POSTER: ConcurORAM: High-Throughput Parallel Multi-Client ORAM

Anrin Chakraborti
Stony Brook University
anchakrabort@cs.stonybrook.edu

Radu Sion
Stony Brook University
sion@cs.stonybrook.edu

ABSTRACT

Oblivious RAM (ORAM) mechanisms have improved rapidly in recent years as increasing amounts of data are outsourced. Although several tree-based ORAMs such as PathORAM [8] and RingORAM [6] have achieved near-optimal bandwidth for *single client* scenarios, their low overall throughput due to high latency of access – as clients need to wait for or know about and coordinate with each other, lest privacy is lost – reduces their applicability for multi-client scenarios.

In this paper, we propose ConcurORAM , a multi-client concurrent ORAM that eliminates waiting for concurrent clients and significantly increases overall throughput. ConcurORAM works by securely allowing multiple clients to asynchronously access the data set in between eviction rounds by judiciously storing ORAM position map data in a smaller parallel de-amortized pyramid ORAM [10] of higher complexity. In effect ConcurORAM reaps the benefits of parallelism at a lower $O(\log(N))$ overall complexity by identifying and securely accessing the absolute critical data structures that require parallel access with privacy (position map) and designing everything else using append-only data structures that can be then merged securely in a separate eviction step.

1. INTRODUCTION

As increasing amounts of confidential data are outsourced in today's cloud-centric environments, providing confidentiality and privacy becomes critical. To ensure confidentiality, all data and associated meta-data can be encrypted at client-side, before being stored on the server. The data remains encrypted throughout its lifetime on the server and is decrypted by the client upon retrieval.

However, simply encrypting the data is not enough for ensuring privacy [4]. Even on encrypted data, the sequence of locations read and written leaks information regarding the user's *access pattern* and the stored data.

Oblivious RAM (ORAM) allows a client to hide data access patterns from an untrusted server hosting the data. Informally, the ORAM adversarial model ensures indistinguishability between multiple equal-length client query sequences. Since the original ORAM construction by Goldreich and Ostrovsky [3], a large volume of literature [6, 8, 10] has been dedicated to developing more efficient ORAMs.

Of these, for a client with $O(n)$ storage (and small constants), PathORAM [8], based on the original binary tree ORAM construction by Shi et al.[7] is widely accepted as the asymptotically the most *bandwidth efficient* ORAM. RingORAM [6] further optimizes PathORAM [8] for practical deployment by reducing the bandwidth complexity constants.

Problem Definition. Although, recent tree-based ORAM designs have achieved near-optimal *bandwidth* for single-client scenarios, one critical challenge, yet to be fully addressed, is to provide these ORAMs with the ability to accommodate multiple concurrent clients.

Note that it is relatively straight-forward to deploy existing ORAM mechanisms to support multiple clients by sharing ORAM credentials and storing related data structures (e.g., the stash and the position map in the case of a tree-based ORAM) on the storage server to ensure state consistency. In this setup, for privacy, only *one client at a time* can be allowed to access the server-hosted data structures. This reduces the overall throughput and increases the query response time by a factor of the number of concurrent clients. A client (in the worst case) might need to wait for *all* other clients to finish before retrieving the required data item. Since ORAMs often have non-trivial latencies of access (e.g., due to multiple round trips of $O(logN)$ accesses), a client may need to wait a significant amount of time before being able to proceed with the query.

Existing Parallel Constructions. Recent work on oblivious parallel RAM (OPRAM) constructions [1, 9] achieve parallelism at no additional bandwidth cost but under the assumption of constant *inter-client awareness and communication.* Although a step forward, this assumption poses impracticality barriers often times difficult to handle in real scenarios. In contrast, our aim is to achieve parallelism that works without explicit inter-client communication.

Taostore [5] is another interesting parallel ORAM which allows concurrent client queries for an undrlying PathORAM [8] instance, through a trusted proxy client. All client queries are redirected to the trusted proxy which then queries for the corresponding paths from the PathORAM data tree. Further, the proxy runs a secure scheduler to ensure that the multiple paths read do not overlap and leak correlations in the underlying queries. Although, Taostore [5] achieves a significant throughput increase, it can support only a limited number of parallel clients before the throughput reaches

CCS'16 October 24-28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2989062

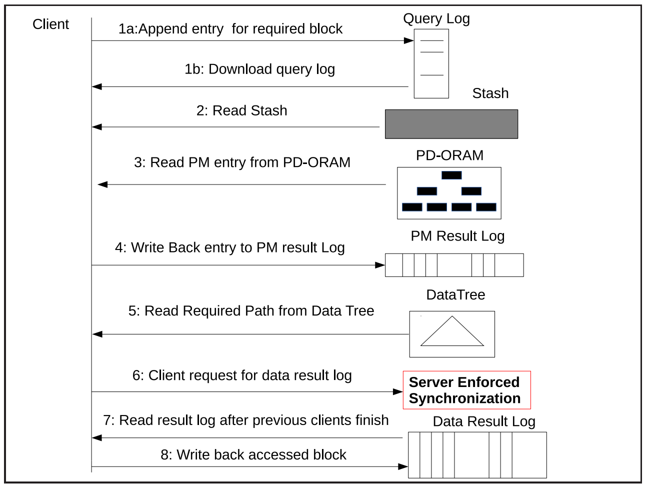
Client
1a:Append entry for required block
Query Log
1b: Download query log
2: Read Stash
Stash
3: Read PM entry from PD-ORAM
PD-ORAM
4: Write Back entry to PM result Log
PM Result Log
5: Read Required Path from Data Tree
DataTree
6: Client request for data result log
Server Enforced Synchronization
7: Read result log after previous clients finish
Data Result Log
8: Write back accessed block

