Pharos: Privacy Hazards of Replicating ORAM Stores

Victor Zakhary, Cetin Sahin, Amr El Abbadi, Huijia (Rachel) Lin, Stefano Tessaro
University of California, Santa Barbara, CA 93106
[victorzakhary,cetin,amr,rachel.lin,tessaro]@cs.ucsb.edu

ABSTRACT

Although outsourcing data to cloud storage has become popular, the increasing concerns about data security and privacy in the cloud blocks broader cloud adoption. Recent efforts have developed oblivious storage systems to hide both the data content and the data access patterns from an untrusted cloud provider. These systems have shown great progress in improving the efficiency of oblivious accesses. However, these systems mainly focus on privacy without considering fault-tolerance of different system components. This makes prior proposals impractical for cloud applications that require 24/7 availability. In this demonstration, we propose **Pharos**, the **P**rivacy **Ha**zards of **R**eplicating **O**RAM **S**tores. We aim to highlight the data access pattern privacy hazards of naively applying common database replication and operation execution techniques such as locking and asymmetric quorums.

1 INTRODUCTION

Outsourcing data to cloud storage has become increasingly popular due to its promise of better scalability and availability. However, existing confidentiality concerns [6] make potential users often skeptical about joining the cloud. While necessary, encryption alone is not sufficient to solve all privacy challenges posed by outsourcing private data. Although encryption hides the data content, the cloud provider can monitor the *access patterns* of each data block. This includes the recency and frequency of access in addition to the type of access such as reads and writes of each data block. Revealing data access patterns presents a real threat to the privacy of the outsourced data. Data access patterns can leak sensitive information using prior knowledge. For example, Islam et al. [11] show a concrete inference attack against an encrypted e-mail repository exploiting only access patterns.

Oblivious RAM (ORAM) – a cryptographic primitive originally proposed by Goldreich and Ostrovsky [9] as a solution for software protection – is the standard approach to hide access patterns. ORAM aims to hide the *target block of each access operation* and the *access operation type*. To achieve these *two* goals, each logical access is translated to a sequence of random looking accesses. In addition, ORAM shuffles and re-encrypts the data content in each data access, making access patterns of any two equally long sequences of read/write operations completely indistinguishable.

Hiding access patterns was initially considered in the context of memory access [9]. While classical ORAM schemes with small client memory apply directly to the memory access setting, in cloud applications a client has more storage space and is capable of storing more data locally and more importantly can outsource the storage of a large dataset to the cloud. The benefits, as well as, the fast adoption of the cloud encouraged the research community in the past several years to develop new secure data services. Many ORAM schemes have been constructed for secure cloud storage systems [3, 4, 12–15, 17]. The initial ORAM proposals focused on developing ORAM constructions for a single client setting. These systems do not fit the requirements of cloud deployments, since accesses to the storage are performed sequentially (e.g. [9, 16]). To overcome the limitations of single client ORAM constructions, recent proposals develop new constructions that adopt real-world storage requirements by enabling multi-client concurrent and asynchronous accesses while preserving access privacy [3, 13, 14]. Figure 1 shows the typical architecture of an ORAM node in a cloud setting. The data is outsourced to the cloud and clients communicates with a trusted proxy to serve their requests. The trust proxy is responsible for data retrieval from the cloud storage in an oblivious manner.

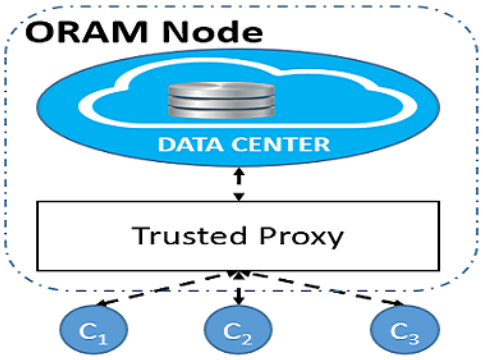

Figure 1: An ORAM node and clients.

