-associative LLC
    where row is the total sets and assoc is the associativity.
    victim_index and max_age are two variables.*/
    k = 0
    while k ≠ assoc do
        if checkOwner(sId, cache[sID, k], omitP) is FALSE then
            victim_index = k
            max_age = cache[sID, k].age
            break
        end
        k++
    end
    while k ≠ assoc do
        if (max_age < cache[sId, k].age) then
            if checkOwner(sId, cache[sID, k], omitP) is FALSE
             then
                victim_index = k
                max_age = cache[sID, k].age
            end
        end
        k++
    end
    return cache[sId, victim_index]
end
```

Algorithm 1: Modified Eviction Policy for LRU

that TPPD applies a dual victim policy only to the suspicious sets. Also, the need for selecting the alternative victim is only required when Trojan and spy try to remove each other's block. Section 5 shows experimental analysis on different threshold values (th_s and th_t) that can be used for this algorithm.

4.4. How does TPPD effectively mitigate covert channel attack?

In this part, we give a theoretical analysis of how TPPD dismantles LLC-based cross-core covert channel attacks with minimal impact on system performance. Our observations and their explanations are described as follows:

Observation 1: The communication between the spy and the Trojan of the covert channel attack is degraded when the suspicious process pair is prevented from accessing all the ways of a targeted cache set.

Explanation: A low and high cache access latency pattern observed by a spy to determine the bit transmitted by Trojan is at the foundation of a successful covert channel attack. Trojan establishes this pattern by removing all of the spy's data in one case (e.g., bit 1) or none at all for another bit (bit 0). In TPPD design, cache interference between spy and Trojan is decreased by restricting them from evicting each other' block when the current number of blocks of these processes falls below the threshold. As a result, the Trojan is unable to replace all of the spy's data for sending bit 1, and the spy cannot fully prime the cache set by replacing Trojan's data with its. In this way, the spy's easily identifiable access latency pattern is obscured.

Observation 2: A targeted defence mechanism that affects only suspected sets and processes limits the overall system performance degradation.

Explanation: The main feature of our proposed defence mechanism is that it is a targeted countermeasure. It affects only processes and cache sets detected to be participating in covert channel attack by the CCA detector. Modified replacement policy to implement TPPD design

Input: i: set index, p: process requesting replacement.
Output: w: way of block to be evicted from set i

```
Function EvictionVictim(i, p):
    w=getLRUVictim(i)
    if isUnderAttack(i) is FALSE then
        return w  // Return V(i).
    else
        /* The set i is under attack.                    */
        pT=getTrojan(i); pS=getSpy(i)
        if checkOwner(i, w, pS) is TRUE then
            // If the owner of w is spy.
            p_w = pS
        else if checkOwner(i, w, pT) is TRUE then
            // If owner of w is Trojan.
            p_w = pT
        else
            p_w = −1  // if owner of w is an innocent
             process.
        end
        if ((p! = pS)&(p! = pT)))||(p == p_w) then
            /* Incoming block from innocent process
             or both incoming and victim from same
             process.                                   */
            updateCounter(i, p, p_w); return w
        else if ((p_w == pS)&(CsP[i] < th_s))||((p_w ==
         pT)&(CsT[i] < th_t)) then
            w'' = findVictimExcept(i, p_w)
            if checkOwner(i, w', p) is FALSE then
                updateCounter(i, p, −1)
            end
            return w''
        else
            updateCounter(i, p, p_w); return w
        end
    end
end
Def updateCounter(i, p_in, p_out):
    /* p_in is the process of incoming block and p_out
     is the process of outgoing (or victim)
     block.                                             */
    if p_in! = p_out then
        if p_out == pS then
            CsP[i] − −
        else if p_out == pS then
            CsT[i] − −
        end
        if p_in == pS then
            CsP[i] + +
        else if p_in == pS then
            CsT[i] + +
        end
    end
end
```

Algorithm 2: Modified LRU Replacement Policy for implementing TPPD

is activated on suspicious sets, and it restricts only conflict misses occurring between the suspicious processes. Original replacement policy is implemented for remaining sets not participating in covert channel attack. On suspicious sets, Trojan and spy processes have restrictions on evicting each other's block. The innocent processes have access to the entire cache set and can evict blocks of any process as determined by the original replacement policy. However, when suspicious processes are restricted from evicting each other's block, a new block is selected

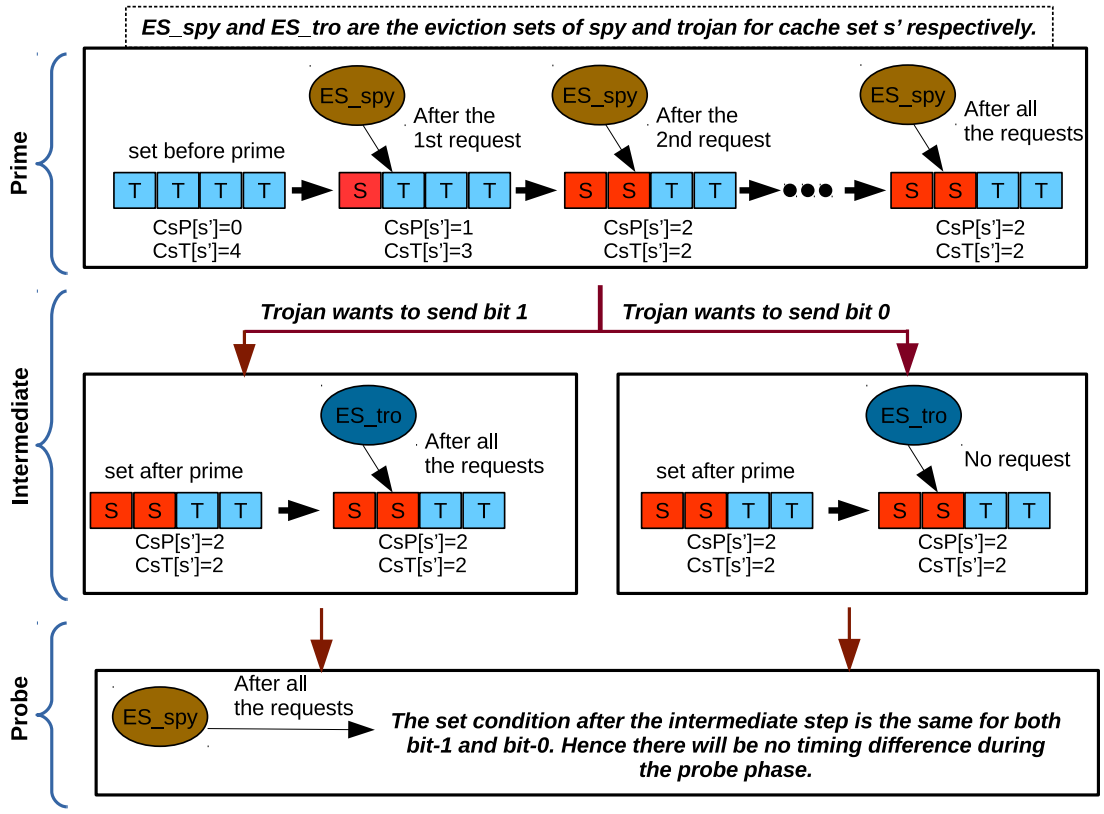


Fig. 5. An example to demonstrate the working of TPPD after detecting a CCA on set s' for pS (as spy) and pT (as Trojan). We have assumed that the set s' contains all the blocks of Trojan during the detection. The prime process starts with this assumption. The TPPD configuration considered here is TPPD-2. The figure shows that the Prime+Probe method cannot perform CCA once TPPD-2 becomes active.