Figure 1: Overview of a query. Steps 1-5 can proceed concurrently. In step 6, clients wait for previous clients to finish before downloading the data result log which is enforced through a server "barrier" for synchronization.

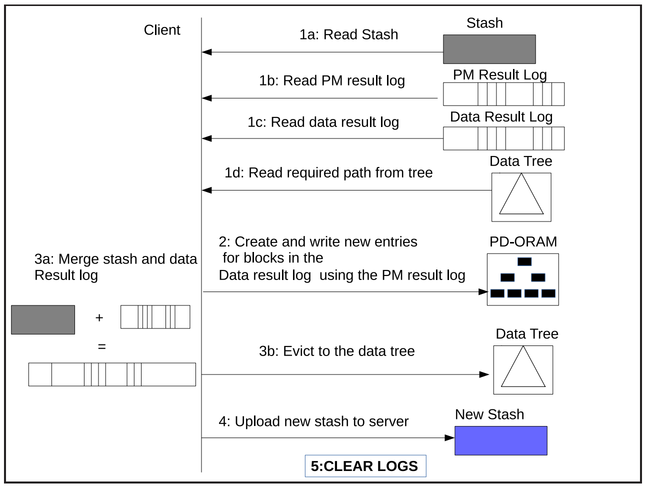
Client
1a: Read Stash
Stash
1b: Read PM result log
PM Result Log
1c: Read data result log
Data Result Log
1d: Read required path from tree
Data Tree
3a: Merge stash and data Result log
2: Create and write new entries for blocks in the Data result log using the PM result log
PD-ORAM
+
=
3b: Evict to the data tree
Data Tree
4: Upload new stash to server
New Stash
5:CLEAR LOGS

Figure 2: High level description of the eviction process. Eviction steps, data items and locations, are randomized and independent of the client queries or data values.

a maximum and the query response time starts increasing. As mentioned in [5], this is because the proxy needs to fetch (and write-back) an equivalent number of paths per access from the data tree as the number of parallel clients in the system. The network traffic is however constrained by the bandwidth, and hence increasing parallel clients increases overall throughput only until the network bandwidth does not limit the benefits of fetching multiple paths of $O(logN)$ data blocks. In contrast, we achieve parallelism while allowing clients to query independently (without the need for a proxy) and thus support a higher number of parallel clients.

2. PROPOSED SOLUTION

ConcurORAM provides a simple construction for achieving parallelism without additional bandwidth requirements and inter-client communication. Below we detail the main insights for ConcurORAM .

Position map access concurrency. Tree-based ORAMs use a special "position map" data structure mapping logical data item IDs to IDs of tree leafs defining a corresponding path from the root "within" which the data items are placed. Specifically, a data item "mapped" to leaf ID l can reside in any of the nodes along the path from the root to leaf l. In a single-client scenario, the position map can be stored client-side. This however is not an option in a multi-client scenario, lest inter-client consistency is lost.

[7] shows how to store the position map server-sides, by dividing it into fixed sizes blocks and storing them recursively in smaller ORAMs. In this case, an access requires reading the position map from the smaller ORAMs (and recursively their position map ORAMs in turn) to obtain the leaf ID for the required data item and then reading the corresponding path to retrieve the data item. Storing the position map recursively however presents a security problem in a multi-client scenario where clients access the position map blocks concurrently. Clients accessing the same position map block concurrently, – especially before random ORAM remapping can de-couple item locations – may leak information and show inter-query correlation.

To mitigate this, ConcurORAM stores the smaller position map in PD-ORAM [10], a concurrent level-based pyramid ORAM ([3]) which is asymptotically more expensive. Interestingly, the small size of the position map ensures that the overall asymptotic complexity of accessing the position map from PD-ORAM [10] is no worse than accessing the recursive position map ORAMs in [8].

Asynchronous operations. Tree-based ORAMs feature two different classes of accesses to the server: (i) fetching data (reading a root-to-leaf path) and eviction (writing back some of the previously read data items to the root-to-leaf path). Some ORAMs couple evictions with queries [2, 8]. Even if a multi-client design can be envisioned in which metadata is stored on the server – for consistency, this coupling forces a synchronous design where only *one* client is in charge at any given time.

This prevents multi-client concurrency. For concurrency, the fetching and eviction steps must be decoupled. Fortunately, RingORAM [6] can be used to securely fetch multiple times before an eviction – a client queries for a fixed number of blocks and writes them to a local stash before evicting the blocks to a deterministically chosen path on the tree. ConcurORAM partially deploys parts of the overall RingORAM design. In ConcurORAM , multiple concurrent client data queries are followed by an eviction step performed by a *single* client.

