Poster: Oblivious Data Structures

Xiao Wang*, Kartik Nayak*, Chang Liu*, Elaine Shi*, Emil Stefanov†, Yan Huang*
* {wangxiao, kartik, liuchang, elaine, yhuang}@cs.umd.edu
University of Maryland, College Park
† emil@cs.berkeley.edu
University of California, Berkeley

***Abstract*—We are among the first to systematically investigate (memory-trace) oblivious data structures. We propose a framework for constructing a variety of oblivious data structures, achieving asymptotic performance gains in comparison with generic Oblivious RAM (ORAM). We evaluate the performance of our oblivious data structures in terms of their bandwidth overheads, and also when applied to a secure computation setting. Finally, we leverage our new framework to design an efficient oblivious memory allocator which is particularly useful due to the community's recent efforts in compiling programs targeting ORAM-capable secure processors.**

I. Introduction and Motivation

It is well-known that access patterns, to even encrypted data, can leak sensitive information such as encryption keys [1], [2]. This problem of access pattern leakage is prevalent in numerous application scenarios, including cloud data outsourcing, [3], design of tamper-resistant secure processors, [4], [5], as well as secure multi-party computation [6], [7].

General approaches for hiding access patterns is referred as Oblivious RAM (ORAM). While it is powerful and allows the oblivious simulation of any program, state-of-the-art ORAM constructions incur moderate bandwidth overhead despite the latest progress in this area. Therefore, it will be beneficial to have customized, more efficient constructions for a set of common algorithms that are widely used in practice. This will have numerous applications. Below we present some background as well as motivating application settings.

a) Cloud data outsourcing: Oblivious data structures will be particularly useful for outsourcing a database (e.g., SQL database) to a cloud server, and subsequently making database queries. Since modern database implementations rely heavily on data structures, it will be interesting to consider using our oblivious data structures to build oblivious databases.

b) Secure processor: Tamper-resilient secure processors have been designed to resist physical attacks where an adversary may have physical access to the victim's computing platform (e.g., captured devices, malicious insiders or intruders for cloud data centers). The idea is to reduce the hardware trusted computing base (TCB) to the processor itself. The trust boundary is drawn around the chip, and anything off-chip is considered insecure, such as memory and peripheral devices. In particular, it is well-known that memory probes or cold-boot style attacks can be feasible with relatively low cost.

Existing secure processors designed in academia and industry alike, including AEGIS [8] and Intel SGX encrypt and authenticate memory; however, memory addresses are still transferred in the clear on the memory bus. Unsurprisingly, researchers have shown that such memory buses can leak sensitive information such as secret encryption keys [1].

Recently, with the advances in making ORAM practical, researchers have designed the first secure processor prototypes that obfuscate memory access patterns in a provably secure manner [4], [5]. Since generic ORAM incurs moderate overhead, it would be interesting to consider providing hardware support for our oblivious data structure. Since our framework for constructing oblivious data structures rely on a position-based ORAM such as Path ORAM (without recursion), it is conceptually not hard to modify existing ORAM-capable secure processor prototypes [4], [5] to support ODS as well.

c) Secure computation: Traditionally, secure computation begins by converting a function of interest into a circuit. Recently, researchers have observed the inadequacy of circuit-model secure computation when applied to a scenario with big data [6]. In particular, a compiler that translates programs into circuits would have to copy the entire data for every dynamic memory access whose address depends on sensitive data.

RAM-model secure computation [6] was recently proposed as an attractive alternative to the traditional circuit model. In theory, RAM-model secure computation asymptotically scales better for programs that access a big dataset. In standard RAM-model secure computation, ORAM is used to prevent information leakage through access patterns to data. Our oblivious data structures can be used as a more efficient alternative to generic ORAM.

II. Contribution

We are among the first to systematically formulate and investigate *oblivious data structures* (ODS). Data structures are a particularly important class of algorithms due to its prevalent usage in many practical settings, such as graph algorithms, genomic algorithms, and database applications. Our contributions include the following.

A framework for constructing oblivious data structures. We propose a new framework for constructing oblivious data structures, and under this framework we derive a suite of efficient data structure implementations, including the commonly used `map/set`, `priority_queue`, `stack`, `queue`, `doubly-linked list`, and `deque`.

Asymptotic performance savings. We achieve asymptotic performance gains for our oblivious data structure constructions. Using state-of-the-art generic ORAMs to construct oblivious data structures would result in $O(\log^2 N)$ or

Data structure	Client side storage	Overhead
stack	$O(\log N)\omega(1)$	$O(\log N)$
queue	$O(\log N)\omega(1)$	$O(\log N)$
priority_queue	$O(\log N)\omega(1)$	$O(\log N)$
map/set	$O(\log N)\omega(1)$	$O(\log N)$
(doubly-linked) list	$O(\log N)\omega(1)$	$O(\log N)$
deque	$O(\log^2 N)$	$O(\log N)$
Naive ORAM baseline	$O(\log^2 N)$ or $O(1)$	$O(\log^2 N)$ or $O(\frac{\log^2 N}{\log\log N})$

TABLE I: **Asymptotic performance of our oblivious data structures in comparison with naive ORAM baseline.** A note on the notation $g(N) = O(f(N))\omega(1)$:this notation means that for any $\alpha(N) = \omega(1)$, it holds that $g(N) = O(f(N)\alpha(N))$.

