# Novel Design for IE-Cache to Mitigate Conflict-based Cache-side Channel Attacks with Reduced Energy Consumption

Saqib Javed[1], Muhammad Asim Mukhtar[1], Muhammad Khurram Bhatti[1] and Guy Gogniat[2]

[1]*Information Technology University, Lahore, Pakistan*
[2]*Universite Bretagne Sud, Lorient, France*

Keywords: IE-Cache, V-Way Cache, Timing Cache Side Channel Attacks, Prime+Prune+Probe, Prime+Probe Attack.

Abstract: Cache-based side-channel attacks have raised serious security concerns in the contemporary cache architectures. To mitigate these attacks, various cache architectures have been proposed that rely on cache partitioning and random memory-to-cache mapping based methods. Unfortunately, these cache methods are not adopted by the mainstream processors because of unfavorable security and performance trade-off. In literature, the Indirect-Eviction Cache (IE-Cache), a random memory-to-cache mapping based cache architecture, has shown high security and faster execution time by introducing the principles of multi-indexing and relocating the cache lines. However, IE-Cache requires relocation of cache lines that result in high energy consumption along with security. In this paper, we alleviate the energy consumption issue in IE-Cache by introducing a pointer-based mapping between tag and data store, which we call PIE-Cache (Pointer-based IE-Cache). This enables relocation of pointers in the tag-store without relocating a large cache line in the data-store, yielding low energy consumption compared to IE-Cache. We have developed the PIE-Cache model in the gem5 simulator to evaluate the energy consumption with Micro-benchmark. The results show that the energy consumption of 1MB PIE-Cache with 4 ways and 3 levels is 20% less compared to IE-Cache with the same capacity, ways and levels over Micro-benchmark. Moreover, we have performed the security evaluation of PIE-Cache in the same way as proposed in IE-Cache study to compare the learning time of eviction sets. These results show that the complexity of learning eviction sets is similar to IE-Cache.



## 1 INTRODUCTION

Cache-based side-channel attacks have raised serious security concerns in the contemporary cache architectures. To mitigate these attacks, various cache architectures have been proposed. Among these countermeasures, indirect eviction cache (IE-Cache) has shown a high security against conflict-based cache side-channel attacks and faster execution time (Mukhtar et al., 2020). It introduces the cache line relocation and multi-indexing mechanism to generate a sequence of random cache conflicts on one memory access, and the cache line is evicted as a result of the last cache conflict. This introduces the non-evicting cache conflicts, which has shown impractical to learn by the attacker using existing and expected eviction set profiling techniques. However, this incurs high energy consumption because of the cache lines relocation. In this paper, we provide a solution for the high energy utilization problem so that IE-Cache can provide prevention against eviction-base cache side-channel attacks along with low energy consumption. We have observed that two types of relocation are happening in IE-Cache, i.e., tag lines and data lines relocation. We have replaced the direct mapping of tag-to-data lines with pointers-based mapping. This mapping allows for the relocation of tag lines only. This reduces the relocation overhead and energy consumption along with the same security level against conflict-based cache side-channel attacks as IE-Cache. The contribution of this paper are mentioned below:

- A solution is proposed for low energy utilization in IE-Cache by adding a pointer in tag-store towards data-store.
- The performance of PIE-Cache is analyzed experimentally using gem5 simulator and compared with IE-Cache and set-associative cache.
- Security performance is also analyzed and as expected, the security performance of IE-cache remains intact in PIE-Cache.



Javed, S., Mukhtar, M., Bhatti, M. and Gogniat, G.
Novel Design for IE-Cache to Mitigate Conflict-based Cache-side Channel Attacks with Reduced Energy Consumption.
DOI: 10.5220/0011326700003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 675-680
ISBN: 978-989-758-590-6; ISSN: 2184-7711

- Energy utilization and cache performance are evaluated on gem5 simulator using Micro-benchmark. Energy utilization is reduced of about 20% on average for 1MB PIE-Cache with 4 ways and 3 levels.

# 2 BACKGROUND AND MOTIVATION

This section describes the background on conflict-based cache side-channel attacks, IE-Cache and eviction set profiling techniques.

