Caching and Prefetching for Improving ORAM Performance

Naohiro Hayashibara
Faculty of Information Science and Engineering
Kyoto Sangyo University
Kyoto, Japan
naohaya@cc.kyoto-su.ac.jp

Kazuaki Kawabata
Faculty of Information Science and Engineering
Kyoto Sangyo University
Kyoto, Japan
g2154686@cc.kyoto-su.ac.jp

***Abstract*—Oblivious RAM (ORAM) provides secure data storage by hiding memory access patterns. This technique is fundamental to numerous secure applications. Despite extensive endeavors, while established ORAM schemes offer optimal asymptotic complexity, their actual costs remain excessively high for many compelling applications. This arises due to ORAM's access strategy, which involves interacting with multiple locations to retrieve a single required data block. While existing approaches often focus on optimizing the locations of data, queries, and memory accesses through architectural improvements, there is less focus on data caching and prefetching. Through our experiment, we identify that data caching is essential for improving the latency of ORAM and show that data prefetching can contribute to further improvement of ORAM performance. We then discuss the integration of ORAM with the existing caching and prefetching techniques.**

***Index Terms*—ORAM, security, privacy, storage, caching, prefetching**

I. Introduction

Many cloud platforms, such as Amazon AWS, Microsoft Azure, and Google Cloud, enable users to upload their data to the cloud and provide services utilizing the outsourced data. Developing applications using cloud services offers a cost-efficient approach to scalability and reliability. However, the necessity of accessing data remotely introduces trust concerns, and the significant risk of exposing sensitive information poses a considerable challenge.

In order to protect sensitive information on a cloud, encrypted databases can be used to ensure its privacy and security. Even with the data encryption in place, the access patterns of users' queries and operations remain vulnerable to data privacy breaches and the inadvertent leakage of sensitive information [1]. This is due to the fact that curious observers can still exploit these patterns to extract valuable insights, posing a threat to the overall security of the data.

Oblivious Random Access Memory (ORAM) [1] is a privacy-enhancing technology designed to protect the confidentiality of data stored in computer systems. Traditional storage systems expose data access patterns, allowing adversaries to infer sensitive information. ORAM, however, provides a powerful solution by obfuscating the access patterns, making it extremely difficult for adversaries to determine which data is being accessed and in what order. This is achieved by introducing dummy accesses and encrypting the data so that the actual access pattern is indistinguishable from random access.

The concept of ORAM has found numerous applications in various fields, including cloud computing, secure multi-party computation, and privacy-preserving machine learning. By utilizing ORAM, individuals and organizations can enjoy stronger privacy guarantees and protect their valuable data from the adversary.

It is essential to keep in mind that using ORAM protocols can improve privacy and security, but they come with a significant overhead that may not be suitable for most of applications. In particular, traditional ORAM setups such as PathORAM [2] require excessive bandwidth, overhead, and storage to ensure efficient memory access.

Existing ORAM implementations address this problem by optimizing primitives and/or architecture [3]–[10].

Although numerous efficiency improvements for ORAM have been proposed, little focus has been placed on utilizing the Stash for caching and prefetching data. The findings presented in [8], [10], [11] indicate that the stash hit ratio greatly influences access latency. By leveraging the stash as a cache and incorporating data prefetching, there is potential to significantly enhance ORAM's efficiency and performance.

Initially, this paper presents experimental findings regarding the effectiveness of caching within ORAM. Subsequently, we introduce the architecture of the ORAM controller, which incorporates caching and prefetching to enhance ORAM's performance, and we explore prefetching mechanisms tailored to specific applications. This controller also plays a role in masking access request pattern dispatched to the server by issuing requests for data blocks alongside requests for dummy blocks. The design objective of the ORAM controller is to enhance performance through the use of stash for caching and to bolster security by equalizing the access frequency across individual blocks, thereby obscuring the real frequency of access.