Although these systems have shown great improvements in terms of efficiency and throughput, to the best of our knowledge, none of the earlier oblivious cloud storage systems were designed to tolerate the failure of different system components. Designing systems that run on commodity machines should consider failures of different system components as the norm such as machine crashes and network partitioning [7]. By ignoring fault tolerance as an important design principle, existing oblivious cloud storage solutions do not fit the requirements of real-world applications.

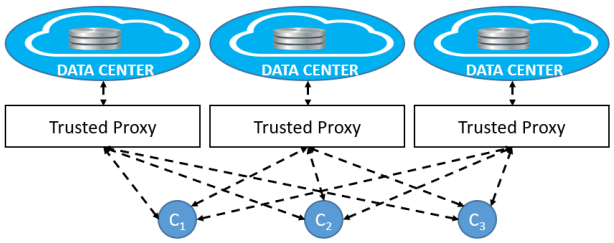

Figure 2: Replicated Cloud ORAM Node.

This demonstration, Pharos, is an initial attempt to illustrate the hazards of ignoring privacy when blindly applying standard database techniques to ensure fault- tolerance. The standard database approach for handling concurrency is locking. The standard

approach to ensure fault-tolerance is replication, and typically quorums are used, where quorums of sets of replicas are chosen so that any two quorums have a none empty intersection. In a database system, read operations are expected to be much more frequent than write operations, hence in a typical database system read operations acquire shared read locks while write operations acquire exclusive write locks. Furthermore, in a replicated fault-tolerant setting, asymetric quorums are typically used, i.e., read operations use read quorums and write operations use write quorums, where read quorums do not need to intersect, but should intersect with write quorums. Since read operations dominate database workloads, read quorums are typically much smalled than the write ones. In fact, if write availability is not critical, a read quorum of one replica and a write quorum of all replicas is often recommended (or to increase write fault-tolerance, a read quorum of two replicas and a write of all replicas minus one).

Pharos supports oblivious access to replicated cloud ORAM nodes. Figure 2 shows an example of a replicated cloud ORAM node. The trusted proxy and the cloud storage are replicated to tolerate the failure of different system components (the cloud or the trusted proxy). Clients independently communicate with different trusted proxies to serve their read and write operations. During the demo, the attendees will be challenged to support a concurrency control and replication scheme. The traditional approaches of locking and asymmetric quorums will be proposed and contrasted with lock-free and symmetric quorums. Either the attendee or the system will propose an attack to illustrate the possible data access privacy violations in the chosen solutions. Pharos is a demonstration that will help bridge the gap between the database community and the security and privacy community by highlighting the privacy hazards in standard database solutions. Pharos presents attendees different attacks, thus giving them a real sense of the potential hazards of using a replication technique or an operation execution technique on the privacy of data access patterns.

2 DATA MODEL

We give a brief overview of oblivious storage systems that depend on a trusted proxy [3, 13, 14]. As shown in Figure 1, an **ORAM node** consists of a *storage* service that is outsourced to the cloud and a *trusted proxy* that is deployed in between to mediate client server communication as well as to execute the oblivious algorithm. Data is stored in a key-value store, which is encrypted and outsourced to the storage and the meta-data used to locate objects in the storage is maintained in the trusted proxy. Clients submit single operations: object lookups, *get(k)*, and object updates, *put(k,v)*, to the trusted proxy, where k is a key, and v is a value. The trusted proxy translates these requests into oblivious retrievals (OR) and oblivious evictions (OE). An **oblivious retrieval** translates a client get or put of an object O into fetching multiple objects where accessing O is obfuscated among the fetched objects. The trusted proxy has to write-back the retrieved objects by performing an oblivious eviction. An **oblivious eviction** hides the type of client access, (get or put), and the access frequency of different objects by shuffling and re-encrypting all the fetched objects at the trusted proxy before writing them back to the storage. Also, the trusted proxy has to update its meta-data to be able to locate these objects in the storage for later accesses.

3 THREAT MODEL

We consider the threat model for asynchronous ORAM that is introduced in [13]. The threat model assumes an honest-but-curious adversary which can see the raw storage and network communication of the server. It controls the asynchronous links where arbitrary delays can be added to different communication messages. Additionally, an adversary can adaptively schedule access, read and write operations and learn the timing of the responses. This security definition is called *aaob-security*. It formalizes the obliviousness in asynchronous and concurrent multiple access deployment scenarios and ensures that two timing consistent executions should be indistinguishable in the described threat model.

