Seeds of SEED: H^{2}Cache: Building a Hybrid Randomized Cache Hierarchy for Mitigating Cache Side-Channel Attacks

Xingjian Zhang
Zhejiang University, China
zhxj9823@gmail.com

Ziqi Yuan
Zhejiang University, China
ziqiyuanss@gmail.com

Rui Chang
Zhejiang University, China
crix1021@zju.edu.cn

Yajin Zhou
Zhejiang University, China
yajin_zhou@zju.edu.cn

***Abstract*—Cache side-channel attacks can leak critical information from the target programs. The cache randomization methodology has proven to be an efficient way to mitigate such attacks. However, existing works do not take the cache hierarchy into consideration, failing to address the issue that different levels of caches have different performance and security requirements. In this work, we propose and implement a hybrid randomization scheme, named H^{2}Cache, to mitigate cache side-channel attacks. H^{2}Cache leverages two randomization approaches and applies them to different levels of caches. It strengthens the security of cache modules, while satisfying the performance and resource utilization requirements. Specifically, we design a table-based randomization method for the L1 cache, which uses a hashed virtual index to look up the actual cache set index. The L2 cache in H^{2}Cache takes a computation-based randomization function to calculate the cache set index. We have implemented a prototype of H^{2}Cache and extensively evaluated it using a self-designed RISC-V processor on the FPGA platform. We demonstrate the security of H^{2}Cache through simulated attack programs and quantitative analysis. Meanwhile, the evaluation results of performance and resource utilization have shown its efficacy.**

***Index Terms*—Cache hierarchy, Cache side-channel attacks, Randomization, RISC-V**

I. Introduction

Modern computer systems use cache as an efficient method to bridge the speed gap between CPU and external memory. This has been proven to be an essential module to speed up program execution after decades of evolution. However, most cache designs primarily focus on enhancing their efficiency, but lack concerns about potential cache side-channel attacks. Cache-based side-channel attacks have drawn much attention since their emergence, especially after the disclosure of Meltdown [1] and Spectre [2] vulnerabilities. They utilize micro-architectural features instead of software vulnerabilities to extract critical information. The vulnerabilities affect a wide range of devices, from cloud servers [3], [4] to mobile devices [5], [6]. Besides, cache-based side-channel attacks have the capability to launch powerful attacks such as extracting cryptographic keys [7]–[10], and bypassing ASLR [11]. Therefore, how to mitigate such widespread and influential attacks needs to be a key consideration in contemporary cache designs.

To defeat cache side-channel attacks, researchers have proposed various mechanisms. Some systems [12]–[14] solve the problem from the software's perspective. They are easy to be deployed in existing hardware. But their efficacy still relies on the underlying hardware micro-architecture. In contrast, hardware-based defense mechanisms change the micro-architecture implementation, and thus solve the problem from the basis. In particular, the cache randomization scheme has emerged as a promising approach. It uses a randomization mapping to break the fixed mapping from the request address to the cache set index. Different randomization schemes include table-based randomization and computation-based randomization. The table-based randomization scheme [15]–[17] uses a table to store the mapping from the address to the cache set index, which introduces low performance overhead but suffers from scalability problems due to the storage requirement of the table. The computation-based approach [18]–[20], on the other hand, does not have the burden of the table storage. Using cryptographic algorithms, it generates the cache set index from the address, so it does not suffer from the scalability issue but introduces a higher performance overhead. Therefore, the two schemes bear different merits and can be used in different scenarios. For example, the L1 cache needs to provide low latency with a small size, so it may be better suited for the table-based approach. The L2 cache has a larger size and latency and may match the features of the computation-based approach.

One issue that requires consideration in the current secure cache design is cache hierarchy, as modern processors have widely adopted this structure. Existing works often target a specific level of the cache hierarchy and bring out a proposal that suits for a cache module. However, they leave the interactions between different levels of cache modules out of consideration. The interactions may impact the access behavior of the cache modules and the security attributes of the cache module. Therefore, new defense mechanisms should consider the cache system as a whole, rather than separate modules.

Another thing to be noted is that current works are mainly designed for a generic architecture or specific ISAs like x86 and ARM, leaving RISC-V out of the spotlight. The RISC-V architecture is an emerging open-source ISA [21], [22], which boosts the development of open-source processors and research prototypes. However, recent studies [23]–[25] have shown that the RISC-V architecture is not immune to cache side-channel attacks, as its current design does not consider those issues. Besides, current works mainly use a simulation-based implementation and evaluation environment without fully considering hardware features, especially for the RISC-V architecture. Therefore, how to design an efficient and yet secure cache system for the RISC-V system is still an open question.