$O(\log^2 N/\log\log N)$ bandwidth overhead [9], [10] (where N is the total data size), while consuming constant to polylogarithmic client-side storage. In comparison, our oblivious data structure schemes achieve $O(\log N)$ overhead while consuming (poly-)logarithmic client-side storage. Table I summarizes the asymptotic performance of our oblivious data structures.

Practical performance savings. We evaluated our oblivious data structures with various application scenarios in mind. For the outsourced cloud storage and secure processor settings, bandwidth overhead is the key metric; whereas for a secure computation setting, we consider the number of AES encryptions necessary to perform each data structure operation. Our simulation shows a $10\times$–$15\times$ speedup under moderate data sizes, in comparison with using generic ORAM, shown in Figure 1. Since the gain is shown to be asymptotic, we expect the speedup to be even greater when the data size is bigger.

Oblivious Memory Allocator We applied our ODS to an important operating system task: memory management. We shall show that naive extensions to existing memory management algorithms using ORAM are not secure, since the total number of memory accesses may still leak information. We also show that padding in this case results in a secure but inefficient solution. Due to these reasons, we develop a new memory management algorithm which stores meta data in a tree-like structure, which can be implemented as an ODS, so that each memory allocation operation can be executed in the same time as one ORAM operation.

III. Technical Highlight.

Our key insight is that common data structures have much more restricted access pattern graphs than generic RAM programs that make arbitrary random accesses to data. For example, for a binary search tree or heap, memory accesses can only go from one tree node to an adjacent one. Therefore, we should be able to gain some efficiency (compared to ORAM) by not hiding some publicly known aspects of the access patterns.

Our techniques are inspired by Gentry *et al.* [11] who describe a technique for performing a binary search in a single ORAM lookup, using any position-based recursive ORAM. We generalize their technique to common data structures. In the most rudimentary form, the key idea is to store children's position tags in the parent node (of the access pattern graph), such that when one fetches the parent node, one immediately obtains the position tags of its children, thereby eliminating the need to perform position map lookups (hence a logarithmic factor improvement). Making this basic idea work for dynamic data structures such as balanced search trees is more intricate, since the access pattern structure may change during the lifetime of the data structure. Our idea (among others) is to use a cache to store nodes fetched during a data structure operation, and guarantee that for any node we need to fetch from the server, its position tag already resides in the client's cache.

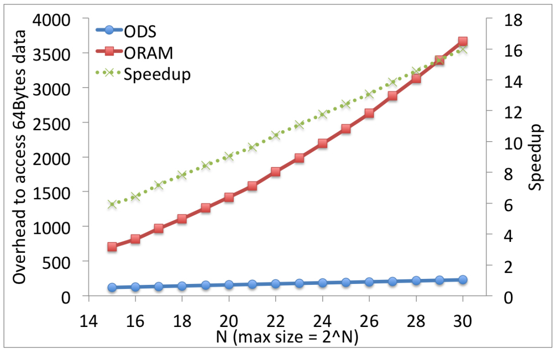

Fig. 1: **Bandwidth overhead of oblivious Map/Set in comparison with naive ORAM.** Payload = 64Bytes. The speedup curve has the y-axis label on the right-hand side.

References

[1] X. Zhuang, T. Zhang, and S. Pande, "Hide: an infrastructure for efficiently protecting information leakage on the address bus," *SIGARCH Comput. Archit. News*, vol. 32, no. 5, pp. 72–84, Oct. 2004.

[2] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Network and Distributed System Security Symposium (NDSS)*, 2012.

[3] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM," in *NDSS*, 2012.

[4] 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 *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[5] C. W. Fletcher, M. v. Dijk, and S. Devadas, "A secure processor architecture for encrypted computation on untrusted programs," in *STC*, 2012.

[6] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis, "Secure two-party computation in sublinear (amortized) time," in *CCS*, 2012.

[7] S. Lu and R. Ostrovsky, "Distributed oblivious RAM for secure two-party computation," in *Theory of Cryptography Conference (TCC)*, 2013.

[8] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the 17th annual international conference on Supercomputing*, ser. ICS '03. New York, NY, USA: ACM, 2003, pp. 160–171.

[9] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM – an extremely simple oblivious ram protocol," in *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[10] E. Kushilevitz, S. Lu, and R. Ostrovsky, "On the (in)security of hash-based oblivious RAM and a new balancing scheme," in *SODA*, 2012.

[11] C. Gentry, K. A. Goldman, S. Halevi, C. S. Jutla, M. Raykova, and D. Wichs, "Optimizing ORAM and using it efficiently for secure computation," in *Privacy Enhancing Technologies Symposium (PETS)*, 2013.