II. Background and Related Work

We describe the fundamental assumption and underlying issue concerning the concealment of access patterns. This assumption revolves around the dynamic interplay between a client and a server, particularly in the context of their deployment on a cloud platform. The client, equipped with a relatively small yet secure memory capacity, called *stash*, seeks to securely store and retrieve data from a cloud storage system that is significantly larger in size but lacking in trustworthiness. Various data storage structures have been proposed, yet this paper concentrates on ORAM, which employs a tree structure for data storage [4], [7], [9]–[11]. The key focus here lies in safeguarding the privacy of the data throughout this process, a task that is full of challenges.

A. Threat Model

We consider the "semi-honest" assumption in a cloud server, which is consistent with existing work [4], [9], [12], and a trusted client. The server is treated as both a passive and active adversary. First, the server or cloud platform underlying it might passively observe how the server interacts with the client to learn information about the user's encrypted data. Second, it may additionally try to tamper or destroy data stored in the storage to influence the program's outcome and/or learn information about the encrypted data.

B. Basic Concept of ORAM

ORAM is a cryptographic protocol created to entirely obscure data access patterns. Within ORAM, a singular sequence of data requests $\overrightarrow{x} = (a_1, a_2, \ldots, a_n)$ of length n is converted into a series of data accesses $S(\overrightarrow{y})$, referred to as an ORAM request sequence. The ORAM protocol is considered to be secure on the premise that if any two ORAM request sequences $S(\overrightarrow{x})$ and $S(\overrightarrow{y})$ converted from $\overrightarrow{x}$ and $\overrightarrow{y}$ with the same length are computationally indistinguishable [13].

C. Existing ORAM Implementations

Many approaches in ORAM have been proposed so far. Tree-based ORAMs are common ORAM implementations, thus, we focus on them in this section.

Ring ORAM [7] introduced a novel approach that reduces overall and online bandwidth requirements by putting metadata for each bucket. It improved the efficiency of data blocks in a bucket and made the bandwidth independent of the ORAM bucket size.

PrORAM [11] introduced a dynamic prefetching mechanism that operates on a "superblock", defined as a group of blocks allocated to the same path. This approach showed that dynamic prefetching combined with caching in ORAM can markedly enhance performance. The emphasis of this work is on the critical roles of caching and prefetching within ORAM. Nevertheless, the underlying assumption differs from our work, as it treats ORAM as a form of local memory.

Hitchhiker ORAM [8] introduces dynamic scheduling to significantly improve ORAM efficiency. This method enables ORAM to process multiple requests along the same path and execute path writing in batches, which minimizes unnecessary memory access. This work underscored the significance of caching within ORAM but did not provide a prefetching strategy/mechanism to enhance the cache hit ratio.

D. Caching and Prefetching

Caching and prefetching are well-developed techniques used to improve memory access performance by predicting future memory accesses and bringing the required data into the cache before it is actually needed. To effectively boost the hit ratio via data prefetching, it is essential to analyze and take into account the data access pattern of the specific application in question. By recognizing and understanding how the application accesses data, developers can implement prefetching strategies that are tailored to improve efficiency and optimize performance.

One of the recent advances is the prefetching based on the analysis of the access pattern extracted from the application binary [14]. It classifies the extracted memory access into several access patterns. This optimizes the prefetching strategy and improves caching efficiency. The prefetcher for big data analysis, called Bingo [15], is based on the TAGE [16] predictor, which uses the multiple cascaded history tables. This has notably enhanced the efficiency of spatial data prefetching and significantly boosted overall performance.

Machine learning stands out as a viable approach for data prefetching across various applications. This approach involves learning the pattern of data access to develop a predictive model that can forecast which data will be accessed next. Several prefetching mechanisms utilizing machine learning have been proposed so far [17]–[20]. Voyager [20] is a neural network-based data prefetcher based on address correlations in addition to the delta correlations. It showed significant precision compared to the existing neural network-based solutions. Peled et al. utilize a fully-connected feedforward network (FFN) to enhance the precision of their data prediction efforts [18].