In this paper, we propose H^{2}Cache, which leverages both randomization-based mechanisms based on different demands across the cache hierarchy to mitigate cache side-channel attacks. Specifically, H^{2}Cache uses a table-based randomization design in the L1 cache that incurs the lowest latency, and a computation-based randomization scheme in the L2 cache to achieve better scalability. We have implemented H^{2}Cache on a self-designed RISC-V processor, called ZJV, and performed the extensive evaluation. We demonstrated the security of H^{2}Cache through attack testing and security analysis. Meanwhile, the results of performance and resource utilization evaluation have shown its efficacy.

In summary, our work makes the following main contributions.

- We propose a defense scheme against cache side-channel attacks from the perspective of the whole cache hierarchy, including both L1 and L2 caches.
- We present a hybrid randomized cache hierarchy, which leverages the benefits of both table-based and computation-based

randomization schemes to meet security and performance needs.
- We design and implement a cache system, H^{2}Cache, on a self-designed RISC-V processor, and perform extensive evaluation including security, performance, and resource utilization on the FPGA platform.

The rest of the paper is structured as follows. Section II introduces cache side-channel attacks and defines our threat model. Section III introduces our design and implementation of H^{2}Cache. Section IV shows the evaluation results in terms of security, performance, and resource utilization. Section V discusses the limitations and future works of H^{2}Cache, and section VI talks about our related work. Section VII concludes the paper.

II. Threat Model

A. Cache Side-channel Attacks

Cache side-channel attacks utilize micro-architectural features of the cache to infer the access pattern of the target program, which would leak critical information of the program. Prime+Probe attack [4], [8] is a variant of cache side-channel attacks. To successfully launch such an attack, the attacker needs to prepare the cache to a definite state by priming the cache with one's own data. Then the attacker would let the target program execute critical functions which may access the critical data. After that, the attacker measures the access time of different blocks to check if there is a time difference with previous accesses, and misses could be viewed as access to the critical data by the victim. Repeating this process enables the attacker to gain sufficient information about the critical data, which breaks the security boundaries between different entities.

To launch the attack, one key step is to find addresses that fall in the same cache set as the target address. The set of those addresses is called the eviction set. For traditional cache structures, this process would be trivial, as the cache modules directly use part of the address as the cache set index. For a secure cache scheme, the amount of effort to find an eviction set indicates its security.

B. Assumption

This paper targets cache side-channel attacks based on cache set conflicts as illustrated in Section II-A, while other attacks, including transient execution attacks and other micro-architectural attacks, are out of our scope. We make the following assumption about the attacker and the environment.

- The attacker could know the address of the critical information, but cannot access it directly.
- The attacker could access a sufficient range of address space.
- The attacker could measure the latency of cache access precisely without noise interference.
- The attacker could have the awareness of the targeted cache micro-architecture as well as its security mechanism against the attacks, but cannot modify them.

C. Requirement

Besides the assumptions from the attack side, there are also some requirements for the secure cache proposal as the cache modules need to improve the performance of the system and interact with other modules. We list the following requirements for the defense mechanisms.

- **Efficiency:** The introduced cache security mechanism should have minimal impact on the performance of the system.
- **Compatibility:** The security-enhanced cache system should require minimal modifications of other parts of the system.
- **Security:** The security mechanism should mitigate or eliminate potential cache side-channel attacks successfully.

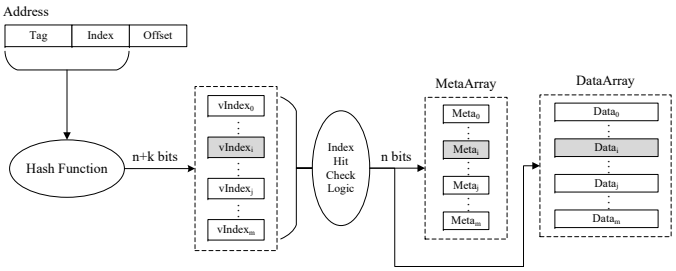


Fig. 1. Table-based randomization design for L1 cache in H^{2}Cache.

III. Design and Implementation

A. Overview

We consider H^{2}Cache with a two-level cache hierarchy. The L1 cache in H^{2}Cache is separated into an instruction cache and a data cache, and the L2 cache in H^{2}Cache is a unified structure. Each cache module is set-associative. This topology is common in processors, while our proposal is not confined to this arrangement. For L1 cache modules, they are the smallest and fastest cache modules in the hierarchy, as the system requires. The L2 cache module, on the other hand, is relatively large and slow. Therefore, different responsibilities of cache modules lead to their different structures.

In general, we use the cache randomization method as an approach to defending against cache side-channel attacks. It breaks the mapping from the requested address to the cache set index, which increases the difficulty of finding the eviction set. To satisfy the requirements listed in section II-C, we use a table-based randomization approach for the L1 cache and a computation-based randomization approach for the L2 cache. This hybrid randomization scheme for the different levels of cache modules makes a balance between efficiency and security requirements. Besides, we also adopt a skewed cache [26] design to further strengthen the security without degrading the performance. H^{2}Cache is compatible with various architectures and requires no modification to software or other parts of the system.

