University of California
Santa Barbara

Enabling Data Security and Privacy for Database Services in the Cloud

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Cetin Sahin

Committee in charge:

Professor Amr El Abbadi, Chair
Professor Divyakant Agrawal
Professor Omer Egecioglu
Professor Huijia (Rachel) Lin
Professor Stefano Tessaro

September 2017

The Dissertation of Cetin Sahin is approved.

Professor Divyakant Agrawal

Professor Omer Egecioglu

Professor Huijia (Rachel) Lin

Professor Stefano Tessaro

Professor Amr El Abbadi, Committee Chair

July 2017

Enabling Data Security and Privacy for Database Services in the Cloud

To my family

Acknowledgements

This work would not be possible without the continuous support of many people around me. My words cannot fully express my gratitude to them.

I first would like to acknowledge my advisor, Prof. Amr El Abbadi, for providing continuous support, guidance, mentoring throughout my Ph.D. He taught me how to do a research and he was so patient with me even if I was progressing slowly. He was not only a great researcher but also a great mentor that knows how to behave and communicate with a graduate student. He kept me motivated with his enthusiasm and excitement for all the progress that I made. I am just so lucky to have a chance to have my advisor. Thank you, Amr. You made this Ph.D. possible for me.

I would thank Prof. Divy Agrawal for his collaboration during the early years of my Ph.D. and his insightful comments and questions in the lab. My additional thanks for Divy is for serving on my dissertation committee. I would also thank Prof. Ömer Eğecioğlu, Huijia (Rachel) Lin and Stefano Tessaro for their collaboration and serving on my committee. Ömer helped me analyze privacy problems mathematically, Rachel and Stefano introduced me Oblivious RAM and helped me think from a privacy perspective. I am happy to have a chance to collaborate all of my committee members. Therefore, I would also like to thank Computer Science department of UCSB for providing such a great working environment. I really appreciate the National Science Foundation for funding my research through grants CNS-1528178 and CCF-1442966.

I had the privilege to work with wonderful people at Distributed Systems Lab (DSL). I would like to thank my lab mates, Faisal, Vaibhav, Victor, Theodore, Xiaofei for creating such a lovely social and work environment.

Although I was away from home, I need to thank my Turkish friends Oğuz, Buğra, Onur Sinan, Arda, Faruk, Semih for making Santa Barbara home for me. It was a lot easier with you.

I would like to thank my amazing family, my mother, my father, and my brother, for always

being there for me. My parents had always believed in me and supported me endlessly even if I fail. My brother taught me how to stay strong and work hard towards my goals. I cannot stress enough how lucky I am to have you. Thank you! You are the main reason I am where I am today.

Finally, I would like to thank my beautiful and lovely wife, Menşure Çağla, for her great support and understanding. Her smile and energy always keep me motivated for my study.

Curriculum Vitæ
Cetin Sahin

Education

2017	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2016	M.S. in Computer Science, University of California, Santa Barbara.
2011	B.S. in Computer Engineering, Bilkent University.

Publications

Cetin Sahin, Aaron Magat, Victor Zakhary, Amr El Abbadi, Rachel Lin, Stefano Tessaro. Understanding the Security Challenges of Oblivious Cloud Storage with Asynchronous Accesses [demo], 2017 IEEE International Conference on Data Engineering, ICDE 2017, San Diego, California, USA, April 19-22, 2017.

Cetin Sahin, Amr El Abbadi. Data Security and Privacy for Outsourced Data In the Cloud [tutorial]. Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.

Cetin Sahin, Brandon Kuczenski, Omer Egecioglu, Amr El Abbadi. Towards Practical Privacy-Preserving Life Cycle Assessment Computations [poster paper]. CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017.

Brandon Kuczenski, Cetin Sahin, Amr El Abbadi. Privacy-preserving aggregation in life cycle assessment. Environment Systems and Decisions (2017) 37: 13, 2017.

Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. Taostore: Overcoming asynchronicity in oblivious data storage. In 2016 IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 23-25, 2016, 2016

Divyakant Agrawal, Amr El Abbadi, Vaibhav Arora, Ceren Budak, Theodore Georgiou, Hatem A. Mahmoud, Faisal Nawab, Cetin Sahin, Shiyuan Wang: Mind your Ps and Vs: A perspective on the challenges of big data management and privacy concerns. BigComp 2015: 1-6

Erdinc Korpeoglu, Cetin Sahin, Divyakant Agrawal, Amr El Abbadi, Takeo Hosomi, Yoshiki Seo. Dragonfly: Cloud Assisted Peer-to-Peer Architecture for Multipoint Media Streaming Applications. IEEE CLOUD 2013: 269-276

Papers In preperation

Cetin Sahin, Brandon Kuczenski, Omer Egecioglu, Amr El Abbadi. Privacy-Preserving Certification of Sustainability Metrics