for eviction. This can lead to premature eviction of benign process blocks, thus slightly impacting overall system performance.

4.5. Calculating performance of TPPD

TPPD can use different threshold values for th_s and th_t. These values can be either the same or different for spy and Trojan. In the rest of the discussion, we have considered the same values ($th_s = th_t$) for both spy and Trojan. The TPPD-z is considered as a configuration having $th_s = th_t = z$. The minimum value of z is 1, and the maximum is $A/2$, where A is the associativity of the LLC. The value of z can be changed dynamically, but for most of our experiments, we have fixed the value of z throughout the execution.

We have measured the isolated performance impact on benign applications through the following steps:

- When only the attacker processes are executing in the system, the impact of TPPD-z in terms of LLC misses ($DIFF_z$) is calculated by differentiating number of misses in LLC having TPPD-z and no defence.

$$DIFF_z = Misses_z - Misses_0 \quad \forall z \in (1, A/2) \tag{1}$$

  Here $Misses_z$ means total misses on LLC while TPPD-z is active. $Misses_0$ means total LLC misses when no defence mechanism is implemented in LLC.
- In the next scenario, a Parsec benchmark [35] is run parallel to the attack processes. The difference in LLC misses ($DIFF'_{z,b}$) encountered in TPPD-z ($Misses_{z,b}$) and without defence ($Misses_{0,b}$) is measured for benchmark b.

$$DIFF'_{z,b} = Misses_{z,b} - Misses_{0,b} \quad \forall b \in PARSEC \tag{2}$$

  Here $Misses_{z,b}$ means total LLC misses while TPPD-z is active. $Misses_{0,b}$ means total LLC misses without a defence mechanism.
- The isolated impact of TPPD-z on a benign application ($D_{z,b}$) and average impact($AvgD_z$) is calculated as described in Eqs. (3) and (4) respectively.

$$D_{z,b} = DIFF'_{z,b} - DIFF_z \tag{3}$$

$$AvgD_z = \sum_{b \in PARSEC} \frac{D_{z,b}}{T_n} \tag{4}$$

  Here, T_n is the total number of PARSEC applications considered.

These equations are used in Section 5 for performance analysis. The higher positive difference ($AvgD_z$) indicates more significant performance degradation of benign processes. A difference close to zero will signify that there has been no effect on these processes.

4.6. An example of TPPD

An example discussing how our proposed approach deteriorates covert channel communication on LLC is shown in Fig. 5. Consider the Prime+Probe based CCA attack mounted on set s' of LLC. After the attack detector detects this attack; it sends the set id s' along with process ids pS and pT to TPPD. When TPPD-2 is deployed initially, we assumed the entire cache set s' contains Trojan's block (Trojan is sending 1) as shown in Step 2 of Fig. 2. This assumption is considered without any loss of generality and a better understanding of TPPD working.

During the prime phase, the spy requests for blocks from its eviction set. From the figure, it can be observed that at the end of all the block requests of the prime phase, the spy could not able to remove all the Trojan's blocks from the set. This is because of the dual victim policy of TPPD-2. In its first two accesses, the spy is able to replace Trojan's block as intended. However, for the next spy access, if the original LRU victim spy block is evicted, the number of spy blocks present in the set falls

Table 2
System configuration for the experimental setup.

Parameter	Specification
Simulator	gem5 [21]
Architecture	4 cores each at 2.0 GHz
Coherence Protocol	MESI two level
Level 1 Cache	Inclusive L1I/L1D Private, 64 KB, 4-way, 2 cycles latency, 64B blocks
Level 2 Cache (LLC)	Inclusive Shared, 2MB 8-way, 18 cycles latency, 64B blocks
LLC structure	Single Bank. Uniform Cache Access (UCA)
Replacement Policy	LRU
Main Memory	DRAM, 250-cycle latency

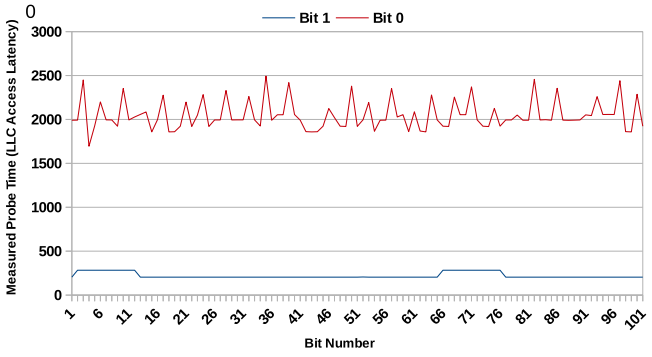


Fig. 6. Cache access latency observed by spy for bit 0 and 1 in LLC with no TPPD.

below the threshold of 2, which is not allowed by TPPD-2. In this case, as per the dual victim policy of TPPD, an alternative replacement victim that does not belong to the Trojan process is selected using $V_x(s', pT)$. After the prime phase, in the case of sending bit 1, the Trojan removes all the blocks of spy by requesting a block from its own eviction set. However, because of TPPD-2, the Trojan cannot remove all the spy's blocks. From the figure, it can be observed that, after the intermediate phase, the status of the set is the same for both bit 1 and bit 0. In the probe phase, when spy probes set s' for the blocks from its eviction set, it encounters 2 misses and 2 hits in both cases of a bit 1 and bit 0.

5. Experiments and results

To perform the experimental analysis, TPPD is implemented on a full system cycle-accurate simulator, gem5 [21]. We consider a 4-core system setup with two levels of the cache hierarchy. Each core has its private L1 cache, and the L2 cache is considered a shared LLC. Other parameters of the setup are shown in Table 2. To simulate the CCA attack, we have developed our own Trojan and spy applications. These applications are written in C++ and can be executed on gem5. We considered a generic Prime+Probe attack as described in Fig. 2 in this application. As mentioned in Section 1, we have successfully tested these attacks on Intel Core i9-10885H, i7-9700 and i7-7700. The ability to perform CCA attack by these two applications have experimented first on gem5. Parsec benchmarks [20] are used to measure the performance of innocent applications in the presence of attacker applications. From the 4-cores, two cores are assigned to the attacker processes (spy and Trojan). A multithreaded Parsec benchmark is binded to the other two cores. Different configurations of TPPD (TPPD-z) are considered in order to fully analyse the performance behaviour of TPPD. The value of z varies from 1 to 4 (8 is the associativity of the LLC). TPPD-0 means baseline design with no defence mechanism.

We have considered all the Parsec benchmarks as innocent applications. To measure the worst-case performance impact, the attacking applications are developed such that once the attack starts, it continues during the execution of the system. Fig. 8 shows the pseudocode of the two attacking applications (spy and Trojan) used in our experiments. Both sender and receiver have colluded and decided on the targeted set i, prime interval Tp and have their own eviction set (ESi and $ES'i$) corresponding to this set. In the first step, receiver accesses the addresses of $ES'i$ to prime set i. Priming of targeted set is done multiple times for interval Tp to ensure presence of ES addresses in the set. Next, based on the value of the bit sent, either sender replaces receiver's data in the targeted set by accessing addresses from its own eviction set (for bit 1) or remains idle (for bit 0). In the next step, receiver accesses the set using the same $ES'i$ and measures the access time. Based on this value of the access time, bit sent by the sender is determined. After probing, receiver waits for the prime interval for synchronisation purposes.

