ObliviStore: High Performance Oblivious Distributed Cloud Data Store

Extended Abstract

Emil Stefanov (UC Berkeley) Elaine Shi (UMD)

1 Introduction

It is well established that access patterns to encrypted data can leak a considerable amount of sensitive information [13]. Oblivious RAM (or ORAM for short) [5–11, 14, 18–20, 26, 28], originally proposed by Goldreich and Ostrovsky [8], is a cryptographic construction that allows a client to access encrypted data residing on an untrusted storage server, while completely hiding the access patterns to storage.

Particularly, the sequence of physical addresses accessed is independent of the actual data that the user is accessing. To achieve this, existing ORAM constructions [5–11, 14, 18–20, 26, 28] continuously re-encrypt and and reshuffle data blocks on the storage server, to cryptographically conceal the logical access pattern.

Aside from storage outsourcing applications, ORAM (in combination with trusted hardware in the cloud) has also been proposed to protect user privacy in a broad range of online services such as behavioral advertising, location and map services, web search, and so on [4, 15].

While the idea of relying on trusted hardware and oblivious RAM to enable access privacy in cloud services is promising, for such an approach to become practical, a key challenge is the practical efficiency of ORAM. ORAM was initially proposed and studied mostly as a theoretic concept. However, several recent works demonstrated the potential of making ORAM practical in real-world scenarios [15, 25, 28, 29].

1.1 Our Contributions

We design and build ObliviStore, an efficient ORAM-based cloud data store, securing data and access patterns against adversaries in the malicious model. To the best of our knowledge, ObliviStore is the fastest ORAM implementation that has ever been built.

Our evaluation suggests that in a single client/server setting with 7 rotational hard disk drives (HDDs), ObliviStore is *an order of magnitude faster* than the concurrent and independent work PrivateFS by Williams *et. al.* [29] – with parameters chosen to best replicate their experimental setup.

We evaluate the performance of ObliviStore with both rotational hard drives (HDDs) and solid-state drives (SSDs). With 11 trusted nodes (each with a modern CPU), we achieve a throughput of **31.5MB/s** with a block size of 4KB.

Asynchronizing ORAM operations. We propose novel techniques for making the SSS ORAM [25] asynchronous and parallel. We chose the SSS ORAM since it is one of the most bandwidth efficient ORAM constructions known to date. Due to ORAM's stringent security requirements, asynchronizing ORAM operations poses unique challenges. We must prevent information leakage not only through access patterns as in traditional synchronous ORAM, but also through the timing and out-of-order processing of I/O events. To address this issue, we are the first to formally define the notion of *oblivious scheduling*. We prove that our construction satisfies the oblivious scheduling requirement. Particularly, our ORAM scheduler relies on carefully placed semaphores. To satisfy the oblivious scheduling requirement, operations on semaphores (e.g., incrementing, decrementing) must depend only on information observable by an adversary who is not aware of the data request sequence.

Distributed ORAM. Typical cloud service providers have a distributed storage backend. We show how to adapt our ORAM construction for a distributed setting.

Note that naive methods of partitioning and distributing an ORAM may violate security. For example, as pointed out in [25], even if each block is assigned to a random partition when written, accessing the same block twice in a row (read after write) can leak sensitive information. Our distributed ORAM construction applies the SSS partitioning framework [25] twice to achieve secure partitioning of an ORAM across multiple servers.

We also propose a novel algorithm for securely scaling up a distributed ORAM at run-time, as naive techniques can easily break security. Our techniques allow additions of new processors and storage to an existing distributed ORAM without causing service interruption.

2 Experimental Results

2.1 Results with Rotational Hard Disk Drives

We ran experiments with a single ORAM node with an i7-930 2.8 Ghz CPU and 7 rotational WD1001FALS 1TB 7200 RPM HDDs with 12 ms random I/O latency [1]. To be comparable to PrivateFS, our experiments are performed over a network link simulated to have 50ms latency. We also choose the same block size, i.e., 4KB, as PrivateFS.