4 PRIVACY HAZARDS OF TYPICAL DATABASE TECHNIQUES

Replication has always been used to provide fault tolerance to database systems. Many of the database replication techniques were developed to focus on performance and data consistency assuming that all the data replicas are trusted. Designing systems where privacy is considered a first class requirement narrows the design choices. Many common design alternatives that enhance the performance might lead to a data access pattern privacy violation. We first illustrate the privacy hazards of using *asymmetric read and write quorums*. Then we demonstrate the problems of using *lock-based concurrency control* solutions.

4.1 Asymmetric Quorums Hazard

Database replication is commonly used to tolerate server failures. Once data is replicated, consistency becomes an important challenge. We assume linearizability [10] as a correctness condition for object accesses. Reading an object should always return the most recent committed update to this object. Although replicas can have different versions of the same object, client reads should always return the latest value of an object. This behavior is defined as operation consistency [2]. Operation consistency requires that clients receive the correct expected results regardless of the state consistency of replicas.

Different replication strategies introduce a trade off between fault tolerance, *the number of replica failures f that the system can tolerate before it stops completely*, and performance, *the number of replicas that should be accessed per read and update operations* for a given consistency requirement. We present the trade offs of different replication strategies.

Linearizability and operation consistency on the object level are achieved using quorums and version numbers [8]. A read quorum (q_r) is the minimum number of replicas that need to be accessed to retrieve the latest value of an object. A write quorum (q_w) is the minimum number of replicas that need to be updated to guarantee consistent reads. To achieve operation consistency, any read quorum should intersect with all write quorums $q_r \cap q_w \neq \phi$. This intersection guarantees that a read will always access the latest value of an object. To achieve total order on updates, a centralized sequencer can be used to assign total order version numbers for updates. In this case, write quorums do not have to intersect (e.g. write one read all). However, a total order can be achieved in a distributed way using quorums. In this case, any two write quorum q_{w_1} and q_{w_2} should intersect in at least one replica $q_{w_1} \cap q_{w_2} \neq \phi$. This intersection guarantees that objects will be updated in the same order in all the quorum replicas.

Table 1: Summary of different replication strategies, their requirements, and their guarantees.

Replication Strategy	$\|q_r\|$	$\|q_w\|$	f	Hazardous?
Master/Slave (read optimized)	1	N	0	Yes
Master/Slave (write optimized)	N	1	0	Yes
Majority quorums	N/2	N/2	N/2 - 1	No
Grid quorums [5]	$\sqrt{N}$	$2.\sqrt{N}$	$\sqrt{N}-1$	Yes
Tree quorums [1]	$log(N)$	$log(N)$	$log(N)-1$	No

Table 1 summarizes the commonly used quorum sizes and their degree of fault tolerance in the worst case. Majority quorums tolerate the failure of any number of replicas less than a majority. It executes read and write operations from majority quorums. Master/slave (read optimized) requires write quorums of size N. If updates are sent to all the replicas, reading from any single replica returns the most up to date version of an object. Similarly, master/slave(write optimized) requires read quorums of size N and write quorums of size 1. Reading all the replicas of an object guarantees freshness given that updates are applied to at least one replica. In a master/slave model, the failure of one replica halts any update(read optimized) or read(write optimized) and hence stops the whole system. Many proposals are based on the notion of logical structures that are imposed on the replicas and used to reduce the cost of quorums, while maintaining the intersection property. As a result, usually, one operation, eg, read operations, requires smaller quorums and hence is cheaper, but then write operations are more expensive. Cheung et al. [5] proposed the grid protocol to maintain replicated data. Replicas are ordered in a grid and a read quorum is any row **or** any column in the grid and a write quorum is any row **together with** any column in the grid. The failure of a full row or a full column halts the system and prevents any updates. Agrawal and El Abbadi [1] present tree quorums where read quorums and write quorums are paths in the tree from the root to a leaf. The failure of a full path stops the system.