## 2.1 Conflict-based Cache Side-Channel Attacks

In conflict-based cache side-channel attacks, the attacker tries to observe the conflicts on specific cache lines with a victim program. For example, in Prime+Probe attack (Liu et al., 2015), the attacker executes an attack in four steps to observe cache conflicts: profiling of eviction set, initialization of cache state, waiting state, observation of cache state. Using this memory access pattern, the attacker can steal secret information (Liu et al., 2015; Gruss et al., 2016; Yarom and Falkner, 2014).

## 2.2 Indirect Eviction Cache (IE-Cache)

Indirect eviction cache (IE-Cache) belongs to the class of random memory-to-cache mapping based countermeasures. IE-Cache basic organization is like skew associative cache with one way in each skew. To make it resilient against the conflict-based cache attacks, IE-cache proposed multi-indexing and relocation of cache lines for eviction. On cache miss, cache controller selects the cache line from each way and picks one of them at random, let us say this picked cache line as first level candidate. This first level candidate is indexed again for finding further cache lines where it can be relocated in cache. After passing first level candidates via indexing function of each cache way except in which it is already residing, multiple cache lines are selected. Then, again one cache line is selected from these, which we call second level candidate. This is repeated multiple times depending on the design limit. On the last indexing level, one cache line selected at random will be evicted. This selection process is like breadth-first graph walk. Assuming each search step as level and each cache line as graph node, the selection process can be represented in a tree form as shown in Figure 2. Lastly, each cache line selected at the lower level is relocated to the higher level cache lines to accommodate the incoming memory address in the lowest level. In a security perspective, this entire indexing process introduces the cache conflicts that relocates instead of eviction, which are called non-evicting members of eviction set. Finding these non-evicting members are impractical in IE-Cache because of these cannot be directly observed via timing channels attack like Prime+Prune+Probe profiling techniques (Purnal and Verbauwhede, 2019).

## 2.3 Prime+Prune+Probe and Branch-breaking

Prime+Prune+Probe is an advanced profiling technique to learn the eviction sets in existing famous random memory-to-cache mapping countermeasures (ScatterCache (Werner et al., 2019) and Ceaser (Qureshi, 2018)). It has shown that the eviction set in these countermeasures can be learned in few seconds (Purnal and Verbauwhede, 2019). This technique is also like Prime+Probe attack instead to starting with random memory addresses. First, the attacker develops a group of randomly selected memory addresses (say G) and fills the cache with the memory addresses belonging to group G. After that, attacker measures the access latency of the group members by accessing them again. In case of long access latency, attacker excludes that memory address from the group G. The attacker repeats these steps till he finds all memory addresses with the short memory access latency. This reduces the G size, say G'. Lastly, attacker uses this G' group to learn the conflict behaviour with the victim execution. However, he can only learn the conflict behaviour via timing channel, non-evicting members of IE-Cache cannot be learned via Prime+Prune+Probe technique. To learn non-evicting members of the IE-Cache, authors of IE-Cache presented the expected technique which is called branch-breaking technique (Mukhtar et al., 2020). In this technique, the attacker uses same group G' to find the non-evicting members indirectly. In branch-breaking technique, the attacker excludes one memory address from the group and observes the evicting behaviour of already found evicting members. If the attacker finds the evicting members remain in the cache in multiple turns, it means the excluded member is the parent (or non-evicting member) of the evicting member.

# 3 POINTER-BASED INDIRECT EVICTION CACHE (PIE-CACHE)

We observed in IE-Cache that the actual conflict happens at tag-store of cache. Because of the direct mapping between tag-store and data-store, data-store also needs to relocate along with tag on conflict at tag-store level. We find that the relocation of tag-store only is sufficient in IE-Cache to achieve security against conflict-based cache side-channel attacks. On relocation of tag-store, if data remains at same place in data-store, this will not depict an observable timing effect. This is because the both tag and data remains in cache after relocation of tag only.