Throughput and response time. Figure 2 shows the throughput of our ORAM against the ORAM capacity. *For a 1TB ORAM, our throughput is about* 364*KB/s*. Figure 3 plots the *response time* for data requests with various ORAM capacities. *For a 1TB ORAM, our response time is about* 196*ms*. We stress that throughput and response time are measured under maximum load – therefore, they account for both the online data retrieval and the offline shuffling overhead.

In both Figures 2 and 3, we also marked data points for PrivateFS and PD-ORAM for comparison. For a 1 TB ORAM, ObliviStore has about 18 times higher throughput than PrivateFS. Note that we set up this experiment and pa-

Scheme	Experimental setup			Results		
	block size	ORAM capacity	processors	private RAM consumed	response time	throughput
Secure co-processors (IBM 4768), distributed setting						
Lorch *et. al.* [15]	10 KB	320 TB	10,000*	300 GB	360 ms	28 KB/s
7 HDDs, 50ms network latency to storage, 12ms disk seek latency, single modern processor (client-side)						
PrivateFS‡ [2, 29]	4 KB	100MB	1	$<$ 2 GB †	$>$1s†	110 KB/s† (peak performance [2])
PD-ORAM‡ [29]	10 KB	13 GB	1	$<$ 2 GB†	$>$1s	15 KB/s
ObliviStore	4 KB			**0.46 GB**	**191 ms**	**757 KB/s**
PrivateFS‡ [2, 29]	4 KB	1 TB	1	$<$ 2 GB†	$>$1s	20 KB/s†
ObliviStore	4 KB			**2.3 GB**	**196 ms**	**364 KB/s**
Distributed setting, 20 SSDs, 11 modern processors (1 oblivious load balancer + 10 ORAM nodes, each node with 2SSDs)						
ObliviStore	4 KB	3 TB	11	**36 GB**	**66 ms**	**31.5 MB/s**
ObliviStore	16 KB	3 TB	11	**33 GB**	**276 ms**	**43.4 MB/s**

Figure 1: **Comparison with concurrent work.**

Throughput means average total throughput measured after warming up the ORAM with $O(N)$ accesses, unless otherwise indicated.
†: These numbers obtained through personal communication [2] with the authors of PrivateFS [29]. PrivateFS reports the amount of private memory provisioned (instead of consumed) to be 2GB.
‡: Based on personal communication with the authors, the PrivateFS paper has two sets of experiments: PD-ORAM experiments and PrivateFS experiments. Based on our understanding: *i)* PD-ORAM seems to be an older version of PrivateFS; and *ii)* the experimental methodology for these two sets of experiments are different.
*: Based on a combination of experimentation and theoretic projection. Due to the constrained computational power of IBM 4768 secure co-processors, unlike PrivateFS and ObliviStore, the ORAM implementation by Lorch *et. al.* [15] is mainly constrained by the computational power and memory available on these off-the-shelf secure co-processors.

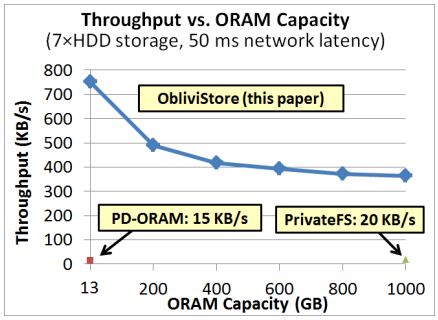

Figure 2: **ObliviStore throughput with 7 HDDs.** Experiment is performed on a single ORAM node with the following parameters: 50ms network latency between the ORAM node and the storage, 12ms average disk seek latency, and 4KB block size.

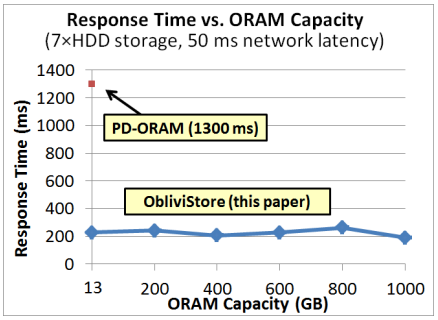

Figure 3: **ObliviStore response time with 7HDDs.** Experiment is performed on a single ORAM node with the following parameters: 50ms network latency between the ORAM node and the storage, 12ms average disk seek latency, and 4KB block size.