III. Experimental Result on the Efficiency of Caching

We have implemented the modified version of Ring ORAM [7] in Java to measure the impact of caching in the stash on the read operation of ORAM. We altered the eviction mechanism so that certain data blocks are not evicted from the stash, allowing us to measure the latency and throughput of the ORAM read operation.

We have conducted the experiment with the server and the client of Ring ORAM [7] located in the same LAN. We use the term *Stash hit* when a client requests a data block and finds that the data is already present in the local Stash. Fig. 1 and 2 show the latency and throughput of the read operation of Ring ORAM against the stash hit ratio. Stash hit ratio of 0% indicates that data requests are always sent from the client to the server, whereas a 100% ratio signifies that all necessary blocks are located within the local stash, eliminating the need for server interaction.

The findings indicate that with a stash hit ratio of 60%, there's an approximate 36% reduction in read latency, and a further decrease to around 46% is observed when the hit ratio increases to 80%, compared to situations where the hit ratio is 0%. Furthermore, read throughput saw an improvement of

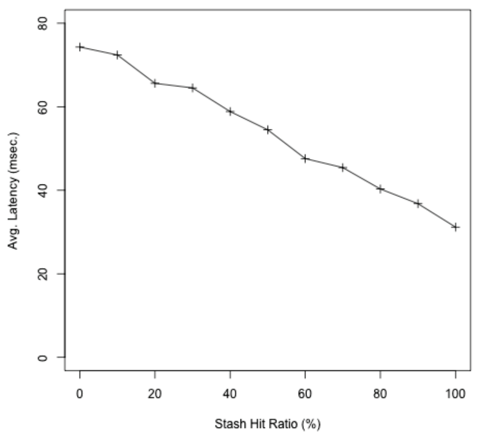

Fig. 1. Average Latency of Read operation.

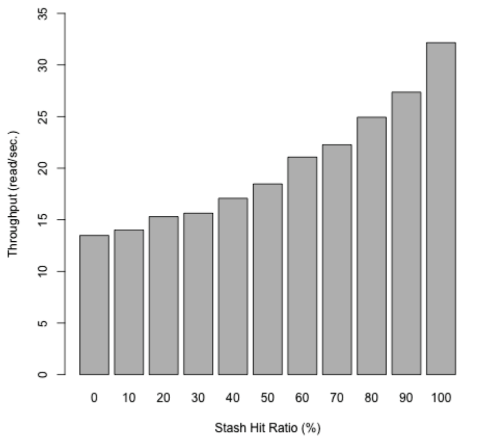

Fig. 2. Throughput of Read operation.

1.56 times at a 60% hit ratio and 1.84 times at an 80% hit ratio, compared to scenarios devoid of caching.

Experimental results from Hitchhiker ORAM [8] revealed that Path ORAM [2] and Fork Path ORAM [12] exhibited negligible rates around 0.15%, whereas Hitchhiker ORAM showed a stash hit ratio of 23.1% on average. Collectively, these findings suggest the potential for enhancing ORAM performance by boosting the stash hit ratio via data prefetching strategies.

IV. Caching and Prefetching in ORAM

We presented the experimental findings in Section III, illustrating that the use of caching in the stash significantly reduces the latency and improves the throughput of read operation in Ring ORAM. Additionally, it showed a direct relationship between an elevated stash hit ratio and a decrease in latency. To improve the stash hit ratio, the prefetching techniques stated in Section II-D are essential. In this section, we discuss the integration of the ORAM controller and the prefetch mechanism.

A. The Design of Prefetcher for ORAM

We show the design of the ORAM controller, including the caching and prefetching mechanisms in Fig. 3.