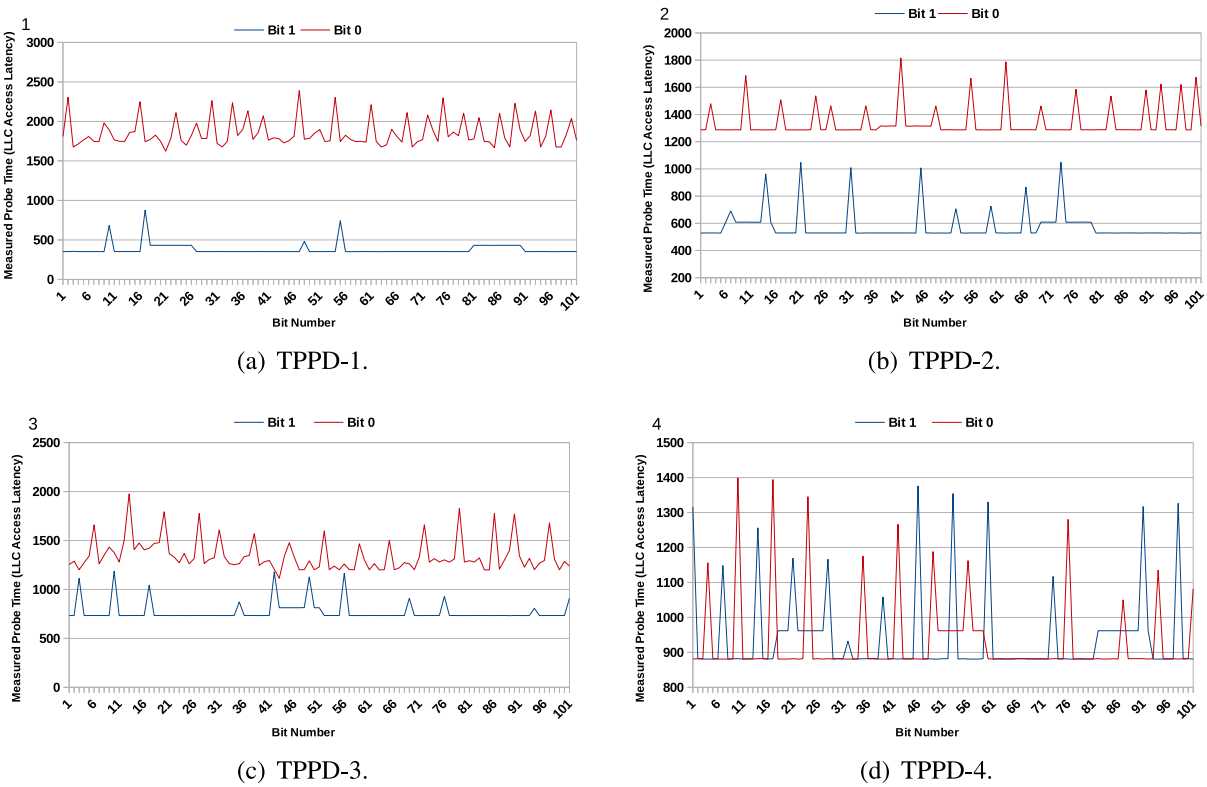
5.1. Security analysis

We assess the effectiveness of TPPD against cross-core based covert channel attacks on LLC. As described in Section 4.4, TPPD reduces the difference in probe time observed by the spy for bit 0 and 1. This difference is substantially more significant when no defence mechanism is active, as evident by Fig. 6. Here bit 0 represents cache miss and bit 1 as a cache hit. TPPD reduces this difference such that the information received by the spy becomes noisy. This difference becomes close in TPPD-1, TPPD-2 and TPPD-3 but does not overlap as represented in Fig. 7(a), 7(b) and 7(c) respectively. These TPPD configurations are effective against attacks based on fine-grained information, i.e., observing the number of cache accesses by victim applications in the targeted set. However, in a low-speed covert channel attack, we considered these configurations would not be effective, especially when there is low system noise. However, TPPD-4 was able to completely obfuscate the receiver's access time pattern used to identify bit 0 and 1 as illustrated in Fig. 7(d). Hence, TPPD-$A/2$ is the recommended configuration to prevent most CCA attacks. Here A is the associativity of the LLC.

As suggested in [36] we have also tested both the CCA attack and the proposed countermeasure on different LLC sizes, associativity, and replacement policies. The attack is tested for LLC sizes of 1MB, 2MB, and 4MB. The replacement policies tested are LRU, SRRIP [37], Random, and Least Frequently Used. Based on our experimental analysis, we have observed that the CCA is possible in almost all configurations. However, the configurations with random replacement policy shows some noises. The proposed countermeasure, TPPD, also works effectively for all these configurations.

5.2. Performance analysis

While deconstructing cache timing channel-based attacks, the attack defence mechanism must guarantee that system performance does not suffer. Fig. 9 represent the total LLC misses encountered in different benchmark applications for TPPD-z and TPPD-0. The upper part of the figure shows the actual values (in millions), while the bottom part shows the normalised LLC misses. The values are normalised w.r.t. TPPD-0. As mentioned above, the benchmarks are combined with the attacker applications during the execution. However, in the figure, the benchmark named as "only-attack" is not a Parsec benchmark. This is designed only with the combination of Trojan and spy. To analyse the performance impact of TPPD on innocent applications, it is important to first analyse the performance of "only-attack". From the figure, it can be observed that, in the case of "only-attack", if TPPD is applied to prevent CCA, the LLC misses are increased by 17% as compared to TPPD-0 (no defence). All the configurations of TPPD show almost similar results. However, for the rest of the benchmarks, the increment is less. On average, while a Parsec benchmark executes with two attacker applications, which are constantly trying to covertly communicate, the miss count in TPPD is 5%, 7%, 6%, and 9% more



(a) TPPD-1. (b) TPPD-2.

(c) TPPD-3. (d) TPPD-4.

Fig. 7. Cache access latency for bit 0 and bit 1.

Sender

input: ES_i:Eviction set of size w for set i
// w: cache associativity
Step 1: Wait for Tp time for receiver to prime the target set.
start = time()
while (time()-start<Tp) {
Do Nothing }
Step 2: Send bit
If (bitToSent==1) {
start = time()
while (time()-start<Tp) {
Access all addresses from ESi } }
else {
while (time()-start<Tp) {
Do Nothing. } }
Step 3: sender waits for Tp for receiver to probe the target set.
start = time()
while (time()-start<Tp) {
Do Nothing }

Receiver

input: ES'_i:Eviction set of size w for set i
// w: cache associativity
Step 1: For interval Tp, prime set i.
start = time()
while (time()-start<Tp) {
Access address from ES'_i }

Step 2: wait for Tp time as sender sends the bit
while (time()-start<Tp) {
Do Nothing. }

Step 3: Probe set i & measure access time.
start = time()
totalTime=accessTime of ES'_i
if totalTime>th //th is threshold
bitSent=1
else
bitSent=0
while (time()-start<Tp) {
Do Nothing }

Fig. 8. Pseudo code for Prime+Probe attack considered in this work. Sender means Trojan and Receiver means spy.

than TPPD-0 for TPPD-1, TPPD-2, TPPD-3, and TPPD-4 respectively. Similar to TPPD-4 results were observed for TPPD-8 when the attack is reconfigured for 16-way associative LLC. The degradation (high miss count) in TPPD-4 is more because a higher threshold value may evict some more innocent blocks from the cache. A similar result can be seen in Fig. 10 for Misses per Thousand Instructions (MPKI).