B. Table-Based Randomization for L1 Cache in H^{2}Cache

Fig. 1 illustrates our table-based randomization scheme of L1 cache in H^{2}Cache. Traditional cache modules derive the cache set index directly from the part of the requested address with the width of n, which corresponds to 2^n cache sets in the cache. In our secure L1 cache design, we use all but the offset bits in the address to calculate a virtual index with additional k bits using a hash function, which virtually maps to a cache array with 2^{n+k} entries. The computed virtual index is then matched against each of the 2^n $(n+k)$-bit entries in the virtual index table. If there is a match, then the physical index of the corresponding virtual index entry is the actual index for the address. After this lookup process, the cache module would continue the cache access like the traditional cache. Particularly, if there is no matching virtual index entry in the table, the module would randomly choose an entry for the virtual index. If a conflict in the physical index occurs, the cache line is evicted and the corresponding virtual index entry is updated.

In the implementation, we use an array of registers to store the virtual index table, which can complete the lookup process and continue the cache access quickly. The hash function we adopt in the module is

$$\text{vIndex} = \text{addr}_{upper} \oplus \text{addr}_{lower} \oplus \text{key}.$$

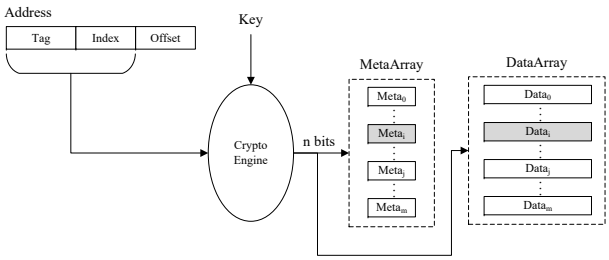


Fig. 2. Computation-based randomization design for L2 cache in H²Cache.

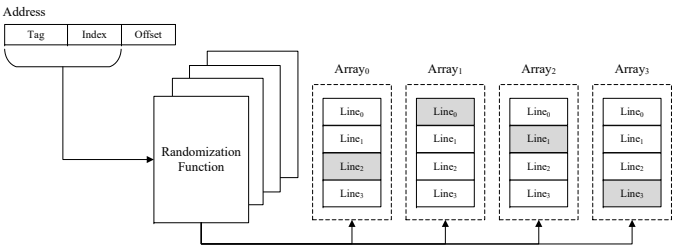


Fig. 3. Skewed cache design in H²Cache.

This function performs the exclusive-or operation with the upper half and the lower half of the address, and the result would be encrypted with an exclusive-or operation using a randomly generated key. We use the configuration of $k = 3$ to make a balance between the storage need and security. The complexity of the hash function and the value of k can be adapted to different performance and security requirements.

This randomization function deployed in the L1 cache satisfies the requirements in section II-C. For the security requirement, this approach clearly breaks the direct mapping from the requested address to the cache set index, so the attacker cannot directly get the index of the target address. Furthermore, this approach would also change the physical index after eviction, which would make eviction set construction even harder. For the efficiency requirement, this scheme would not induce extra latency as the lookup process can finish quickly. Besides, this approach could, in theory, increase the utilization rate of the cache as it can reduce the possibility of cache conflicts.

C. Computation-Based Randomization for L2 Cache in H²Cache

Fig. 2 shows our computation-based randomization mechanism of L2 cache in H²Cache. As the table-based design shown in section III-B may not be suitable for larger caches like the L2 cache, we use the computation-based approach to offer the security guarantee and satisfy the resource and latency requirements. In our design, we compute the actual index for a requested address using a keyed cryptographic function. This function takes the key and the truncated address (only tag and index) as the input, and the result would be truncated to serve as the access index. Cache metadata and data would be fetched according to the produced index. All other components within and outside the cache module can be left unmodified.

In our implementation, we use $QARMA_7$-64-σ2 as the cryptographic function to determine the cache set index. $QARMA$ [27] is a family of tweakable block ciphers dedicated to high-performance hardware design. Compared to traditional block ciphers like AES, $QARMA$ features its acceptable hardware and performance overhead and yet strong security guarantee. We take the truncated address as its input plaintext with a randomly generated 128-bit key and the zero tweak. The lower bits from the ciphertext are used as the cache set index.

This approach clearly strengthens the security of the cache module as it uses a cryptographic result instead of a plaintext as the cache set index. From the performance perspective, the process would not complex the key management for the OS, but may cause the degradation of the performance since completing the cryptographic calculation requires some cycles, which is four cycles as to our implementation. However, additional latency can be partially hidden by other operations within the cache module, as previous work [20] has stated. For example, both $QARMA$ cryptographic function and cache operations can be implemented in a pipelined style, which can overlay several inflight cache requests, and thus lower the overall latency.