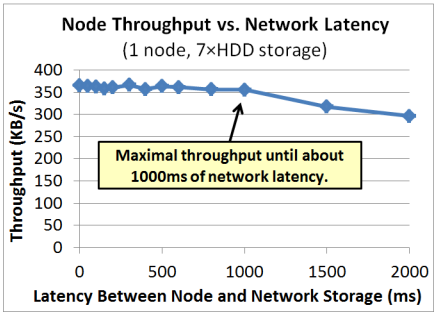

Figure 4: **Effect of network latency on throughput with 7HDDs.** Experiment is performed on a single ORAM node with 7 HDDs, 12ms average disk seek latency, and 4KB block size.

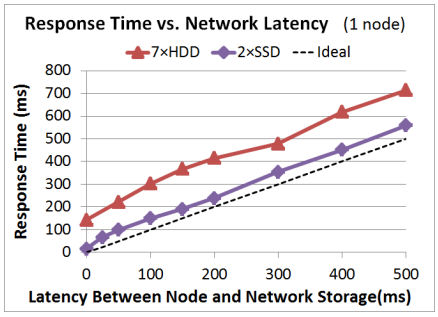

Figure 5: **Effect of network latency on response time.** Experiment is performed on a single ORAM node with 7 HDDs (12ms average seek latency), and again with 2 SSDs. Block size = 4KB. The ideal line represents the roundtrip network latency.

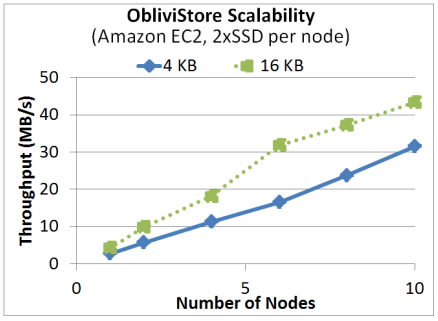

Figure 6: **Scalability of ObliviStore in a distributed setting.** 1 oblivious load balancer, 2 SDDs attached to each ORAM node. Throughput is the aggregate ORAM throughput at the load balancer which distributes the load across all ORAM nodes.

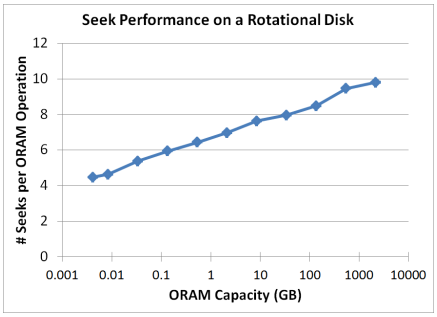

Figure 7: **Average number of seeks of ObliviStore per ORAM operation.** Includes all I/O to storage (reads and writes/shuffles). Experiment is performed on a single ORAM node with 4KB block size.

rameters to best replicate the exact setup used in the PrivateFS and PD-ORAM experiments [29].

Small number of seeks. Our optimizations for reducing disks seeks help greatly in achieving (relatively) high performance. Figure 7 plots the average number of seeks per ORAM operation. For 1TB to 10TB ORAMs, ObliviStore does under 10 seeks per ORAM operation on average.

Effect of network latency. In Figures 4 and 5, we measure the throughput and latency of a 1 TB ObliviStore ORAM under different network latencies. The results suggest that for rotational hard drives, the throughput of ObliviStore is almost unaffected until about 1 second of network latency. To obtain higher throughput beyond 1s network latency, we can increase the level of parallelism in our implementation, i.e., allowing more concurrent I/Os – but this will lead to higher response time due to increased queuing and I/O contention.

The response time of ObliviStore (single node with 7 HDDS) is consistently 140ms to 200ms plus the round-trip network latency. The additional 140ms to 200ms is due to disk seeks, request queuing, and I/O contention.

2.2 Summary of Results with Solid State Drives

The throughput of ObliviStore with 2x1TB SSDs of storage is about 6-8 times faster than with 7 HDD. For a typical 50ms network link, the response time with SSD storage is about half of that with HDDs. Due to space limitations, we could not include detailed SSD experiment results in this abstract except for the distributed setting results in Section 2.3.