In order to alleviate the high energy consumption issue of IE-Cache without degrading the security, the insight is to reduce the number of bytes needed to be relocated by introducing the novel pointer-based mapping between tag-store and data-store, we call this proposed cache as Pointer-based Indirect Eviction Cache (PIE-Cache). This allows to relocate tag without moving data lines as pointer defines the respective line in data-store. This replaces relocation of 64 bytes cache line with only few bytes pointer in tag-store. The number of pointer bytes depends on the data-store size Reduction of such number of bytes in relocation impacts the energy consumption. We experimentally evaluated the impact of energy consumption in Section 5 for varied cache sizes over Micro-benchmark.

PIE-Cache threat model is identical to IE-Cache, which we now make explicit. The attacker has user-level privileges only. The attacker can measure memory access latency using hardware timers and also via other methods like threads as counters. We also consider that the memory de-duplication feature is not enabled, which is a fair assumption as it is often disabled in real environment for security reasons(Mukhtar et al., 2019). This also eliminates the shared memory cache-based side-channel attacks such as Flush+Reload and its variants.

Structurally, PIE-Cache is similar to IE-Cache. It also contains skews with one cache way and each cache way is indexed by cryptographic based indexing function. The indexing process uses the multi-indexing and relocation of cache lines in the same way as IE-Cache discussed in Section 2. PIE-Cache implements random replacement policy to select the cache line for relocation on each indexing level. The main difference of PIE-Cache compared to IE-Cache is the mapping mechanism between tag-store and data-store. Each entry of PIE-Cache tag-store is introduced with pointer that identifies the mapping to the entry in data-store. Pointer in each tag-store points to a unique entry in the data-store.

On cache hit, PIE-Cache behaves in the same way as IE-Cache On cache miss, indexing mechanism of PIE-Cache is same as IE-Cache but relocation behavior is different. Assuming that the indexing mechanism has finalized the cache line (or candidate) at each level. Then, candidate at each lower level of indexing will be relocated to location of higher level candidate along with the pointers. The last level candidate, which is selected for eviction, will be evicted but its pointer will be assigned to the tag entry where incoming memory address is being placed.

We explain the operation of eviction process in detail using the example in Figure 1. The example uses a small 3 ways cache with 8 cache lines per way. Letters $A$-$Z$ indicate the memory addresses stored in the tag entry and numbers $P1 - P24$ denote the pointers to data line. Figure 2 shows the eviction process to place a memory block ($Y$) in cache with a two level-of-indexing (Mukhtar et al., 2020). In the first level, indexing function using $Y$ selects a tag-store's entry (or tag line) from each way (let us say, $Y$ selects $Q$, $A$ and $K$). Then, one tag line is selected for relocation at random, let us say $Q$ is selected. In second level-of-indexing, $Q$ is given to indexing function to find line in tag-store where it can be relocated, let us say these lines are $P$ and $Z$. Again, one cache line is selected at random, say $Z$. As this is the last level-of-indexing, $Z$ is evicted. Afterwards, series of relocation are made to accommodate the incoming memory address. $Q$ will relocate to $Z$'s position and similarly $Y$ to $Q$ location. The pointer of each tag entry is also relocated to each new line. The pointer associated with the evicted cache line is assigned to the tag entry belonging to $Y$. Final state of cache after eviction process would contain $Y$ with pointer $P4$ at the location of $Q$ and $Q$ with $P1$ at location of $Z$.

## 3.1 Security Perspective

With the addition of pointers the security perspective remains unchanged in PIE-Cache compared to IE-Cache. 1) It increases the size of eviction set, which increases the effort of the attacker for profiling the eviction set. 2) It adds conflicting members in the eviction set that relocate within the cache instead of eviction, we call these members as non-evicting members of an eviction set. As these do not evict, non-evicting members cannot be observed via memory access latency.