D. Skewed Cache in H²Cache

To further enhance the security of the cache system and make use of cache set associativity, we use the design of skewed cache to strengthen the randomization functions, as Fig. 3 shows. The skewed cache approach would complicate the cache set mapping as the number of possible mappings is increased multiple times, and it would not cause performance degradation. This design can thus further avoid index conflicts and reduce the chance of linking cache set index to the original index. Built upon the randomization functions introduced in section III-B and III-C, each way in the cache module can produce different indices even for the same address. For the table-based approach in section III-B, each way manages a distinct virtual index array and applies a different key to the hash function. For the computation-based approach in section III-C, each way uses a different key for the cryptographic function. When there is a need for cache line replacement, we use the random replacement policy for simplicity and applicability.

Using the skewed cache can offer comparable or even better efficiency while hardening its security. With this approach, the rate of conflict misses in the cache might be lower, which boosts the performance of the cache.

IV. Evaluation

In this section, we first set up the experimental environment in section IV-A including our RV64 processor with other hardware and software tools. Then we present the evaluation of H²Cache from section IV-B to IV-E in the following aspects:

- Security analysis through the quantitative approach.
- Security testing using a Prime+Probe-like attack.
- Performance test on the FPGA platform.
- Resource utilization evaluation by Vivado.

A. Environment Setup

We evaluate the performance and resource utilization of our implementation in an FPGA environment. We integrate our cache design within ZJV, our self-designed 64-bit RISC-V processor capable of booting Linux distributions. The processor and peripherals, including DDR, UART, boot ROM, and SD card, form an SOC system, which is synthesized and implemented by Vivado on Digilent Nexys A7 evaluation board with Xilinx Artix-7 Series FPGA. Fig. 4 is the illustration of the greeting logo of ZJV and the initial booting process of RISC-V Linux. The detailed configurations for our experiment are shown in Table I.

We use UnixBench [28] as our benchmark for performance profiling. We run it with the default configuration atop the Debian OS

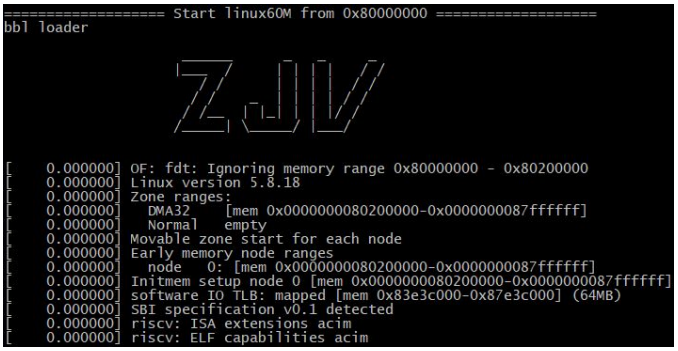


Fig. 4. ZJV boots Linux 5.8.18.

[29]. The UnixBench test suite consists of 12 test programs, which cover 2 computational tests and 10 operating system tests. As for resource utilization analysis, we use the utilization and power reports generated from Vivado 2019.2.

TABLE I
CONFIGURATIONS FOR THE EXPERIMENT

Component	Configuration
OS	Linux 5.8.18, compiled by clang with RV64IMAC and LP64
UnixBench	version 5.1.3, compiled by gcc with RV64GC
FPGA	Nexys-A7 board, xc7a100tcsg324-1, 60 MHz, Vivado 2019.2
Processor	ZJV, RV64 IMAC support, 10-stage pipeline
L1 cache	4/8/16 KB, 2/4 ways, 32B line size
L2 cache	32KB, 2 ways, 64B line size

B. Security Analysis

For the L1 cache module, we assume that its index length is n bits, with additional k bits to form the virtual index, and the cache module has n_{way} ways of associativity. In our secure cache design, there are 2^{n+k} possible virtual index numbers, and each way may have a different virtual index for each address, so there would be $2^{(n+k)\cdot n_{way}}$ different combinations of virtual indices, which may be larger than the address space. Therefore, finding two addresses with the same index mapping would not be possible. Even if the attack does not need to find such an address that is fully congruent with the target address, it would still be hard enough to find addresses that could form an eviction set. For each way in the cache, the possibility of conflicts between two addresses would be 2^{-n}, so there is a possibility of

$$p = 1 - (1 - 2^{-n})^{n_{access}}$$

to evict the target address after n_{access} accesses. Therefore, there needs

$$n_{access} = \frac{\log(1-p)}{\log(1-2^{-n})}$$

accesses to evict an address from one way with the confidence of p, and $n_{access} \cdot n_{way}$ accesses to evict the target address from the cache. Still, the attacker may not be able to infer the access pattern of the target program as the target address may lie in a different cache set after refilling.