The last column of Table 1 indicates if a replication strategy could introduce any privacy hazards on the data access patterns. As shown, all replication strategies that use *symmetric* read and write quorums are safe while others that use *asymmetric* read and write quorums are hazardous.

Recall that in the threat model of Section 3, an adversary monitors all the communication links between the clients and the trusted proxies. When asymmetric read and write quorums are used, the adversary can easily distinguish between read and write accesses by observing the number of messages sent per access. This allows the adversary to identify the type of access for every legitimate client request. Although asymmetric quorums can benefit the system performance, they can easily be used to leak the access type which represents a privacy hazard on the data access patterns.

4.2 Locking Hazards

All trusted-proxy-based ORAM systems assume a single trusted proxy that serves all the client requests. This single trusted proxy is responsible for hiding the access patterns of all client requests and achieving linearizability of access for every data object. Upon replicating the ORAM nodes and their trusted proxies, the problem of consistently updating the data replicas occurs. In particular, concurrent updates on the same object need to be serialized across the different data replicas. A widely used approach to tackle this problem is *locking*. When a client updates a data object, she first acquires an exclusive lock on this object from a write quorum. Upon receiving the lock response from a write quorum, the client updates the data object and releases the locks. This prevents any inconsistencies that may occur from concurrent updates on the same object. Any concurrent access on the same object that is issued by another client has to wait until the locks are released. However, this concurrent access would not block if it updates a different object.

Recall that in the threat model of Section 3, an adversary can adaptively schedule read and write accesses while trying to understand the objects that are being accessed by other clients. When locks are used, the adversary can concurrently schedule write accesses on all the objects immediately after a legitimate client's access. The adversary should receive the requested locks on all the objects except the one that is being access by the legitimate client. By observing the locking responses, the adversary can identify which object is being accessed by the legitimate client request. This attack illustrates the hazard of using locks on the privacy of data access patterns.

5 PHAROS

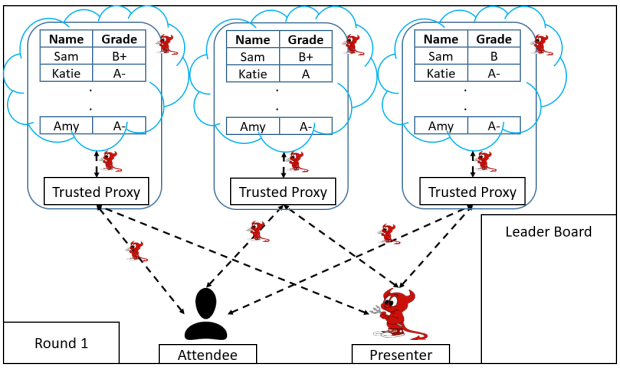

Figure 3: An adversary tries to understand the request types and the data records accessed by a legitimate client.

We propose Pharos to motivate a deeper understanding and appreciation of the privacy hazards that result from the naive application of locks and asymmetric quorums to replicate ORAM stores. Both the attacks on locks and asymmetric quorums result from observing only the communication patterns between the client and the trusted proxy. These attacks are independent of the ORAM implementation. Therefore, we assume a simplified system implementation to illustrate the attacks without distracting the attendee with the ORAM implementation details.

The setup: we assume a simple dataset of 10 student records stored in the ORAM node. Each record has a student name and a grade. The dataset is replicated into three trusted-proxy-based ORAM nodes for fault tolerance. These ORAM nodes are hosted on three cloud machines. The attendee gets to know that locks are used to ensure the consistency of the updates. Also, asymmetric read one write all quorums are used to optimize the latency of read operations. Recall that in the data model of Section 2,

a client can submit *get(student_name)* to read the record of a student and *put(student _name, grade)* to update the grade of a student. There are two types of players in the system: legitimate clients and adversaries. A legitimate client submits requests to read or update a data record. An adversary monitors the network, adds delays to messages, and schedules read and write requests in an attempt to reveal the data record or the type of request submitted by a legitimate client. To simulate adversarial network monitoring, each ORAM node sends the adversary the number of messages it receives and an identification of the source of these messages. The setup and the adversary capabilities are shown in Figure 3.