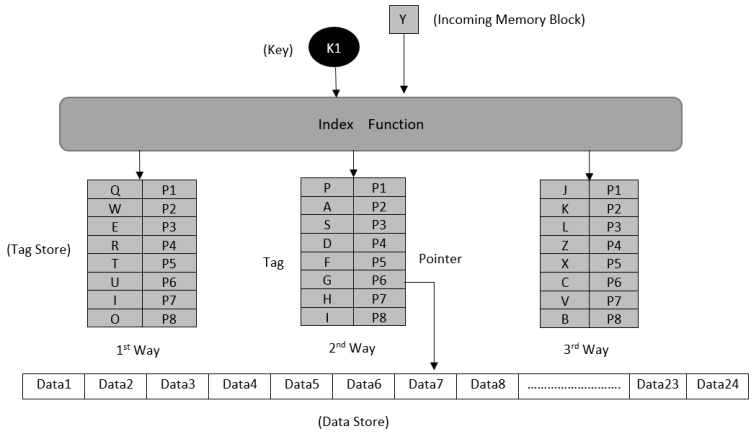


Figure 1: PIE-Cache Architecture.

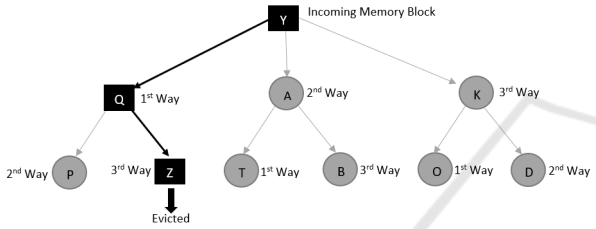


Figure 2: Relocated Candidate Tree.

### 3.1.1 Large Eviction Set Size

Eviction set is the group of memory addresses that are used to generate the cache conflict with the targeted victim's memory address. For instance, in conventional set-associative cache, each memory address is mapped to one cache set due to which the attacker requires to fill all cache lines belonging to a targeted cache set for cache conflict on the targeted victim memory access. Assuming 8 cache lines in cache set, attacker needs 8 memory addresses belonging to cache set to fill it. However, in PIE-Cache, each memory address can be said as member of multiple sets compared to set-associative cache. This is because of the implementation of skew in PIE-cache. The way and skew can be used interchangeably in our case because each skew has one way in PIE-Cache. There is a probability that the victim memory address only conflicts with the attacker memory address in only one way. To guarantee the conflict in that case, if the attacker tries to fill the specific cache way by accessing a specific memory address, random placement makes it uncertain for the attacker about placement of memory address in specific cache way. This forces the attacker to access multiple memory addresses related to specific cache way to guarantee the filling of certain cache way. The number of memory addresses can be seen as popular bin-and-ball problem that how many balls are required to fill the specific bin with a certain confidence. Using this analysis, cache with 2 level-of-indexing and 4 ways, the eviction set size is 768. If number of ways is increased to 8 ways, then the eviction set size becomes 8432. Similarly, if we increase level-of-indexing to 3 for 4 and 8 ways, the eviction sets become 29552 and 253535, respectively.

### 3.1.2 Learning Non-evicting Members

The next difficult task for the attacker is to identify non-evicting members. The attacker does not get access to direct information through timing channel because the non-evicting cache line does not evict due to victim access. However, through indirect timing analysis the attacker can analyze the connection between evicting and relocating cache lines. For instance, Figure 2 shows that if $K$ is not present in cache then it is impossible to evict $D$ from cache. Here $D$ and $K$ are evicting and non evicting cache lines respectively. If attacker is able to find $D$ in profiling step then he can come to conclude that $K$ is also in G' group. In order to find $K$, all the memory addresses of G' are accessed by the attacker. Then the attacker waits for the victim to access, with the purpose that if randomly selected address is $K$ then cache line $D$ will not evict. To determine if eviction occurred or not, the attacker again accesses $D$ during its time measurement. The attacker can monitor that, if $K$ is not a selected candidate to be evicted then $D$ will be evicted multiple times. If the attacker determines that selected member has lowered the eviction chances of $D$ in numerous trials, then he predicts that the selected member is $K$.