For the L2 cache module, we use $n^{'}$ to denote its index length, and $n^{'}_{way}$ for its set associativity. Following a similar process, it can be deduced that the number of accesses to the L2 cache required to evict an address from one set with the confidence of $p^{'}$ would be

$$n^{'}_{access} = \frac{\log(1-p^{'})}{\log(1-2^{-n^{'}})},$$

so there would be $n^{'}_{access} \cdot n^{'}_{way}$ in total to evict an address from the L2 cache. Combining with the result for the L1 cache, there would be $n_{access} \cdot n_{way} \cdot n^{'}_{access} \cdot n^{'}_{way}$ data accesses to evict the target address from the cache system. Even if the target address would be refilled to the same cache set, the cost to detect it would be excessive.

C. Security Testing

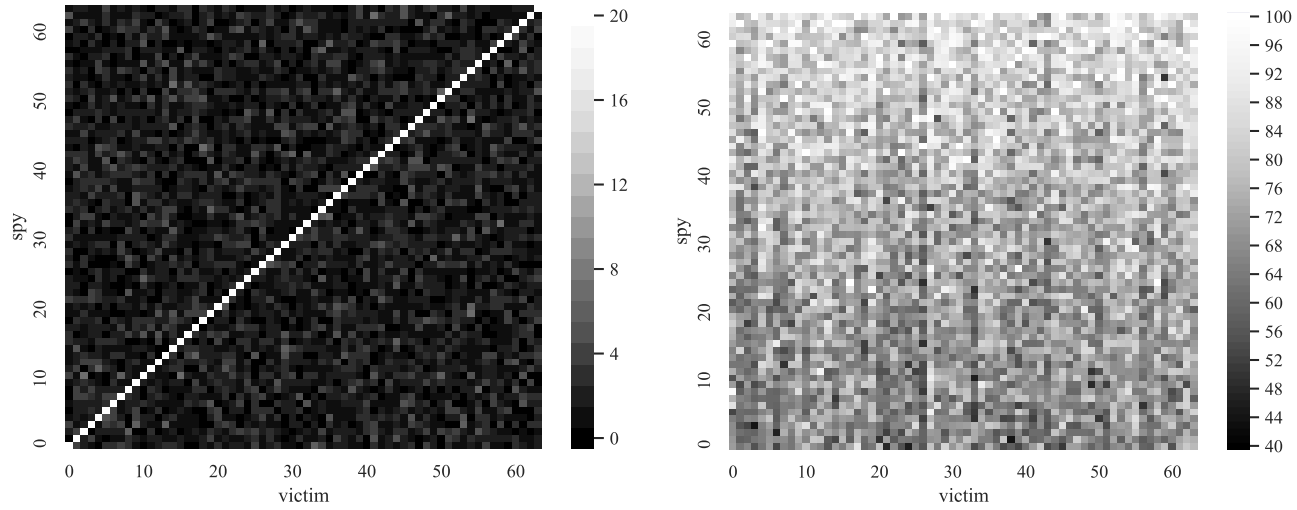
We can also demonstrate the security of our hybrid randomized cache system through a security test. Our test program consists of a spy thread and a victim thread. The spy thread launches a simplified Prime+Probe attack to monitor the cache access pattern of the victim thread with the following steps. In the Prime step, the spy fills all sets in the data cache with its own array of data. In the Access step, the spy sleeps and the victim runs and accesses the critical data, which evicts the corresponding data from the spy in the data cache. In the Probe step, the spy wakes up to access the primed data and measure the data read latency for each set. We set a threshold for the access time of each cache set, and one access with the above-the-threshold latency would be counted as a cache miss, which may be a result of victim accessing. We also add some random data access in the victim thread to emulate a real-world scenario more approximately. To measure the result more precisely and test our design more concretely, we repeat the process for a number of iterations and put the critical data in different locations.

We run our testing program on both traditional design and our secure cache design. Fig. 5 shows the resulting accessing heat map for two designs. The horizontal axis represents the cache set index corresponding to the target address, and the vertical axis represents the number of misses during the probing process for each cache set. Lighter colors stand for more cache misses in the corresponding cache set, which may indicate a higher chance of cache eviction. Fig. 5a has a clear light diagonal, enabling the attacker to correlate the cache misses with the victim access pattern. In contrast, Fig. 5b does not show a similar pattern, and thus it is a demonstration that our secure cache design could mitigate cache side-channel attacks.