This demonstration proceeds in multiple rounds where the attendee gets to play one of the two roles and the presenter plays the other role. *In the first round*, the attendee is asked to update some student record and the presenter, on another machine, tries to reveal the data record. *In the second round*, the attendee and the presenter flip roles. The presenter submits a sequence of read and update requests and the attendee tries to reveal the type of request submitted by the presenter. The attendees who are able to explain how the two attacks work will have their names written on the *leader board*.

6 CONCLUSION

Private access of data stores is becoming very relevant in the context of cloud computing. Prior work in the security community focused on the privacy of access. We strongly believe that for the success of the cloud, **both** privacy and fault-tolerance dimensions need to be addressed. The goal of Pharos is twofold. First we propose some simple but effective attacks that reveal significant security leaks when the standard approaches for replication are incorporated in a straightforward manner with the privacy preserving solutions. Second, Pharos is a pedagogical game that educates the attendee on the privacy hazards of replication. Our aim is to make the database practitioner attendee aware of these challenges and hence be motivated to better understand the privacy concerns when designing fault-tolerant database systems.

7 ACKNOWLEDGEMENT

This work is partially supported by NSF grants CNS-1528178 and CNS-1649469.

REFERENCES

[1] Divyakant Agrawal and Amr El Abbadi. 1991. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems (TOCS)* 9, 1 (1991), 1–20.

[2] Marcos K Aguilera and Douglas B Terry. 2016. The Many Faces of Consistency. *IEEE Data Eng. Bull.* 39, 1 (2016), 3–13.

[3] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. 2015. Practicing Oblivious Access on Cloud Storage: The Gap, the Fallacy, and the New Way Forward *(CCS '15)*. ACM, 837–849. DOI: http://dx.doi.org/10.1145/2810103.2813649

[4] Dan Boneh, David Mazieres, and Raluca Ada Popa. 2011. *Remote Oblivious Storage: Making Oblivious RAM Practical.* Technical Report. MIT. MIT Tech-report: MIT-CSAIL-TR-2011-018.

[5] Shun Yan Cheung, Mostafa H Ammar, and Mustaque Ahamad. 1992. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering* 4, 6 (1992), 582–592.

[6] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. 2009. Controlling Data in the Cloud: Outsourcing Computation Without Outsourcing Control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW '09)*. ACM, New York, NY, USA, 85–90. DOI: http://dx.doi.org/10.1145/1655008.1655020

[7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.

[8] David K Gifford. 1979. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*. ACM, 150–162.

[9] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (May 1996), 431–473. DOI: http://dx.doi.org/10.1145/233551.233553

[10] Maurice P Herlihy and Jeannette M Wing. 1990. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12, 3 (1990), 463–492.

[11] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. 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*.

[12] Jacob R. Lorch, Bryan Parno, James Mickens, Mariana Raykova, and Joshua Schiffman. 2013. Shroud: Ensuring Private Access to Large-Scale Data in the Data Center. In *USENIX FAST '13*. USENIX, 199–213. https://www.usenix.org/conference/fast13/technical-sessions/presentation/lorch

[13] C. Sahin, V. Zakhary, A. El Abbadi, H. Lin, and S. Tessaro. 2016. TaoStore: Overcoming Asynchronicity in Oblivious Data Storage. In *2016 IEEE Symposium on Security and Privacy (SP)*. 198–217. DOI: http://dx.doi.org/10.1109/SP.2016.20

[14] Emil Stefanov and Elaine Shi. 2013. ObliviStore: High Performance Oblivious Cloud Storage. In *IEEE SP '13*. 253–267. DOI: http://dx.doi.org/10.1109/SP.2013.25

[15] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. 2012. Towards Practical Oblivious RAM. In *NDSS '12*. http://www.internetsociety.org/towards-practical-oblivious-ram

[16] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol *(CCS '13)*. ACM, 299–310. DOI: http://dx.doi.org/10.1145/2508859.2516660

[17] Peter Williams, Radu Sion, and Alin Tomescu. 2012. PrivateFS: A Parallel Oblivious File System *(CCS '12)*. ACM, 977–988. DOI: http://dx.doi.org/10.1145/2382196.2382299