According to (Mukhtar et al., 2020), $2^{27}$ years are required by attacker to find single eviction set for IE-Cache of 3 levels, $2^{12}$ cache lines and 4 ways. In our case we have discussed 8 ways IE-cache, for which attacker will require far more time. To notice the evic-

tion behavior on next selected candidate, the attacker needs to flush and fill members of G' again. It is difficult to place non-colliding members in cache because two members can collide in any other replacement if not in first. Random replacement policy changes the placement of members for every access, even in the case of same memory access. Therefore, multiple accesses of all group members is required to measure access latency. If any access comes up with prolonged latency then he has to iterate the action of flush and filling again with the purpose to avoid self-eviction of group members.

### 3.2 Experimental Evaluation

This section first presents the security results of PIE-Cache by showing the work effort required to learn the non-evicting members. Lastly, performance results are presented.

## 4 SECURITY EVALUATION

We have built the python model of PIE-Cache to experimentally verify the attacker efforts to profile the eviction set, as mentioned in (Mukhtar et al., 2020). Following assumptions have been taken during the experiment. Firstly, cache uses random replacement policy and fills the invalid cache line first. Second, keys of indexing function are created using random function and remained same in whole experiment. Third, pointers are also assigned randomly to each entry in tag using *randint* method of random library of python. Lastly, only victim and attacker processes are executing, which favors the attacker in terms of less noise.

We have used Prime+Prune+Probe technique to learn the last level members of eviction set. To find the non-evicting members of the eviction set, break-branch technique has been executed on the group used in Prime+Prune+Probe technique. We have averaged the number of memory accesses required by the attacker to find 1000 evicting and non-evicting members. Then, this average number of memory accesses is multiplied with the number of evicting and non-evicting members required in the eviction set discussed in Section 3.1.1. To calculate the time, we have used the following time for each event: cache hit 9.5 ns, cache miss 50 ns, cache flushing 3.6 ms, victim process execution time 0.5ms. Table 1 shows the results of the experiment in terms of number of memory accesses and time needed to learn an eviction set in PIE-Cache having 4 ways. In Table 1 CL is number of cache lines, L is number of relocation levels, G is group size selected, G' is group in which members do not collide, $n_{rmv}$ is turns required to determine non-colliding members, $n_{pl}$ is number of turns required to load non colliding members in cache, $avg_v$ is average number of victim accesses and $avg_a$ is average number of attacker accesses for victim access.

Results in Table 1 show the results with selection of varied sizes of groups G. For small sizes, it shows that the number of memory accesses and time are increased. This is because the small group size decreases the probability of collision and attacker needs to repeat multiple times to occur the cache conflict with the victim. Inversely, in case of large G, the number of memory accesses and time is reduced but still impractical. The reason of impractical time is because of the self collisions among the members of large G that still remain even after the attacker completely pruned the G. This is because the attacker has only pruned in the one way of G placement in the cache but there is so many possible combinations of placement. To place the pruned in a way such that no self conflicts happen takes longer time. These results approximately match with the results of the IE-cache of same configuration, mentioned in research work of IE-Cache (Mukhtar et al., 2020). Therefore, PIE-Cache provides same protection against Prime+Probe attack as of IE-Cache.

## 5 PERFORMANCE EVALUATION

For performance evaluation, PIE-Cache model was developed on *gem5* simulator. We have modeled a 2 level cache where L1 has split data cache (64 KB) and instruction cache (16 KB). L2 cache is a shared cache of size 1 MB. Each entry of tag-store and data-store is of 44 and 64 bytes respectively. Tag-store entry consists of 4 bytes pointer. Energy consumption is evaluated of PIE-Cache and IE-Cache having 4 ways. This energy consumption is normalized over energy consumption of set-associative cache with same configurations as PIE-Cache and IE-Cache. 40 workloads of Micro-benchmark are used to evaluate the energy consumption. Figure 3 presents the result for normalized energy consumption of PIE-Cache over IE-Cache having 4 ways.