D. Performance

We test the performance of our design with different cache configurations shown in Table I, where the size and set associativity of the L1 cache vary. The traditional cache scheme with the same configuration serves as our baseline, and our secure cache design with the randomized L1 and L2 caches compares with it. Fig. 6 shows the performance overhead of our design, and each bar represents the relative performance slowdown compared to the baseline. The result shows that the overall performance of the secure cache is slightly lower than the traditional cache, with an average of 13.4% overhead across all configurations. Among all configurations, the 2-way 8KB cache has the worst performance degradation, which might be a result of cache thrashing. Apart from that specific case, the average overhead is 10.7%. It should also be noted that as a result of the low frequency of FPGA, the UnixBench result would be low in terms of the performance index, which may influence the resulting overhead.

For each test case in UnixBench, the overall trends remain for separate configurations, while the impacts of applying our secure cache design are different. In specific scenarios, however, our proposed cache design can even boost the performance. For example,



(a) Traditional cache (b) Secure cache

Fig. 5. Result for security testing.

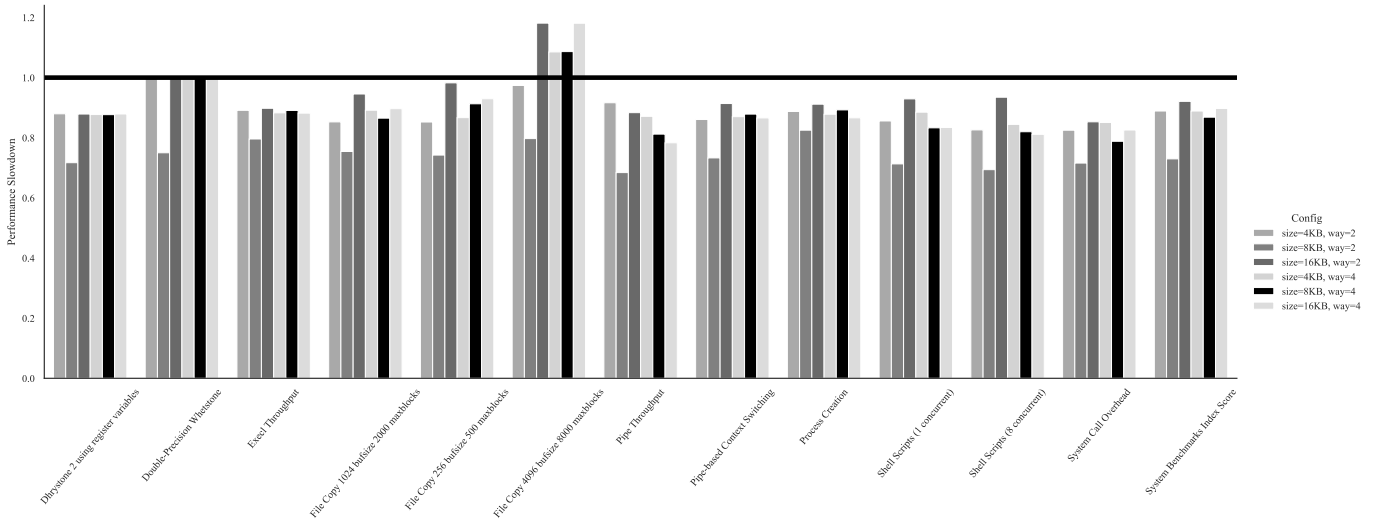


Fig. 6. Performance overhead of Unixbench.

for the 4-way cache with the size of 16 KB, its performance on `File Copy 4096 bufsize 8000 maxblocks` is 18% better than the baseline.

E. Resource Utilization and Power Consumption

We use Xilinx Vivado to evaluate the resource utilization and power consumption of `H`^{2}`Cache`. On Xilinx FPGAs, slice LUTs and slice registers are basic building blocks. They correspond to the combinational logic usage and the sequential logic consumption in digital circuits accordingly. It is worth mentioning that both slice LUTs and registers can be proportionally mapped to ASIC gates [30].

Fig. 7 and 8 show our design's utilization of slice LUTs and slice registers. For the L1 cache, it requires fewer slice LUTs and slice registers with smaller cache size configurations, while more on larger caches. Besides, 4-way associative cache configurations have lower overheads compared to the 2-way ones. The reason is that using virtual index registers would simplify part of cache control logic, but requires more registers to save them. For the L2 cache, the slice LUTs and the slice registers are heavily required for our design, because calculating the $QARMA$ function would require a large amount of combinational and sequential logic. The overall average overheads of the two resource types are 9.5% and 19.6% respectively.

Fig. 9 illustrates the power consumption of our design. The result shows that our design would not introduce much power consumption to the system, with an average of 10% overhead. The L1 cache would even consume less power under some configurations. The L2 cache module consumes more power because of the $QARMA$ cryptographic function as well.