The important point here is that the misses are higher when TPPD applies only to the attacker set in "only-attack". However, the miss overhead is reduced when the cores mix attack applications with some innocent Parsec benchmarks. It means that the miss overhead of TPPD is higher only for the attacker applications. The overhead is less for innocent applications. The two main reasons for this are:

- TPPD is a targeted CCA countermeasure. Hence from the non-targeted set, the victim is selected per the original replacement policy.
- In the targeted set, the innocent blocks have no restrictions, and they can remove any other block from the set.

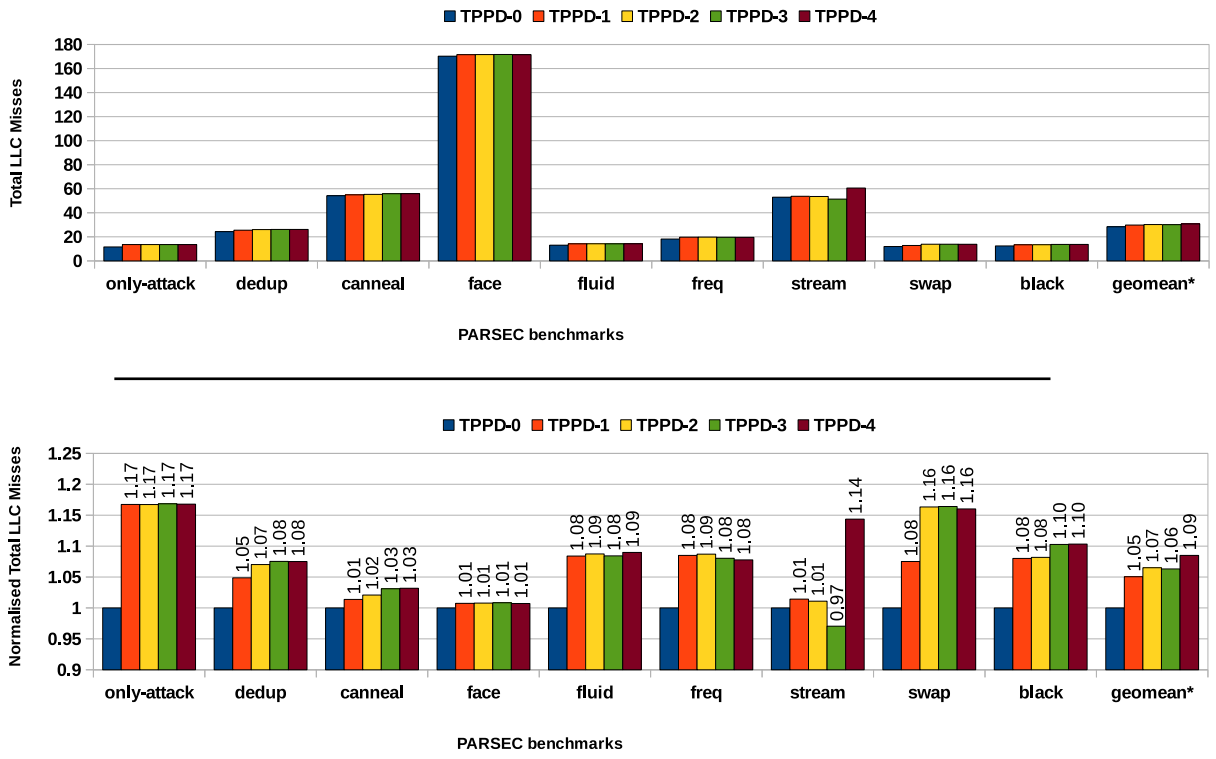


Fig. 9. Total misses in LLC for PARSEC benchmarks on different TPPD configurations. *The calculation of geomean is excluding "only-attack". The lower part of the figure shows the LLC misses normalised to TPPD-0.

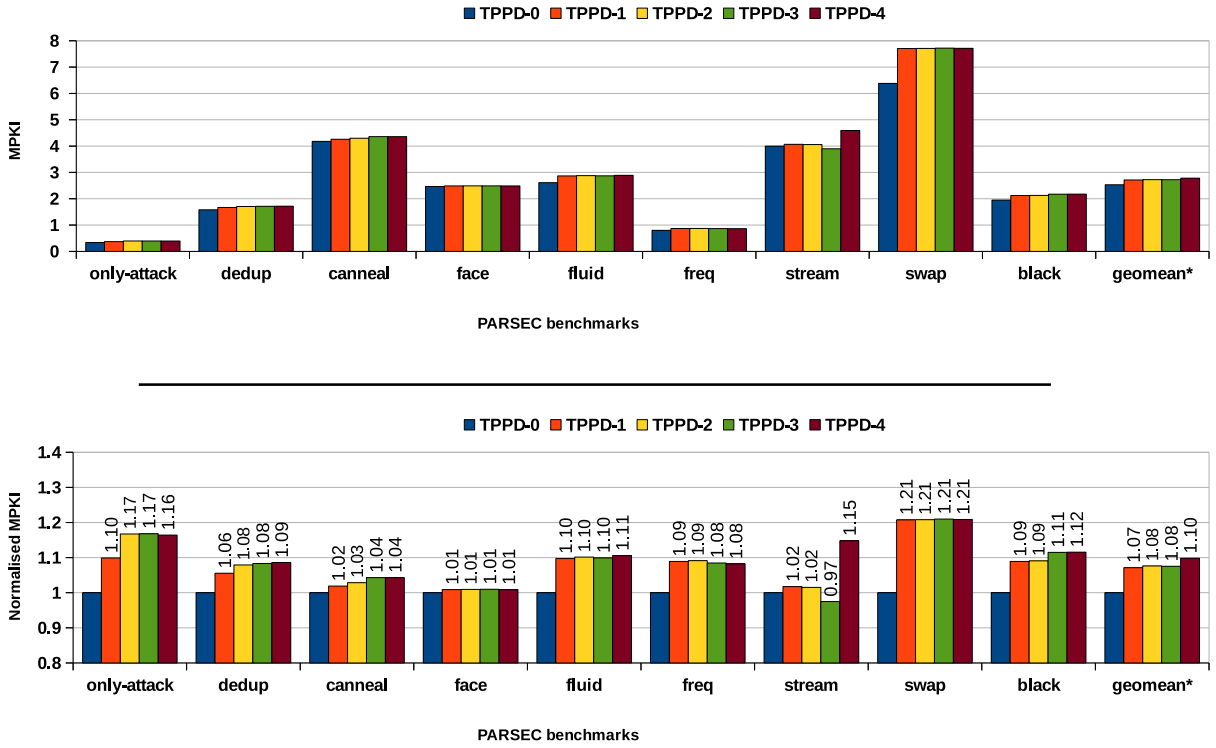


Fig. 10. Total MPKI in LLC for PARSEC benchmarks on different TPPD configurations. The lower part of the figure shows the MPKI normalised to TPPD-0.

Because of these two reasons, TPPD becomes an efficient countermeasure for cross-core covert channel attacks.