In Figure 3, as PIE-Cache energy consumption is normalized over IE-Cache energy consumption, bar less than 100% indicates the less energy consumption compared to IE-Cache. Results in this figure show that the energy consumption for 4 ways PIE-Cache is reduced by 20.8% on average compared to IE-cache. This reduction is because of the elimination of energy to read and write data entries during the cache miss in

Table 1: Time Calculation for Building Eviction Set.

| CL | L | G | G' (k) | $n_{rmv}$ | $n_{pl}$ | Evicting | | | Non-Evicting | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $avg_v$ | $avg_a$ | Time(hr) | $avg_v$ | $avg_a$ | Time(hr) |
| $2^{12}$ | 3 | 16 | 13534 | 161 | 59 | 1.17E7 | 5.83E13 | 302 | 5.84E17 | 1.24E17 | 6.82E12 |
| | | 15 | 13659 | 147 | 88 | 1.52E7 | 6.76E13 | 1254 | 9.95E18 | 3.06E21 | 1.41E13 |
| | | 14 | 13311 | 151 | 43 | 3.25E7 | 2.03E13 | 1379 | 2.16E18 | 2.74E18 | 2.56E12 |
| | | 13 | 13308 | 27 | 31 | 5.93E7 | 8.36E13 | 1709 | 2.03E18 | 4.02E20 | 2.37E12 |

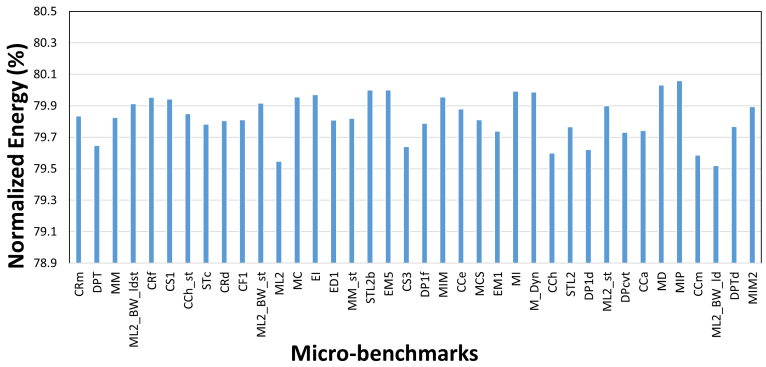


Figure 3: Normalized Energy in 4 ways Last Level Cache.

PIE-Cache compared to IE-Cache. We also observed that the cache miss behavior of both caches are similar in experiments, which indicates that the effect on execution time will be the same in both caches. In experiments, maximum and minimum energy consumption are observed at *MIP* and *ML2* respectively. Overall, the variation in energy consumption among the workloads are slightly different because of same last level cache miss behaviour is observed among the workloads. We have also evaluated the energy consumption for cache sizes with greater number of ways.

## 6 CONCLUSION

This paper proposes a cache which reduces the high energy utilization of IE Cache by introducing pointer-based mapping between tag-store and data-store. It skips the re-locations occurring in data-store entries. It reduces energy utilization of about 20% compared to IE-Cache for Micro-benchmark. Moreover, PIE-Cache replacement process is identical to IE-Cache that increases the eviction set size and introduces the non-evicting members in the eviction set. Because of this, the learning factor of the attacker in IE-Cache remains same as of IE-Cache.

## REFERENCES

Gruss, D., Maurice, C., Wagner, K., and Mangard, S. (2016). Flush+Flush: A fast and stealthy cache attack. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299.

Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. (2015). Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622.

Mukhtar, M. A., Bhatti, M. K., and Gogniat, G. (2019). Architectures for Security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone. In *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, pages 299–304.

Mukhtar, M. A., Bhatti, M. K., and Gogniat, G. (2020). IE-cache: Counteracting eviction-based cache side-channel attacks through indirect eviction. pages 32–45.

Purnal, A. and Verbauwhede, I. (2019). Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache. *ArXiv*, abs/1908.03383.

Qureshi, M. K. (2018). Mitigating conflict-based cache attacks via encrypted-address and remapping. *IEEE/ACM International Symposium on Microarchitecture*, pages 775–787.

Werner, M., Unterluggauer, T., Giner, L., and Schwarz, M. (2019). ScatterCache: Thwarting cache attacks via cache set randomization. In *28th USENIX Security Symposium*, pages 675–692.

Yarom, Y. and Falkner, K. (2014). FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium*, pages 719–732.