V. Current Limitations and Future Works

Our current design is based on a single-core processor with a two-level cache hierarchy, which may represent a common design in an embedded system. However, in a modern processor with higher efficiency demands, there would be more processors with

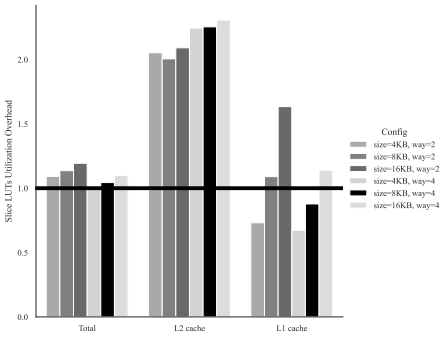


Fig. 7. Utilization of slice LUTs.

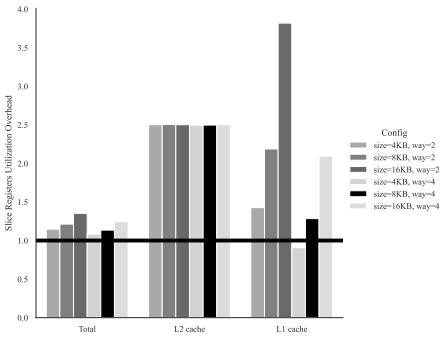


Fig. 8. Utilization of slice registers.

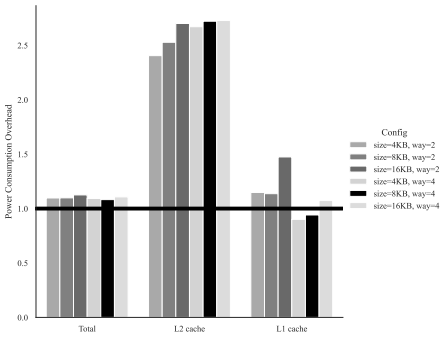


Fig. 9. Power consumption.

more complex topologies, which is out of this work's scope. Our design may also be tailored and optimized for specific hardware environments with various requirements. To extend our work to a multi-core system with a shared last-level cache, we need to consider the necessary modifications to the cache coherence protocol when applying our randomization strategies. This may, in turn, require some changes to the whole cache hierarchy.

Besides, the interactions between different levels of caches could be exploited more extensively as well. Our work has shown that the deployment of a randomization mechanism in the upper-level caches would strengthen the security of the cache modules in the lower level. Applying a more comprehensive and unified approach to the whole cache hierarchy can harden the security and even reduce the overall performance overhead. For example, the lower-level cache modules could utilize the randomization information from the upper-level caches to complicate the cache mapping. However, to implement such a mechanism, one should also consider the cache coherence problem within the same level of modules, as well as the lookup and replacement policies. Meanwhile, this additional feature may complicate the control logic in the cache system as well, which would incur higher performance and utilization overhead.

Our work is based on the RISC-V architecture but has not fully utilized possible RISC-V features for cache security. We may aid our randomization design with features like virtual memory management, PMP, memory fence, or other features for isolation and cache operations. These features would strengthen the security guarantees of the design without the need to modify the upper software.

VI. Related Work

Our work uses cache randomization techniques to mitigate cache side-channel attacks, which can be categorized into the table-based approach and the computation-based approach. The table-based approach needs a lookup table to translate from the address to the cache set index. RPcache [15] uses a permutation table to randomize the index mappings. Newcache [16] uses a similar indirect mapping as in our work, but it may still suffer from cache side-channel attacks. Random Fill Cache [17] places the cache block within a neighboring region and thus provides higher randomness. The computation-based approach uses a hashing function to calculate the cache set index. CEASER [18] uses the LLBC-encrypted address within the module and decrypts the address to make requests to other modules. CEASER-S [19] adds the skewed cache design based on CEASER. ScatterCache [20] randomizes both index and way selection. These techniques mainly apply to the last-level caches as the hashing function may require several cycles of computation, which is not acceptable for the upper-level caches. Besides, previous works mainly use simulators to implement and evaluate their design, while our design is implemented in RTL and evaluated on the FPGA platform, which is more practical and convincing.

Cache partitioning is another approach to prevent cache side-channel attacks. It divides the cache storage into separate regions and allocates them to different processes, which isolates the target program from the attacker. One previous work [31] proposes a partitioned cache design targeting block ciphers. Another work [32] uses cache partitioning based on page coloring, and DAWG [33] provides stronger isolation with a notion of protection domains. These techniques offer strong security guarantees but may have high performance overhead due to hardware constraints.

In the RISC-V community, many works have been done to prevent cache side-channel attacks. One work [34] on the issue applies cache randomization methods to cache modules, but our hybrid approach can provide better security guarantee and performance. Besides, a RISC-V task group is working on Cache Management Operations [35], which can assist the prevention of cache side-channel attacks.