Upon receiving a data access request from an application, the ORAM client passes it to the ORAM controller for processing. If there is a stash hit, meaning the required data is found in the stash, this is managed through interactions labeled ① and ② in Fig. 3. If the data is not in the stash, the controller forwards the request to the ORAM server, as depicted by ③. The controller monitors and examines the access patterns of each application. Additionally, it instructs the prefetcher to acquire data that applications may need soon, as denoted by ④.

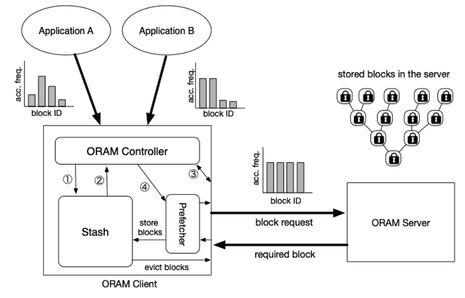

Fig. 3. The architecture of ORAM.

B. Data Prefetching Strategies and Mechanisms

Within the context of the ORAM client shown in Fig. 3, the prefetcher functions as a key element responsible for retrieving data in advance based on specific requests from the ORAM controller. The proactive nature of data prefetching significantly contributes to the optimization of the cache/stash hit ratio, ultimately leading to enhanced system performance. This architecture assumes that the ORAM controller meticulously monitors the data access requests generated by individual applications, conducting thorough analyses to identify suitable prefetching instances.

Prefetching strategies that analyze access patterns typically fall into two categories: those specific to particular domains and those based on machine learning. We discuss prefetching techniques for ORAM for various types of applications.

a) Domain-specific Strategy: Domain-specific prefetching approaches anticipate specific access patterns based on the application type prior to execution [15], [21], [22]. For example, applications like distributed tracing and big-data analytics exhibit unique patterns. Distributed tracing applications typically follow logically related data, such as timestamps and execution histories, whereas big-data analytics applications rely heavily on the spatial locality of data. These types of strategies and mechanisms have been well-developed, with some having the potential to enhance the cache hit ratio.

b) Machine Learning based Strategy: Machine learning (ML) can make a significant contribution to improving the accuracy of data prefetching by predicting irregular access patterns. Although most ML-based prefetchers only predict data based on delta correlations [17]–[19], Voyager provides relatively high accuracy prefetching for applications where data access is not regular. [20]. It introduces a two-tier hierarchical LSTM layer and learns both delta correlations and address correlations. It shows more than 75% accuracy in data prediction for various applications. While different adjustments are required, employing similar prefetching methods across diverse applications could notably enhance ORAM's performance.

C. Access Pattern Obfuscation

The prefetcher is also useful for obfuscating requests to the server. It sends dummy requests alongside the prefetch requests to flatten the access frequency of buckets and blocks in the ORAM storage. This makes it even more difficult for an attacker to identify sensitive data through statistical analysis.

V. Conclusion and Future Work

This paper showed the potential for improving ORAM performance by utilizing stash as a cache through the experimental result. Moreover, we discussed the integration of a prefetching mechanism within ORAM. The proposed design of the prefetching mechanism for ORAM is not only for improving performance through enhancing stash hit ratio but also for obfuscating block requests to the server. We also discussed the prefetching technique for various types of applications to improve the stash hit ratio. Machine learning-based prefetcher is one of the eligible choices for enhancing the hit ratio.

We are planning to develop a prefetching mechanism for the ORAM controller and the prefetcher, aiming to showcase performance enhancements by boosting the stash hit ratio.

Acknowledgement

This work was supported by JSPS Grant-in-Aid for Scientific Research (B) No. 23H03370.

References

[1] Z. Chang, D. Xie, and F. Li, "Oblivious RAM: a dissection and experimental evaluation," *Proceedings VLDB Endowment*, vol. 9, pp. 1113–1124, Aug. 2016.

[2] E. Stefanov, M. Dijk, E. Shi, T. H. H. Chan, and others, "Path ORAM: an extremely simple oblivious RAM protocol," *J. ACM*, 2018.

