TaoStore: Overcoming Asynchronicity in Oblivious Data Storage

Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia (Rachel) Lin, Stefano Tessaro
University of California, Santa Barbara
{cetin, victorzakhary, amr, rachel.lin, tessaro}@cs.ucsb.edu

***Abstract*—We consider *oblivious* storage systems hiding both the contents of the data as well as access patterns from an untrusted cloud provider. We target a scenario where multiple users from a trusted group (e.g., corporate employees) asynchronously access and edit potentially overlapping data sets through a trusted proxy mediating client-cloud communication.**

The main contribution of our paper is twofold. Foremost, we initiate the first *formal* study of asynchronicity in oblivious storage systems. We provide corresponding security definitions and show that CURIOUS (Bindschaedler at al., CCS 2015) is *insecure* under asynchronous scheduling of network communication. Second, we develop and evaluate a new oblivious storage system, called Tree-based Asynchronous Oblivious Store, or TaoStore for short, which we prove secure in asynchronous environments. TaoStore is built on top of a new *tree-based* ORAM scheme that processes client requests concurrently and asynchronously in a non-blocking fashion. This results in a substantial gain in throughput, simplicity, and flexibility over previous systems.

I. Introduction

Outsourcing data to cloud storage has become increasingly popular and attractive. Still, confidentiality concerns make potential users skeptical about joining the cloud. In this context, encryption alone is not sufficient to solve all privacy challenges. Indeed, the *access patterns* are *not* hidden from the cloud provider, i.e., the provider can detect for example whether and when the same data item is accessed repeatedly, even though it does not learn what the item actually is. Data access patterns can leak sensitive information using prior knowledge as shown in [2].

This work targets *cloud storage* where multiple users from a trusted group (e.g., employees of the same company) need to access (in a read/write fashion) data sets which may overlap. To achieve this, users' accesses are mediated by a shared (trusted) proxy which coordinates these accesses and, at the same time, reduces the amount of information leaked to the cloud. *Oblivious RAM* (ORAM) is the standard approach to make access pattern *oblivious*, but most ORAM solutions [4], [1] are not practical enough for our multi-user scenario, as they handle operation requests *sequentially*. ObliviStore [3] leverages parallelism to increase throughput and was the first work to consider the proxy model we assume in this work. ObliviStore was recently revisited by Bindschaedler et al. [1], who proposed a new system called CURIOUS fixing a subtle (yet serious) security flaw arising in concurrent environments.

Motivated by Bindschaedler et al. [1], this work initiates a comprehensive and rigorous study of asynchronicity in oblivious storage systems. We make contributions along two axes: 1) We observe that the previous treatment has not captured crucial security issues related to asynchronicity in oblivious storage. We present a comprehensive security framework, and surface an attack showing that *access patterns in CURIOUS are not oblivious in an asynchronous environment.* 2) We design and evaluate a new provably secure system, called TaoStore, that fully resists attacks in asynchronous settings and also fully leverages the benefits of asynchronicity for better performance. Our system follows a completely different paradigm than previous works – in particular it departs from the ObliviStore framework and is completely tree-based – with substantial gains in simplicity, flexibility, and efficiency.

II. Asynchronous ORAM: Definitions and Attacks

Traditional ORAM security definitions consider synchronous and non-concurrent (i.e., sequential) systems. Here, we introduce the new notion of *adaptive asynchronous obliviousness*, or aaob-security, for short. The attacker schedules read/write operation requests (which are possibly concurrent) at any point in time, and also controls the scheduling of messages. Moreover, the attacker learns *when* requests are answered by the ORAM client (i.e., the client returns an output), which as we see below, is very crucial information difficult to hide in practice.

We now proceed with definition of aaob security, which is an indistinguishability-based security notion. Given an attacker $\mathcal{A}$, we consider an experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ where the ORAM client $\mathsf{OClient}$ accesses a storage server SS via an asynchronous link. In this experiment, $\mathcal{A}$ chooses two equally large data sets, D_0, D_1, and samples a random challenge bit $b \xleftarrow{\$} \{0,1\}$. D_b is encoded and stored on the SS and the secret key is given to $\mathsf{OClient}$. The attacker $\mathcal{A}$ can, at any point in time, invoke $\mathsf{OClient}$ with a *pair* of operation requests $(\mathsf{op}_{i,0}, \mathsf{op}_{i,1})$, where both requests can be for arbitrary read/write operations. Then, operation request $\mathsf{op}_{i,b}$ is handed over to $\mathsf{OClient}$. When the operation completes, the adversary $\mathcal{A}$ is notified, yet it is not told the *actual* value returned by this operation. Finally, the adversary $\mathcal{A}$ outputs a guess b' for b, and the experiment terminates. In particular, if $b = b'$, we say that the experiments outputs `true`, and otherwise it outputs `false`.

We define the aaob-*advantage* of the adversary $\mathcal{A}$ against ORAM as

$$\mathsf{Adv}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A}) \Rightarrow \mathtt{true}\right] - \frac{1}{2}\,.$$

We say that ORAM is aaob-*secure* (or simply, secure) if $\mathsf{Adv}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ is negligible for all polynomial-time adversaries $\mathcal{A}$ (in some understood security parameter λ).

