ConcurORAM: Multi-client concurrency for tree-based oblivious RAM

ABSTRACT

Oblivious RAM (ORAM) technology has

1. INTRODUCTION

With an increasing amount of confidential data being stored on outsourced storage, the privacy of this data is of critical importance. As demonstrated in previous work [4], simply encrypting the data is not enough. Even on encrypted data, the sequence of locations read and written to the storage can leak information regarding the user's *access pattern* and the data stored.

Oblivious RAM (ORAM) is a cryptographic primitive that allows a client to hide its data access patterns from an untrusted storage hosting the data. Informally, the ORAM adversarial model prevents an adversary from distinguishing between multiple equal length sequence of queries made by the client to the server.

Since the original ORAM construction by Goldreich and Ostroevsky [2], a large volume of previous literature [3, 5, 6, 8–10] has been dedicated to developing more efficient ORAM constructions. PathORAM [8], based on the original binary tree ORAM construction by Shi *et al.* [6] is widely accepted to be asymptotically the most *bandwidth efficient* ORAM. RingORAM [5] further improves on the practical overhead of PathORAM [8] by optimizing the constants. Even for ORAMs that divide the data into multiple sub-ORAMs such as ObliviStore [7] and CURIOUS [1], [1] shows that PathORAM [8] is the most suitable ORAM for sub-ORAM design in terms of cost and bandwidth.

Although, recent tree based ORAM designs have achieved near-optimal *bandwidth* for single client scenarios, one critical challenge, yet to be addresses, is to make these ORAMs compatible for concurrent non-overlapping access (access for different data items) for multi-client scenarios, while maintaining security gaurantees. As a motivating example, consider an enterprise that offloads confidential data to a remote storage that deploys an ORAM, and provides access to a group of employees (users). These users should be able to perform non-overlapping queries without a significant performance overhead compared to a scenario where there is only one user accessing the ORAM.

Note that it is trivial to deploy a standard tree based ORAM without concurrency to support multiple clients by sharing the key to the ORAM and storing all related datastructures (the stash and the position map) on the storage server to ensure consistency of state. In this case, only *one* client can access the position map, the stash and the tree at one time while the other concurrent clients must wait for this client to finish. 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 have high latency of access (due to the retrieval of multiple items for one access), this implies that a client would need to wait a significant amount of time before being able to proceed with the query.

In this paper, we propose ConcurORAM , a mechanism to support multi-client concurrency for tree-based ORAMs without sacrificing security. Our work is based on the concurrency scheme proposed by Williams *et al.* [10]. However, there are significant challenges to directly adapting the techniques proposed in [10] for tree based ORAMs. In the following, we discuss these challenges.

Concurrency for position map. First, tree based ORAMs use a position map to store mappings from the logical IDs of data items to the leaf IDs in the tree they are mapped to. Specifically, a data item mapped to leaf ID l can reside in any of the nodes along the path to leaf l from the root. In a single client scenario, the position map can be stored at the client side. For a multi-client scenario the position map must be stored on the server to ensure consistency among the clients. As introduced in [6], the position map can also be stored at the server (to reduce client side storage) in recursively smaller ORAMs. In this case, the position map is divided into fixed size blocks. Thus, an access in this case, requires reading the position map from the smaller ORAMs to obtain the leaf ID for the required data item and then reading the corresponding path to retrieve the data item. Since, the position map is stored recursively, an ORAM storing the position map also has a position map. To ensure that each successive position map is smaller (to ensure

that the recursion terminates with an ORAM that has a a constant size position map), each block in the ORAMs must store multiple position map entries. However, using this in a multi-client scenario allows the server to correlate two client accesses if they access the same position map block. Note that two concurrent clients may access the same position map block even if they are not accessing the same data item.

As one of the main insights, ConcurORAM stores the entries of the position map in a pyramid ORAM ([2, 10]) with multiple levels and uses a hash function to map position map entries to buckets in a level. Note that since the location of an entry is randomized (due to the uniform hash function used), concurrent clients accessing the same bucket, does not leak any correlation between the items queried by the clients. Specifically, ConcurORAM use the concurrent version of PD-ORAM as used in PrivateFS [10] to ensure concurrency for queries.

Decoupling fetching and eviction. Tree based ORAMs divide accesses into two parts – fetching data (reading a root to leaf path) and eviction (writing back the read data to a root to leaf path). To ensure consistency in multi-client scenarios, the fetching and eviction cannot proceed concurrently. Thus, the fetching and eviction must be decoupled. Fortunately, RingORAM [5] provides a mechanism to evict data after a fixed number of fetches. ConcurORAM uses RingORAM and support a fixed number of concurrent queries followed by an eviction by a single client.

ConcurORAM has been implementd and shows an increase in throughput by a factor of **[TODO: Anrin: this number to be filled]** over a standard implementation of RingORAM [5] used non-concurrently for multiple clients.