VII. Conclusion

In this paper, we propose a hybrid randomization design on the whole cache hierarchy and implement `H²Cache` to mitigate cache side-channel attacks. `H²Cache` uses a hybrid table-based randomization and computation-based randomization for different levels in the cache hierarchy. It works on our self-design processor and its security is demonstrated via a quantitative analysis and attack testing. The extensive evaluation also includes performance and resource utilization. The results have shown that `H²Cache` is efficient and has minimal overhead.

References

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 1–19.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 199–212. [Online]. Available: https://doi.org/10.1145/1653662.1653687

[4] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*. IEEE, May 2015, pp. 605–622.

[5] R. Spreitzer and T. Plos, "On the applicability of time-driven cache attacks on mobile devices," in *Network and System Security*, J. Lopez, X. Huang, and R. Sandhu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 656–662.

[6] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on ARM and their implications for android devices," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 858–870. [Online]. Available: https://doi.org/10.1145/2976749.2978360

[7] D. J. Bernstein, "Cache-timing attacks on AES," 2005.

[8] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology – CT-RSA 2006*, D. Pointcheval, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–20.

[9] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[10] O. Aciiçmez, "Yet another microarchitectural attack: Exploiting i-cache," in *Proceedings of the 2007 ACM Workshop on Computer Security Architecture*, ser. CSAW '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 11–18. [Online]. Available: https://doi.org/10.1145/1314466.1314469

[11] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 368–379. [Online]. Available: https://doi.org/10.1145/2976749.2978356

[12] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology – LATINCRYPT 2012*, A. Hevia and G. Neven, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 159–176.

[13] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, C. Clavier and K. Gaj, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17.

[14] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 41–46. [Online]. Available: https://doi.org/10.1145/2046660.2046671

[15] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 494–505. [Online]. Available: https://doi.org/10.1145/1250662.1250723

[16] ——, "A novel cache architecture with enhanced performance and security," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '41. USA: IEEE Computer Society, 2008, p. 83–93. [Online]. Available: https://doi.org/10.1109/MICRO.2008.4771781

[17] F. Liu and R. B. Lee, "Random fill cache architecture," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '47. USA: IEEE Computer Society, 2014, p. 203–215. [Online]. Available: https://doi.org/10.1109/MICRO.2014.28

[18] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '51. IEEE Press, 2018, p. 775–787. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00068

[19] ——, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 360–371. [Online]. Available: https://doi.org/10.1145/3307650.3322246

[20] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "Scattercache: Thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 675–692. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/werner

[21] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual, volume I: User-level isa, version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[22] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual volume II: Privileged architecture version 1.9," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129, Jul 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[23] A. Gonzalez, B. Korpan, J. Zhao, E. Younis, and K. Asanovic, "Replicating and mitigating spectre attacks on an open source RISC-V microarchitecture," in *Third Workshop on Computer Architecture Research with RISC-V (CARRV 2019)*, 2019.

[24] A. Gonzalez, B. Korpan, E. Younis, and J. Zhao, "Spectrum: Classifying, replicating and mitigating spectre attacks on a speculating risc-v microarchitecture," 2019. [Online]. Available: https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F18/projects/reports/project4_report.pdf

[25] A.-T. Le, B.-A. Dao, K. Suzaki, and C.-K. Pham, "Experiment on replication of side channel attack via cache of RISC-V berkeley out-of-order machine (BOOM) implemented on FPGA," in *Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020)*, 2020.

[26] M. Spjuth, M. Karlsson, and E. Hagersten, "Skewed caches from a low-power perspective," in *Proceedings of the 2nd Conference on Computing Frontiers*, ser. CF '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 152–160. [Online]. Available: https://doi.org/10.1145/1062261.1062289

[27] R. Avanzi, "The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," *IACR Transactions on Symmetric Cryptology*, pp. 4–44, 2017.

[28] K. Lucas, "Kdlucas/byte-unixbench." [Online]. Available: https://github.com/kdlucas/byte-unixbench

[29] "Debian – The Universal Operating System." [Online]. Available: https://www.debian.org/

[30] "7 series FPGAs configurable logic block," https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2016.

[31] D. Page, "Partitioned cache architecture as a side-channel defence mechanism," 2005, page@cs.bris.ac.uk 13017 received 22 Aug 2005. [Online]. Available: http://eprint.iacr.org/2005/280

[32] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, ser. CCSW '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 77–84. [Online]. Available: https://doi.org/10.1145/1655008.1655019

[33] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '51. IEEE Press, 2018, p. 974–987. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00083

[34] M. Doblas, I.-V. Kostalabros, M. Moreto Planas, and C. Hernández Luz, "Enabling hardware randomization across the cache hierarchy in linux-class processors," in *Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020)*, 2020, pp. 1–7.
[35] "Riscv/riscv-CMOs," RISC-V. [Online]. Available: https://github.com/riscv/riscv-CMOs