Fig. 11 shows the performance of TPPD in terms of Instruction per Cycle (IPC) as compared to TPPD-0. It can be observed from the figure that the performance of the system slightly degrades in TPPD as compared to the baseline (TPPD-0). The degradations are 0.73%, 0.87%, 0.71% and 0.86% as compared to TPPD-0 in TPPD-1, TPPD-2, TPPD-3, and TPPD-4 respectively. Hence, the proposed countermeasure does not create any significant performance overhead in the system to defend a CCA. Also, the overall degradation shown in the figure is mostly for affecting the spy and Trojan. The innocent application faces less performance reduction. However, compared to a baseline

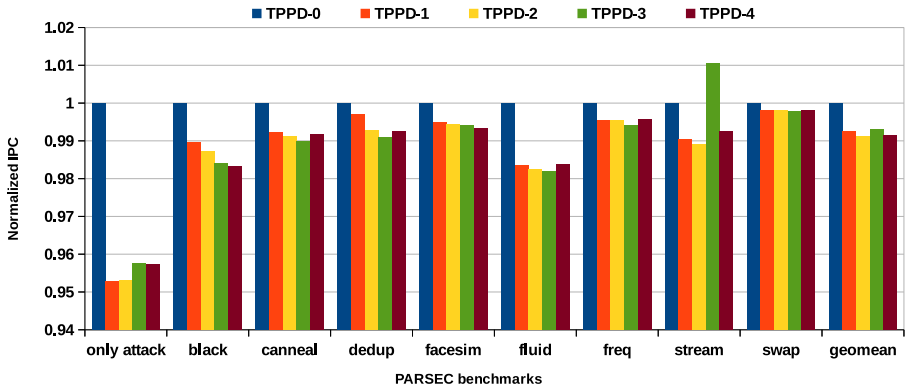


Fig. 11. Normalised IPC over TPPD-0 in single-set attack for PARSEC benchmarks on different TPPD configurations. *The calculation of geomean is excluding "only-attack".

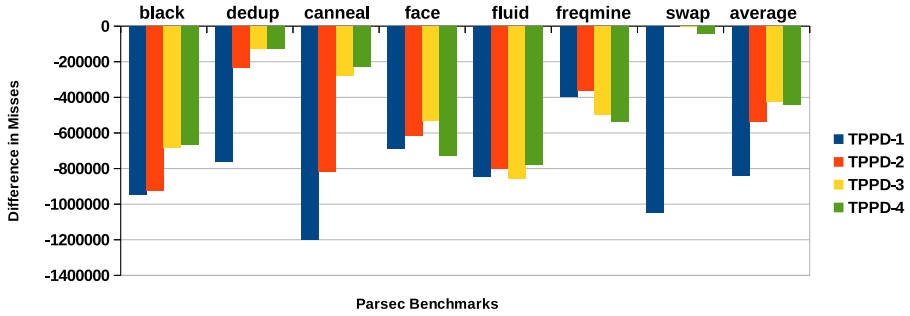


Fig. 12. Isolated LLC misses for PARSEC benchmarks with different TPPD configurations measured using Eq. (3).

where no CCA is happening, TPPD-0 (baseline under CCA) shows some performance degradation. The reasons for such degradation are: (a) congestion in the LLC request queue, and (b) replacing the blocks of innocent application by the attacker application. As the number of targeted set increases in CCA, the performance of the system decreases. The countermeasures like TPPD and HybridCache [38] are not handling this degradation directly. These countermeasures ensure that the CCA is prevented and by doing this no significant performance degradation should occur in addition to degradation facing by TPPD-0. All the TPPD configurations behave same as the baseline which is not under CCA, when no CCA executes in the system.

The impact of TPPD's different configurations on system's overall MPKI compared to TPPD-0 is depicted in Fig. 10. Fig. 12 shows how much change in the misses has occurred to the benign application. The calculation is performed using Eq. (3) and Eq. (4). As evident from Fig. 12, the increment in total LLC misses (compared to TPPD-0) when the benchmark runs alongside the attack process is always less than the misses when just attack runs in LLC augmented with TPPD-z. In this figure, a less number (large negative number) means less overhead on an application. On average, the applications face the lowest overhead in TPPD-1. However, the miss overhead in TPPD-4 (which is the recommended configuration) is also less.

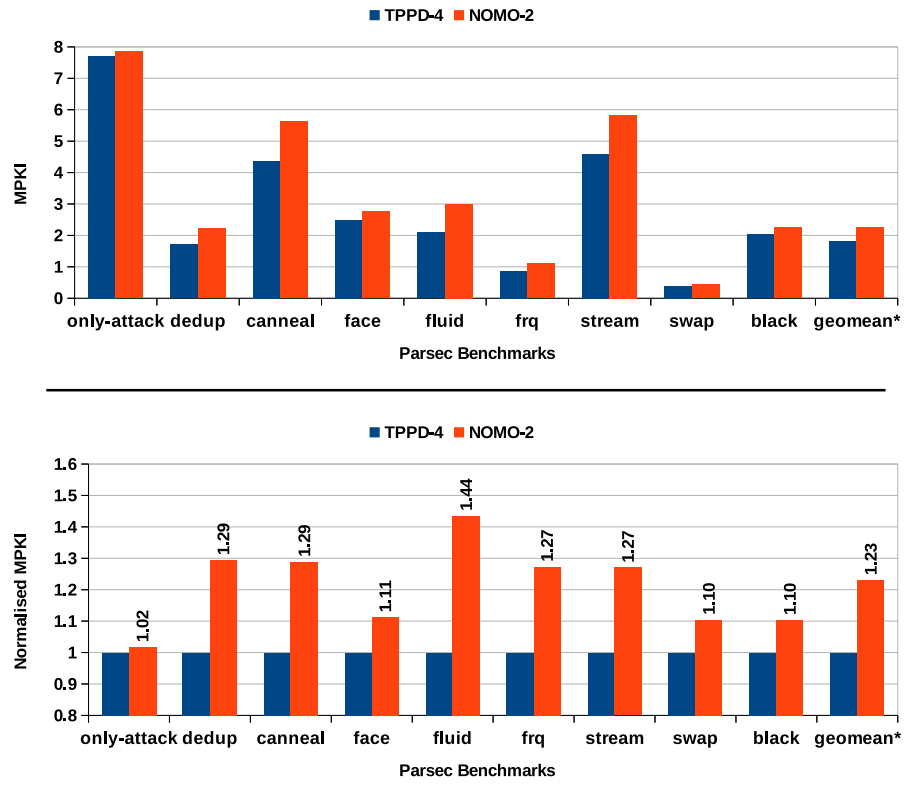
5.3. Comparison with existing works

Existing partitioning-based approaches such as PLCache [24], SecVerilog Cache [39] and SecDCP [26] can combat cache-based side-channel attacks in which an attacker process attempts to reveal the cache access pattern of an innocent victim process in order to leak its secret, such as the private key of cryptography algorithms. These secure cache designs rely on a reliable operating system, compiler, or programmer to recognise security-critical processes or data that could be vulnerable to cache timing channel attacks. PLCache pre-loads and lock the security critical data in the cache and prevents its eviction by unlocked data. In SecVerilog and SecDCP, processes are assigned to different security classes and cache interference between these security domains is restricted. These security measures are ineffective in the presence of covert timing channel attacks where both processes are malicious.