2. MODEL

Deployment. ConcurORAM considers a deployment model with two parties: the ORAM clients (with limited local storage) and the ORAM server (a remote storage that hosts the clients' data). The server stores data in terms of fixed sized"blocks". ConcurORAM considers N blocks of outsourced data on the server. Clients also access data in blocks addressed by a logical block ID denoted by *id*. The logical address space for all blocks is shared by the clients. The parties engage in an interactive query-response based protocol established by ConcurORAM . The communication channel between the clients and the server is considered secure using SSL.

Clients. Clients are considered honest in the ConcurORAM model and do not interact with each other. Further, the clients share the key to the ORAMs and the secret hash functions used for PD-ORAM, which are stored encrypted on the server. Clients can engage the server without having any knowledge of other client states. Any locking mechanism (as required by the protocol) is imposed by the server. ConcurORAM does not consider the case of malicious clients.

Server. ConcurORAM considers an untrusted server that is honest but curious and does not deviate from the ConcurORAM protocol. The server can observe all requests and try to correlate them by saving and comparing snapshots (state of the server after each query). It stores the ORAM keys, hash functions and other access counters as required by the ConcurORAM protocol. Further, the server maintains and duly increments the counters. ConcurORAM does not consider a malicious server than can mount replay attacks and "fork" client views.

Security challenge. Any system that supports multi-client concurrent access for ORAMs needs to prevent two possible security leaks -

- Correlation between data items concurrently accessed in a single round.
- Correlation between data items accessed in successive rounds of one or more conccurent accesses.

In the above context, we define multi-client concurrent access security for ORAMs as a security game, where the challenger is a fixed set of clients, $\mathcal{C} = \{c_1, c_2, \ldots c_n\}$, the adversary, $\mathcal{A}$ is the remote server that hosts a database uploaded by $\mathcal{C}$. All items in the database are indexed and can be accessed concurrently by the clients.

1. $\mathcal{A}$ and $\mathcal{C}$ engage in polynomial rounds of the following query-response based protocol
   (a) $\mathcal{A}$ selects two sets of item indices $\mathcal{O}_1 = \{x_1, x_2, \ldots, x_n\}$ and $\mathcal{O}_2 = \{y_1, y_2, \ldots, y_n\}$ such that $x_i \neq x_j \forall i, j \in [1, n]$ and $y_i \neq y_j \forall i, j \in [1, n]$. $\mathcal{A}$ sends $\mathcal{O}_1(i)$ and $\mathcal{O}_2(i)$ to c_i where $\mathcal{O}_j(i)$ is the i^{th} item in $\mathcal{O}_j$.
   (b) On the basis of a fairly selected bit b, the clients query for the items in $\mathcal{O}_b$.
   (c) Observing the queries in Step 2, $\mathcal{A}$ outputs bit $b^{'}$
   (d) $\mathcal{A}$ wins the round iff. $b^{'} = b$.
2. $\mathcal{A}$ wins the security game iff. she can win any round with non-negligible advantage over random guessing where non-negligibility is defined over an implementation specific security parameter.

The above security game straightforwardly ensures that $\mathcal{A}$ can win a round with non-negligible advantage over guessing if she can distinguish between two sequence of concurrent accesses. Therefore, a mechanism that satifies the game ensures that an adversary cannot correlate items concurrently accessed in the same round.

Further, consider three randomly chosen sets of item $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}_1$. In three successive rounds, $\mathcal{A}$ provides the following sets of items to $\mathcal{C}$:

- Round1: $\mathcal{O}_1$ and $\mathcal{O}_2$
- Round2: $\mathcal{O}_1$ and $\mathcal{O}_3$

In this case, if $\mathcal{C}$ queries for $\mathcal{O}_1$ both in round 1 and 2, and $\mathcal{A}$ could correlate accesses in the current round with previous rounds, $\mathcal{A}$ can win round 2 with non-negligible advantage. Therefore, a mechanism that satisfies the game also ensures that an adversary cannot correlate accesses in successive rounds.

3. REFERENCES

[1] Bindschaedler, V., Naveed, M., Pan, X., Wang, X., and Huang, Y. Practicing oblivious access on cloud storage: The gap, the fallacy, and the new way forward. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 837–849.

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

[3] Goodrich, M. T., and Mitzenmacher, M. Mapreduce parallel cuckoo hashing and oblivious RAM simulations. *CoRR abs/1007.1259* (2010).

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

[5] Ren, L., Fletcher, C., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., and Devadas, S. Constants count: Practical improvements to oblivious ram. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., Aug. 2015), USENIX Association, pp. 415–430.

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

[7] Stefanov, E., and Shi, E. Oblivistore: High performance oblivious cloud storage. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2013), SP '13, IEEE Computer Society, pp. 253–267.

[8] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., and Devadas, S. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 299–310.

[9] Williams, P., Sion, R., and Carbunar, B. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2008), CCS '08, ACM, pp. 139–148.

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