Attack against CURIOUS: To overcome the security issue in ObliviStore, CURIOUS [1] suggested an alternative approach based on the idea that a concurrent operation on the same item should trigger a "fake read". When two concurrent requests for the same item are made in CURIOUS, the first request results in the actual "real read" access to the server fetching the item, whereas the second results in a fake access to the storage server SS to hide the repeated access. This gives the attacker a simple mean to break aaob security, and distinguish the $b = 0$ from the $b = 1$ case, by simply scheduling two pairs of operations $(\mathsf{op}_{1,0}, \mathsf{op}_{1,1}), (\mathsf{op}_{2,0}, \mathsf{op}_{2,1})$, where $\mathsf{op}_{1,0}$ and $\mathsf{op}_{2,0}$ are two read requests for the same item, whereas $\mathsf{op}_{1,1}$ and $\mathsf{op}_{2,1}$ are read requests for distinct items. Concretely, the adversary $\mathcal{A}$ first issues the request pair $(\mathsf{op}_{1,0}, \mathsf{op}_{1,1})$, delays the messages sent by $\mathsf{OClient}$ right after the first operation pair is processed, schedules the second request pair $(\mathsf{op}_{2,0}, \mathsf{op}_{2,1})$, and delivers the associated messages to SS, and its replies back to $\mathsf{OClient}$ immediately. If this results in an answer to the second operation being triggered immediately, the attacker guesses $b = 1$, otherwise it guesses $b = 0$.

III. Overview of TaoStore

Motivated by the above concerns, we develop and evaluate TaoStore, a fully-concurrent provably secure multi-user oblivious data store. It relies on a *tree-based* ORAM scheme aimed at fully concurrent data access. Tree-based ORAMs like Path ORAM [4] organize server storage as a tree, and server access is in form of retrieving or overwriting data contained in a path from the root to some leaf. Our new ORAM scheme – TaORAM – resembles Path ORAM, but features a novel approach to manage *multiple* paths fetched concurrently from the server without waiting for on-going flush and write-back operations to complete. All operations are done asynchronously[1]: a) At the arrival of a request for a certain block, the appropriate read-path request is sent immediately to the server. b) Upon the retrieval of a path from the server, the appropriate read/write requests are answered, and the path is flushed and then inserted into a local *subtree* data structure. c) Immediately after flushing a certain number k of paths, their re-encrypted contents are written back to the server (and appropriate nodes deleted from the local subtree).

To prevent the same attack affecting CURIOUS, TaORAM runs an additional auxiliary module *sequencer*, whose sole function is enforcing that logical requests are replied to in the same order as they arrive.

Obliviousness: Path ORAM crucially relies on the fact that a block is assigned to a fresh new random path after each access to hide future accesses to the same block. However, in TaORAM, a request for a block is processed immediately, without waiting for other concurrent accesses to the same block to properly complete and "refresh" the assigned path. If handled naively, this would lead to fetching the same path multiple times, leaking repetition. TaORAM resolves this issue, by keeping track of all requests for the same block so that at each point, only one request triggers reading the actual assigned path, whereas all others trigger fake reads for a random path.

Security and correctness: TaORAM is aaob-*secure*; due to the space constraints, we defer the formal proof to the full version. In particular, a key contribution of our construction is the introduction of a *sequencer* module aimed at preventing our attacks affecting other systems. Correctness is potentially jeopardized when there are multiple on-going read-path and write-back operations to the server. The most prominent issue is that before all write-back operations complete, the contents at the server are potentially out-of-date; hence answering requests using paths read from the server could be incorrect. To overcome this, TaORAM keeps a so-called *fresh-subtree* invariant: The contents on the paths in the local subtree and stash are always up-to-date, while the server contains the most up-to-date content for the remaining blocks. Moreover, every path retrieved from the server is first "*synched up*" with the local subtree, and only then used for finding the requested blocks, which is now guaranteed to be correct by the fresh-subtree invariant. Additionally, correctness (i.e., atomic semantics) remains guaranteed, regardless of the scheduling of messages sent over the network, which is asynchronous and can even be in total adversarial control.

Evaluation: We implemented a prototype of TaoStore and conducted two different evaluations: (1) A local evaluation (with the same experimental set up as in [3]) to compare it with ObliviStore, and (2) A cloud-based evaluation (using Amazon EC2) to test our system in real-world connectivity scenarios. The first evaluation shows for example that TaoStore can deliver up to 57% more throughput with 44% less response time compared to ObliviStore. Our cloud-based evaluations show that while TaoStore's throughput is inherently limited by bandwidth constraints, this remains its main limitation – our non-blocking write-back mechanism indeed allows TaoStore's performance scale very well with increasing concurrency and decreasing memory availability at the proxy.

[1]Here, we highlight the fundamentals of our approach. See the full version of the paper for more details.

References

[1] V. Bindschaedler, M. Naveed, X. Pan, X. Wang, and Y. Huang. 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*, CCS '15, pages 837–849, New York, NY, USA, 2015. ACM.

[2] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[3] E. Stefanov and E. Shi. Oblivistore: High performance oblivious cloud storage. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 253–267, 2013.

[4] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310, 2013.