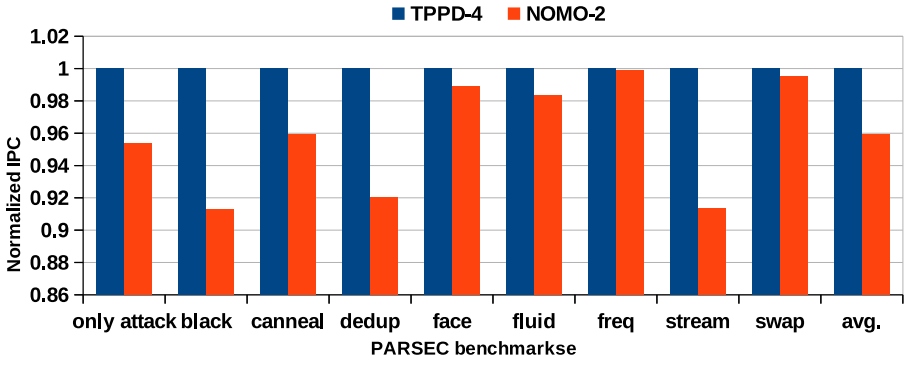
The static cache partitioning based secure cache design, NOMO [25] can defend against covert channel attacks by completely isolating ways of spy and Trojan. In our setup, NOMO-2, in which two ways were fixed per process, was able to dismantle the covert channel attack. However, significant performance degradation was observed in the case of NOMO-2 in comparison to our design TPPD-4 as shown in Fig. 13 in terms of MPKI and IPC. It can be observed from the figure that NOMO shows less miss overhead over TPPD-4 when executed for "only-attack". However, when innocent applications are executed with attacker applications, the overhead increases significantly. The main reason for that is the non-targeted nature of NOMO. The restricted eviction mechanism is uniformly applied to all the sets. Hence, the overhead of misses increases in non-targeted sets also. Since the attackers mainly requested for the blocks mapped to the targeted set, the other set mostly served innocent applications. Uniform implementation of the restricted eviction policy impacts the miss rate of innocent applications on non-targeted sets. On average, NOMO-2 had 23% higher MPKI than TPPD-4. This increased miss rate of benign benchmarks impacts the IPC of NOMO-2 as it suffers 4.09% more degradation compared to TPPD-4.

The performance impact of our proposed secure design was also compared with the recently proposed HybridCache [38]. HybridCache also proposes a process isolation based countermeasure for side channel attack. The LLC is divided into two parts: (a) fully associative (FA) part and (b) set-associative (SA) part. The applications run in secure cores map to the FA part of the LLC, while the applications run in other cores map to the SA part of the LLC. HybridCache is mainly proposed for mitigating side-channel attacks while TPPD is proposed for mitigating covert-channel attacks.

In side-channel attacks, the identity of the targeted security critical process is known to the system. However, this is not the case for covert channel attacks involving two colluding suspicious processes.



(a) MPKI



(b) IPC

Fig. 13. Comparison of MPKI for NOMO-2 vs TPPD-4 when PARSEC benchmark are executed in parallel to CCA. Here *geomean is calculated excluding "only-attack".

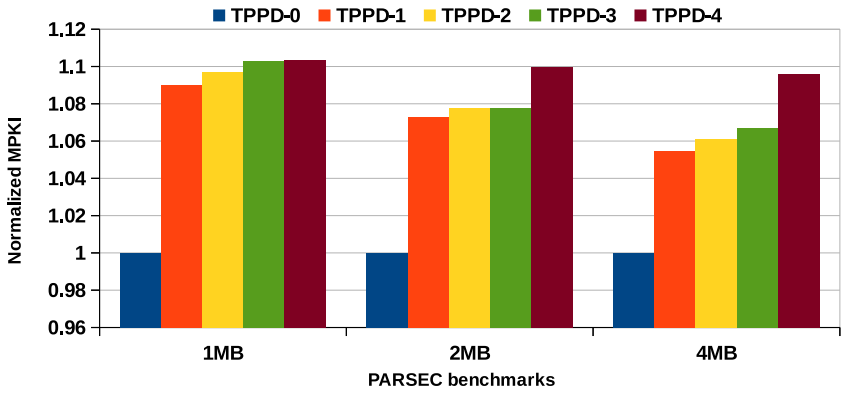
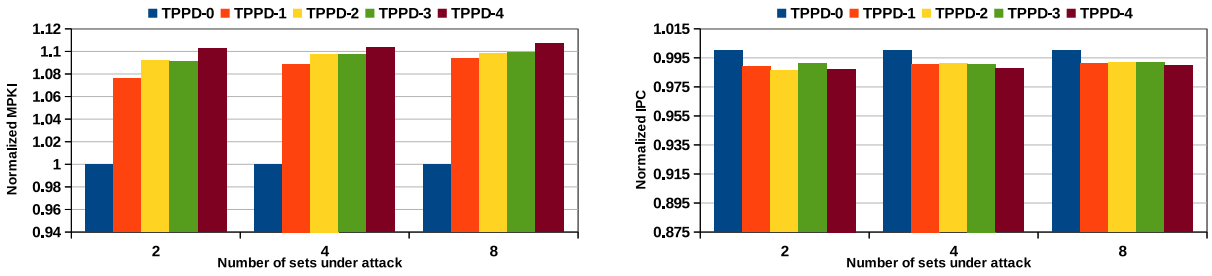


Fig. 14. Average Normalised MPKI of TPPD over TPPD-0 on different LLC sizes. The average MPKI is calculated after executing multiple PARSEC applications.



(a) MPKI. (b) IPC.

Fig. 15. Average Normalised MPKI and IPC over TPPD-0 in multi-set attacks for PARSEC benchmarks on different TPPD configurations. The average MPKI is calculated after executing multiple PARSEC applications.

We have considered three scenarios for comparing the performance of the HybridCache with TPPD: (1) Trojan is mounted on a secure core and accesses the FA part of the LLC, whereas the spy and benchmark applications run on non-secure cores, (2) Benchmark applications and Trojan run on a secure core, and spy runs on a non-secure core, (3) No active attack on the system, one innocent application is running on secure core and another on the non-secure core. The performance impact of HybridCache compared to TPPD-4, when Trojan accesses the FA part of the LLC is presented (scenario 1) in Fig. 16. As observed, HybridCache has 1.29% degradation in overall system IPC as compared to TPPD-4. However, HybridCache significantly impacts the IPC of benign processes, which is 26.34% less as compared to TPPD-4. HybridCache, in this scenario, gets less number of misses for Trojan as compared to TPPD-4. This is because the Trojan has the access to the full FA part which is significantly larger than the 4-ways (of a single set) available in TPPD-4. However, the high access latency of FA part makes congestion in the read/write queue of LLC, which causes performance degradation of innocent applications accessing the SA part.

In the second scenario, HybridCache degrades the overall IPC and the benign process's IPC by 6.06% and 30.02%, respectively as compared to TPPD-4. In the third scenario, when no attack is present, TPPD works the same as the baseline. If a security-critical process is running alongside another innocent process in TPPD, it can access the full cache with no restriction. However, in HybridCache such a security-critical process can only access the FA part. We have observed a degradation of 13.4% and 7% in terms of MPKI and overall IPC respectively for HybridCache as compared to TPPD-4.

5.4. Sensitivity analysis

In this section, we discuss the performance impact of TPPD with different system and attack parameters.

5.4.1. Impact of different LLC sizes

The TPPD is also tested for different LLC sizes. Fig. 14 shows the average MPKI comparisons of TPPD with TPPD-0. The associativity of all the LLC sizes is kept same. The attack considered for this experiment is single-set attack. As the cache size is increasing, the overhead of MPKI is slightly decreasing as compared to TPPD-0. This is because, larger sized LLC has more cache sets and space, which reduces the misses in the LLC.