Server Side Access Logs. A security challenge for all parallel ORAM constructions is to prevent overlapping queries from different clients accessing the same data block. The OPRAM model [1, 9] achieves this through inter-client communication while Taostore [5] uses a trusted proxy to schedule queries securely to the server. Both these mechanisms have practical limitations as described before – inter-client communication maybe prohibitive in many practical scenarios while fetching data for multiple parallel queries at the same time through a *single* proxy is subject to the performance limitation enforced by the network bandwidth.

Instead, ConcurORAM solves this problem through a set of encrypted server side logs – a query log to record all ongoing transactions, a PM result log to record position map items that have already been accessed, and a data result log containing data blocks previously read from the data tree. The logs are maintained by the server as per client requests.

Solution Intuition. Clients use the query log to check for overlapping accesses (accesses to the same data block). Before a client proceeds with a query, it appends an encrypted entry to the query log for the data block it is querying for. Then, the client downloads the query log and checks for overlaps with previous ongoing queries. In the case of an overlap, the client issues a fake query instead.

The PM result log and the data result log effectively perform the function of a server-side cache. Once a client finishes access for an item (data block/ position map item), the most updated value of the item is appended to the corresponding log. By downloading the logs as a whole, clients can thus access items which have been previously accessed by other parallel clients, without leaking privacy. Since the logs are append only and maintained by the server, clients do not need to wait for updating the logs.

Figure 1 provides a high level description of the query protocol in ConcurORAM. In step 6 a client requests for the data result log. The server responds with the data result log only after all *previous* clients have finished accesses. The server enforces this synchronization on the basis of the order of appends to the query log. More specifically, if client i appends an entry to the query log before client j, it receives the data result log before j. Also, client j receives the data result log only after i has finished step 8 in the query protocol. The intuition behind this is as follows: consider that i and j overlap for the same data block. In this case, since i appends an entry to the query log before j, j proceeds with a fake query for step 2-5. After i finishes access, it appends the latest value of the data block to the data result log. Thus, j can now retrieve the data block from the data result log in step 7 without leaking privacy.

Further, the logs are merged and cleared in an eviction step (Figure 2) after a fixed number of accesses which bounds the size of the logs. The eviction tries to greedily place all the blocks from the data result log to the data tree (similar to [6]) while accommodating all blocks that could not be placed in the tree in a fixed sized server-side stash. ConcurORAM locks all server-side data structures during an eviction to ensure consistency.

Clearly, the actual size of the logs and the stash affects performance since clients need to download the logs per access. Note that maximum size of the logs depends on how often an eviction takes place. In fact, an estimation on the basis of empirical numbers reported in [6], the total size of all the logs can be bound to $O(logN)$ while supporting parallelism for upto $O(logN)$ clients without additional bandwidth requirements.

Evaluation. A prototype implementation for ConcurORAM in Java JDK 1.7 has been tested locally for upto 8 parallel clients and shows an almost proportional increase in throughput with increase in the number of clients. The eviction frequency was set equal to the number of parallel clients in the system. We expect the throughput to achieve a maximum with increasing clients due to the proportional increase in the log sizes – this is subject to further experimentation in the future.

3. CONCLUSION

In this paper, we propose the design for ConcurORAM, a multi-client concurrent ORAM based on the idea of identifying and securely accessing the absolute critical data structures that require parallel access with privacy and designing everything else using append-only data structures that can be then merged securely in a separate eviction step.

We plan to further test ConcurORAM and compare it with existing parallel constructions on real cloud settings. Also, a formal bound on log sizes and their impact on performance will be analyzed in future.

4. ACKNOWLEDGEMENT

This work was supported by the National Science Foundation through awards 1161541, 1318572, 1526102, and 152670.

5. REFERENCES

[1] E. Boyle, K.-M. Chung, and R. Pass. Oblivious parallel ram and applications. Cryptology ePrint Archive, Report 2014/594, 2014. http://eprint.iacr.org/.

[2] K. Chung, Z. Liu, and R. Pass. Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead. *CoRR*, abs/1307.3699, 2013.

[3] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43:431–473, 1996.

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

[5] C. S. V. Z. A. E. A. H. R. Lin and S. Tessaro. Taostore: Overcoming asynchronicity in oblivious data storage. IEEE Security and Privacy(Oakland), 2016.

[6] 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)*, pages 415–430, Washington, D.C., Aug. 2015. USENIX Association.

[7] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious ram with o((logn)3) worst-case cost. In *ASIACRYPT*, 2011.

[8] 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 *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

[9] B. C. H. L. S. Tessaro. Oblivious parallel ram: Improved efficiency and generic constructions. Cryptology ePrint Archive, Report 2015/1053, 2015. http://eprint.iacr.org/.

[10] P. Williams, R. Sion, and A. Tomescu. Privatefs: A parallel oblivious file system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 977–988, New York, NY, USA, 2012. ACM.