[3] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "PHANTOM: practical oblivious computation in a secure processor," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, (New York, NY, USA), pp. 311–324, Association for Computing Machinery, Nov. 2013.

[4] C. W. Fletcher, L. Ren, A. Kwon, M. van Dijk, and S. Devadas, "Freecursive ORAM: [nearly] free recursion and integrity verification for position-based oblivious RAM," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (New York, NY, USA), pp. 103–116, Association for Computing Machinery, Mar. 2015.

[5] C. Nagarajan, A. Shafiee, R. Balasubramonian, and M. Tiwari, "ρ: Relaxed hierarchical ORAM," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), pp. 659–671, Association for Computing Machinery, Apr. 2019.

[6] X. Zhang, G. Sun, P. Xie, C. Zhang, Y. Liu, L. Wei, Q. Xu, and C. J. Xue, "Shadow block: Accelerating ORAM accesses with data duplication," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 961–973, IEEE, Oct. 2018.

[7] L. Ren, C. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas, "Constants count: Practical improvements to oblivious {ram}," in *24th USENIX Security Symposium (USENIX Security 15)*, pp. 415–430, 2015.

[8] J. Zhu, M. Li, X. Zhang, K. Bu, M. Zhang, and T. Song, "Hitchhiker: Accelerating ORAM with dynamic scheduling," *IEEE Trans. Comput.*, vol. 72, pp. 2321–2335, Aug. 2023.

[9] L. Ren, C. W. Fletcher, X. Yu, M. van Dijk, and S. Devadas, "Integrity verification for path Oblivious-RAM," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, IEEE, Sept. 2013.

[10] J. Zhu, G. Sun, X. Zhang, C. Zhang, W. Zhang, Y. Liang, T. Wang, Y. Chen, and J. Di, "Fork path: Batching ORAM requests to remove redundant memory accesses," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, pp. 2279–2292, Oct. 2020.

[11] X. Yu, S. K. Haider, L. Ren, C. Fletcher, A. Kwon, M. van Dijk, and S. Devadas, "PrORAM: dynamic prefetcher for oblivious RAM," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 616–628, Association for Computing Machinery, June 2015.

[12] X. Zhang, G. Sun, C. Zhang, W. Zhang, Y. Liang, T. Wang, Y. Chen, and J. Di, "Fork path: improving efficiency of ORAM by removing redundant memory accesses," in *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, (New York, NY, USA), pp. 102–114, Association for Computing Machinery, Dec. 2015.

[13] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM," June 2011.

[14] G. Ayers, H. Litz, C. Kozyrakis, and P. Ranganathan, "Classifying memory access patterns for prefetching," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, (New York, NY, USA), pp. 513–526, Association for Computing Machinery, Mar. 2020.

[15] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 399–411, IEEE, Feb. 2019.

[16] S. Andre, "A case for (partially)-tagged geometric history length predictors," *Journal of InstructionLevel Parallelism*, 2006.

[17] L. Peled, S. Mannor, U. Weiser, and Y. Etsion, "Semantic locality and context-based prefetching using reinforcement learning," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 285–297, Association for Computing Machinery, June 2015.

[18] L. Peled, U. Weiser, and Y. Etsion, "A neural network prefetcher for arbitrary memory access patterns," *ACM Trans. Archit. Code Optim.*, vol. 16, pp. 1–27, Oct. 2019.

[19] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1919–1928, PMLR, 2018.

[20] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, (New York, NY, USA), pp. 861–873, Association for Computing Machinery, Apr. 2021.

[21] B. S. Gill and D. S. Modha, "SARC: Sequential prefetching in adaptive replacement cache," in *USENIX Annual Technical Conference, General Track*, pp. 293–308, 2005.

[22] R. Bera, A. V. Nori, O. Mutlu, and S. Subramoney, "DSPatch: Dual spatial pattern prefetcher," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, (New York, NY, USA), pp. 531–544, Association for Computing Machinery, Oct. 2019.