5.5. Multi-set attack

In this section, we analyse how well TPPD maintains system performance when the attack is extended on multiple sets. We extended the attack up to eight LLC sets whereas the prime probe interval remains the same. Fig. 15 presents the comparison of different TPPD configurations in terms of MPKI and IPC. It can be observed that the performance degradation (of the innocent applications) up to 8-set attack is less than 2%. This is because the innocent process has no restriction in accessing the cache sets. However, we have observed higher performance degradation for the attack on more than eight sets. There are two reasons for such degradation: (a) The request of the LLC gets congested with multiple requests from the attacker application, and (b) The attacker applications evict the blocks of innocent applications. Such performance degradation cannot create any correctness issue in TPPD but an attacker can perform a denial of service (DOS) attack in TPPD using multiple targeted sets. A similar DOS attack is also possible in other techniques like NOMO and HybridCache. We left out how to handle such DOS attacks on TPPD as future work.

Table 3
Storage overhead calculation of TPPD as per the LLC configuration mentioned in Table 2. We have assumed 16 bits to represent a process id as the maximum process id possible in Linux is 32768.

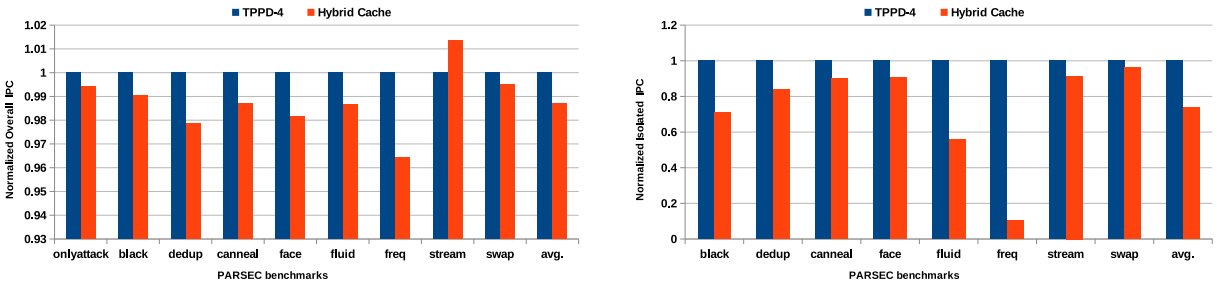
Storage Overhead	*y* as core id		*y* as process id	
	Per set	Per LLC	Per set	Per LLC
Status bit	1	512 Bytes	1	512 Bytes
Suspicious process1	2 bits	2*4096 = 1 KB	16 bits	16*4096 = 8 KB
Suspicious process2	2 bits	2*4096 = 1 KB	16 bits	16*4096 = 8 KB
spy Counter	3 bits	3*4096 = 1.5 KB	3 bits	3*4096 = 1.5 KB
Trojan Counter	3 bits	3*4096 = 1.5 KB	3 bits	3*4096 = 1.5 KB
Total Overhead	**11 bits**	**5.5 KB**	**39 bits**	**19.5 KB**

5.5.1. Implementing TPPD on other replacement policies

The proposed TPPD can be implemented using any replacement policy where the dual victim policy can be implemented. Algorithm 2 can be modified for that. The internal mechanism of the functions called from Line 2 and Line 21 of this algorithm needs to be changed as per the requirement of the replacement policy. We have also experimented TPPD with SRRIP [37], Least Frequently Used, and Random replacement policies. As observed in Fig. 17 the impact across different replacement policies for TPPD-4 compared to TPPD-0 are almost same as LRU.

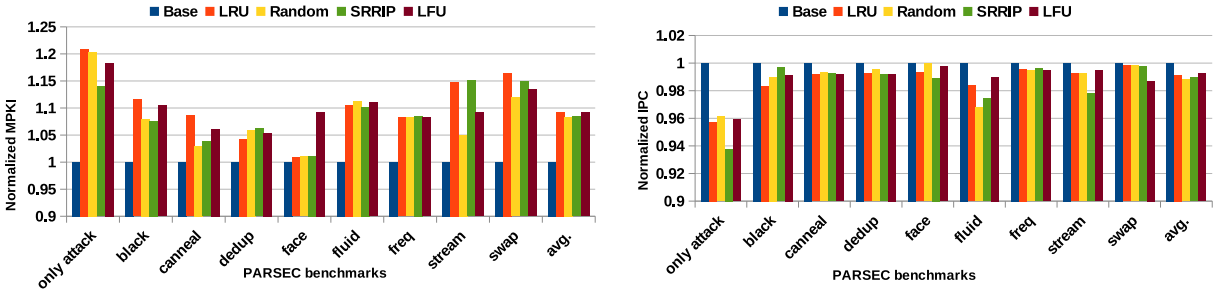
5.6. Storage and latency overhead

The structure of the LLC having TPPD as a countermeasure is already discussed in Section 4.2.1. Fig. 3(b) shows the two additional components required for TPPD: (a) TPPD Components, and (b) CCA Detector. In this section, we have discussed the storage and latency overhead of the TPPD components. As mentioned in Section 4.1, we



(a) IPC. (b) Isolated IPC.

Fig. 16. Average Normalised overall IPC and benign process's IPC over TPPD-4 with Hybrid Cache for PARSEC benchmarks. The average MPKI is calculated after executing multiple PARSEC applications.



(a) MPKI. (b) IPC.

Fig. 17. Average Normalised MPKI and IPC over TPPD-0 with different replacement policies for PARSEC benchmarks on TPPD-4. The average MPKI is calculated after executing multiple PARSEC applications.

have used the existing CCA detector technique [19] to detect the CCA attack. Each cache line is augmented with its owner process id to identify the cache block's owner process, as seen in Fig. 3(b). An alternative technique followed by some existing works [15,16] is to maintain core id instead of process id along with each cache line. However, the overhead of storing process id along with the cache line cannot be considered as the overhead of TPPD. The issues with maintaining process id along with the cache lines are already discussed in Section 4.2.3.

Fig. 3(b) shows all the counters used in TPPD Components. The first counter, $attack\_flag$, is a one-bit counter used to indicate whether or not the set is under attack. The next two fields (pS and pT) specify the attacking process pair if the set is under attack. The last two fields are spy and Trojan counters, each with a size of $log_2(A)$ bits to identify the number of spy and Trojan blocks currently in the set. Here, A is the associativity of the LLC. Thus, the total storage overhead can be determined as $N \times (1 + 2(y + log_2 A))$. Here N is the total number of sets in LLC, and y is the bits required to represent a suspicious process. In our work, y represents the cores in the system as a single process is binded per core making core id enough for identifying attacking pair. Table 3 shows the storage overhead of TPPD as per the LLC configurations mentioned in Table 2. It can be observed from Table 3 that the total storage required for TPPD is $5.5KB$, i.e. 0.26% of total LLC size. However, in real systems, multiple processes may run on the same core, making core id not sufficient for uniquely identifying processes. The table shows that when process id is used instead of core id, the total size of this additional structure is $20.5KB$, i.e. $\approx 1\%$ of total LLC size.

The operations of TPPD are performed in the background without affecting the critical execution path of the system.

6. Conclusion

This paper proposes an effective and efficient mitigation mechanism, TPPD, for cross-core cache timing channel attacks. TPPD implements way-wise partitioning on the cache sets used for covert channel attacks but only for suspicious process pairs. However, remaining benign processes have unrestricted access to these and other sets, reducing the performance impact on system performance. It successfully abolishes LLC based covert communication between Trojan and spy. Experiments have shown that it does not have any significant impact on the performance of benign applications (Parsec benchmark). The total storage overhead required for implementing TPPD design is approximately $\approx 0.26\%$ of LLC size. Compared to the existing partitioning based attack prevention mechanism NOMO, it caused 23% less LLC misses.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Colin Percival, Cache missing for fun and profit, in: In Proc. of BSDCan, 2005.