2.3 Distributed Setting

We measure the scalability of ObliviStore in a distributed setting. We consider a deployment scenario with a distributed TCB in the cloud. We assume that the TCB is established through techniques such as Trusted Computing, and that the TCB is running on a modern processor. How to implement code attestation to establish such a distributed TCB has been addressed in orthogonal work [16,17,21,22], and is not a focus of this evaluation.

For the distributed SSD experiments, each ORAM node was a hi1.4xlarge Amazon EC instance with 2x1TB SSDs of storage directly attached, and the load balancer ran on a cc1.4xlarge instance. Although our instances have 60GB of *provisioned* RAM, our implementation used far less (under 3 GB per ORAM node, and under 3.5 GB for the load balancer). The load balancer and the ORAM nodes communicate through EC2's internal network (under 5ms RTT).

Figure 6 suggests that the throughput of ObliviStore scales up linearly with the number of ORAM nodes, as long as we do not saturate the network. The total bandwidth overhead between the oblivious load balancer and all ORAM nodes is 2X, and we never saturated the network in all our experiments. For example, with 10 ORAM nodes and 4KB block size, the ORAM throughput is about 31.5 MB/s, and the total bandwidth between the load balancer and all ORAM nodes is about 63 MB/s. We also measured that the response time in the distributed setting is about 60ms for 4KB blocks and is mostly unaffected by the number of nodes.

The throughput of ObliviStore using HDD storage (also tested on Amazon EC2) similarly scales linearly with the number of nodes (please refer to the full paper).

References

[1] `http://www.storagereview.com/php/benchmark/suite_v4.php?typeID=10&testbedID=4&osID=6&raidconfigID=1&numDrives=1&devID_0=368&devCnt=1`.

[2] Personal communication with radu sion and peter williams., Nov. 2012.

[3] D. Asonov and J.-C. Freytag. Almost optimal private information retrieval. In *PET*, 2003.

[4] M. Backes, A. Kate, M. Maffe, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *S & P*, 2012.

[5] D. Boneh, D. Mazieres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical. Manuscript, `http://dspace.mit.edu/bitstream/handle/1721.1/62006/MIT-CSAIL-TR-2011-018.pdf`, 2011.

[6] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly secure oblivious RAM without random oracles. In *TCC*, 2011.

[7] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*, 1987.

[8] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 1996.

[9] M. T. Goodrich and M. Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP*, 2011.

[10] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *ACM Cloud Computing Security Workshop (CCSW)*, 2011.

[11] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*, 2012.

[12] A. Iliev and S. W. Smith. Protecting client privacy with trusted computing at the server. *IEEE Security and Privacy*, 3(2):20–28, Mar. 2005.

[13] 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.

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

[15] J. R. Lorch, J. W. Mickens, B. Parno, M. Raykova, and J. Schiffman. Toward practical private access to data centers via parallel ORAM. *IACR Cryptology ePrint Archive*, 2012:133, 2012.

[16] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig. Trustvisor: Efficient TCB reduction and attestation. In *S & P*, 2010.

[17] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *EuroSys*, 2008.

[18] R. Ostrovsky. Efficient computation on oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC)*, 1990.

[19] R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.

[20] B. Pinkas and T. Reinman. Oblivious RAM revisited. In *CRYPTO*, 2010.

[21] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, 2004.

[22] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: a new abstraction for building trusted cloud services. In *Usenix Security*, 2012.

[23] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *ASIACRYPT*, pages 197–214, 2011.

[24] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Syst. J.*, 40(3):683–695, Mar. 2001.

[25] E. Stefanov, E. Shi, and D. Song. Towards practical oblivious RAM. In *NDSS*, 2012.

[26] P. Williams and R. Sion. Usable PIR. In *NDSS*, 2008.

[27] P. Williams and R. Sion. Round-optimal access privacy on outsourced storage. In *CCS*, 2012.

[28] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *CCS*, 2008.

[29] P. Williams, R. Sion, and A. Tomescu. Privatefs: A parallel oblivious file system. In *CCS*, 2012.