[2] Eran Tromer, Dag Arne Osvik, Adi Shamir, Efficient cache attacks on AES, and countermeasures, J. Cryptol. 23 (1) (2010) 37–71.
[3] Dag Arne Osvik, Adi Shamir, Eran Tromer, Cache attacks and countermeasures: The case of AES, in: Cryptographers' Track At the RSA Conference, 2006, pp. 1–20.
[4] Daniel J. Bernstein, Cache-timing attacks on AES, 2005.
[5] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, Ruby B. Lee, Last-level cache side-channel attacks are practical, in: Symp. on Security and Privacy, 2015, pp. 605–622.
[6] Jaspinder Kaur, Shirshendu Das, A survey on cache timing channel attacks for multicore processors, J. Hardw. Syst. Secur. (2021) 1–21.
[7] Yangdi Lyu, Prabhat Mishra, A survey of side-channel attacks on caches and countermeasures, J. Hardw. Syst. Secur. 2 (1) (2018) 33–50.
[8] Vincent Rijmen, Joan Daemen, Advanced encryption standard, in: Federal Information Processing Standards Publications, National Institute of Standards and Technology, 2001, pp. 19–22.

[9] Ronald L. Rivest, Adi Shamir, Leonard Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.
[10] Don Johnson, Alfred Menezes, Scott Vanstone, The elliptic curve digital signature algorithm, Int. J. Inf. Secur. 1 (1) (2001) 36–63.
[11] Onur Acıiçmez, Çetin Kaya Koç, Trace-driven cache attacks on AES (short paper), in: Information and Communications Security, 2006, pp. 112–121.
[12] David Gullasch, Endre Bangerter, Stephan Krenn, Cache games–bringing access-based cache attacks on AES to practice, in: IEEE Symp. on Security and Privacy, 2011, pp. 490–505.
[13] S. Sari, O. Demir, G. Kucuk, FairSDP: Fair and secure dynamic cache partitioning, in: 4th Intl. Conf. on Computer Science and Engineering, 2019, pp. 469–474.
[14] Fan Yao, Hongyu Fang, Miloš Doroslovacki, Guru Venkataramani, COTSknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies, in: Hardware Oriented Security and Trust (0000).
[15] Moinuddin K. Qureshi, Yale N. Patt, Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, in: 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), IEEE, 2006, pp. 423–432.
[16] Prateek D. Halwe, Shirshendu Das, Hemangee K. Kapoor, Towards a better cache utilization using controlled cache partitioning, in: 2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, IEEE, 2013, pp. 179–186.
[17] Chen Yang, Leibo Liu, Kai Luo, Shouyi Yin, Shaojun Wei, CIACP: A correlation- and iteration-aware cache partitioning mechanism to improve performance of multiple coarse-grained reconfigurable arrays, IEEE Trans. Parallel Distrib. Syst. 28 (1) (2016) 29–43.
[18] Anurag Agarwal, Jaspinder Kaur, Shirshendu Das, Exploiting secrets by leveraging dynamic cache partitioning of last level cache, in: Design, Automation Test in Europe Conference Exhibition, DATE, 2021.
[19] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, Guru Venkataramani, Prodact: Prefetch-obfuscator to defend against cache timing channels, Int. J. Parallel Programm. 47 (4) (2019) 571–594.
[20] Christian Bienia, Benchmarking Modern Multiprocessors, Princeton University, 2011.
[21] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, et al., The gem5 simulator, ACM SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.
[22] G. Irazoqui, T. Eisenbarth, B. Sunar, S$A: A shared cache attack that works across cores and defies VM sandboxing – and its application to AES, in: IEEE Symp. on Security and Privacy, 2015, pp. 591–604.
[23] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, Josep Torrellas, Attack directories, not caches: Side channel attacks in a non-inclusive world, in: Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World, 2019.
[24] Zhenghong Wang, Ruby B. Lee, New cache designs for thwarting software cache-based side channel attacks, SIGARCH Comput. Archit. News 35 (2) (2007) 494–505.
[25] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, Dmitry Ponomarev, Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks, Trans. Architect. Code Optimiz. 8 (4) (2012) 1–21.
[26] Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, G. Edward Suh, Secdcp: Secure dynamic cache partitioning for efficient timing channel protection, in: 2016 53nd ACM/EDAC/IEEE Design Automation Conference, DAC, 2016, pp. 1–6, http://dx.doi.org/10.1145/2897937.2898086.
[27] Moinuddin K. Qureshi, CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2018, pp. 775–787.
[28] Moinuddin K. Qureshi, New attacks and defense for encrypted-address cache, in: Proceedings of the 46th International Symposium on Computer Architecture, ISCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 360–371.
[29] Pratik Kumar, Chavhan Sujeet Yashavant, Biswabandan Panda, DAMARU: A denial-of-service attack on randomized last-level caches, IEEE Comput. Archit. Lett. 20 (2) (2021) 138–141.
[30] Gorka Irazoqui, Thomas Eisenbarth, Berk Sunar, Systematic reverse engineering of cache slice selection in intel processors, in: 2015 Euromicro Conference on Digital System Design, IEEE, 2015, pp. 629–636.
[31] Akanksha Jain, Calvin Lin, Back to the future: Leveraging Belady's Algorithm for improved cache replacement, in: Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16, IEEE Press, 2016, pp. 78–89.
[32] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., Joel Emer, High performance cache replacement using re-reference interval prediction (RRIP), ACM SIGARCH Comput. Archit. News 38 (3) (2010) 60–71.
[33] Kousik Kumar Dutta, Prathamesh Nitin Tanksale, Shirshendu Das, A fairness conscious cache replacement policy for last level cache, in: 2021 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2021, pp. 695–700.
[34] Tripti S. Warrier, B. Anupama, Madhu Mutyam, An application-aware cache replacement policy for last-level caches, in: International Conference on Architecture of Computing Systems, Springer, 2013, pp. 207–219.
[35] Christian Bienia, Kai Li, Parsec 2.0: A new benchmark suite for chip-multiprocessors, in: Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, vol. 2011, 2009, p. 37.
[36] Xiaodong Yu, Ya Xiao, Kirk Cameron, Danfeng Daphne Yao, Comparative measurement of cache {Configurations'} impacts on cache timing {Side-Channel} attacks, in: 12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19), 2019.
[37] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., Joel Emer, High performance cache replacement using re-reference interval prediction (RRIP), ACM SIGARCH Comput. Archit. News 38 (3) (2010) 60–71.
[38] Ghada Dessouky, Tommaso Frassetto, Ahmad-Reza Sadeghi, {HybCache}: Hybrid {Side-Channel-Resilient} caches for trusted execution environments, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 451–468.
[39] Danfeng Zhang, Yao Wang, G. Edward Suh, Andrew C. Myers, A hardware design language for timing-sensitive information-flow security, Acm Sigplan Not. 50 (4) (2015) 503–516.

Jaspinder Kaur A Ph.D scholar in the CSE department of Indian Institute of Technology Ropar. She has completed M.Tech (CSE) from the Department of Computer Engineering of Punjabi University Patiala, India. Her research interests include Computer Architecture, Cache Prefetching and Cache Security against timing channel attacks.

Dr. Shirshendu Das received a Ph.D. degree (CSE) from the Indian Institute of Technology Guwahati, India, in 2016. Previously he did M.Tech (CSE) from the Indian Institute of Technology Guwahati, India. Presently he is an Assistant Professor in the Department of CSE, Indian Institute of Technology Ropar, Punjab, India. His area of research includes Computer Architecture, Network-on-Chip, and Hardware Security.