Cetin Sahin, Tristan Allard, Reza Akbarinia, Amr El Abbadi. PINED-RQ: A Differentially Private Index on Encrypted Databases for Supporting Range Queries

Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. Fault-tolerant Oblivious Cloud Storage

Professional Experience

NEC Laboratories America, Cupertino, California USA
Summer Research Intern in Data Management Team **June 2014 - September 2014**
Mentor: Dr. Jagan Sankaranarayanan
Project: Mobility's Dilemma: To Push or Not Push

Summer Research Intern in Data Management Team **June 2013 - September 2013**
Mentor: Dr. Jagan Sankaranarayanan
Project: Context Transfer in the Cloud for Seamless Mobility

Ipek Computer Industry and Office Automation, Kayseri, Turkey
Part-Time Researcher and Developer **December 2010 - February 2011**
Developed different types of mobile applications for museum systems. These applications are not publicly available.

Summer Internship **Summer 2010**
Developed "Ulusoy Bilet": An iPhone Ticket Automation for Ulusoy.

Omega Advanced Technology, Ankara, Turkey
Part-Time Researcher and Developer **October 2010 - January 2011**
Developed "Securitas TR": An iPhone Application for Professional Security Company.

Summer Internship **Summer 2009**
Project: Advertisement Management Module for Shopping Malls

Honors and Awards

2017 IEEE ICDE Student Travel Grant, 2017

Doctoral Student Travel Grant, 2017

Graduated from Nuh Mehmet Kucukcalik Anatolian High School as a top student, 2006

Attended Bilkent University with full scholarship, 2006

Received 8 High Honor certificates in 8 semesters at Bilkent University, 2007-2011

Our senior project called “Software Product Line for Online Voting Systems” was selected the second best project in IBM Software Academy, 2011

Abstract

Enabling Data Security and Privacy for Database Services in the Cloud

by

Cetin Sahin

Substantial advances in cloud technologies have made outsourcing data to the cloud highly beneficial today (*e.g.,* costs savings, scalability, provisioning time). However, strong concerns from private companies and public institutions about the security of the outsourced data still hamper the adoption of cloud solutions. This reluctance is fed by frequent massive data breaches either caused by external attacks against cloud service providers or by negligent or opaque practices from the service provider itself. For broader adoption of cloud services, this dissertation addresses the data security and privacy concerns in the cloud setting. The goal is to ensure security and privacy of outsourced data while maintaining the ability to execute queries efficiently. Security/privacy comes at a cost of functionality/performance. Therefore, we seek for a proper balance in the space of security, privacy, functionality, and performance. This dissertation works the problems of range query execution over encrypted data, privacy preserving data mining in the context of environmental sustainability studies, and access privacy in the cloud. To enable efficient and secure range query processing over traditional databases, we introduce PINED-RQ, a highly efficient and differentially private range query execution framework that constructs a novel differentially private index over an outsourced database. Second, this dissertation presents a comprehensive study of the environmental sustainability metrics. Our contributions in this context are twofold: 1) to better evaluate the environmental impacts of the industrial processes privately, we formally define *privacy preserving certification* paradigm and develop a framework that enables untrusted third party to certify parties based on a well agreed upon set of criteria. 2) to explore the privacy concerns over publicizing the industrial

activities in the form of life cycle assessment (LCA) computations, which is a standard way of evaluating an impact of a product and service. This dissertation initiates a study to explore privacy and security challenges that prevent organizations from making public disclosures about their activities. Finally, this dissertation explores access privacy in the cloud setting. We design and develop TaoStore, a highly efficient and practical cloud data store, which secures data confidentiality and hides access patterns from adversaries. Additionally, we propose a new ORAM security model, called $aaob$-security, which considers completely asynchronous network communication and concurrent processing of requests. This dissertation shows that it is possible to deliver practical and high-performance data services in the cloud without sacrificing security and privacy if the requirements of each application are analyzed correctly and a correct balance is found in the space of security, privacy, functionality, and performance.

Contents

Chapter 1

Introduction

1.1 Data Security and Privacy in the Cloud

Recent advances in cloud technologies have made outsourcing personal and corporate data to cloud storage servers increasingly popular and attractive, due to its promise of high scalability and availability. However, this increase in utility comes with a risk of exposing data to a number of security threats since the cloud is a popular and tempting attack target. It hosts many businesses at different scales using a shared infrastructure. When an attacker attacks the cloud, it has access to consolidated data, which can have great financial value. For example, a curious administrator might snoop on private data or an adversary might gain unauthorized access to sensitive information. Therefore, potential customers remain skeptical about joining the cloud due to existing confidentiality and privacy concerns [1]. For broader adoption of cloud services, concerns about data security and privacy must be addressed. The question here is how to ensure security and privacy of outsourced data while maintaining the ability to execute queries efficiently.

Providing secure and privacy-preserving data services over outsourced data is challenging. Both the database and the cryptography communities have shown great interest in providing

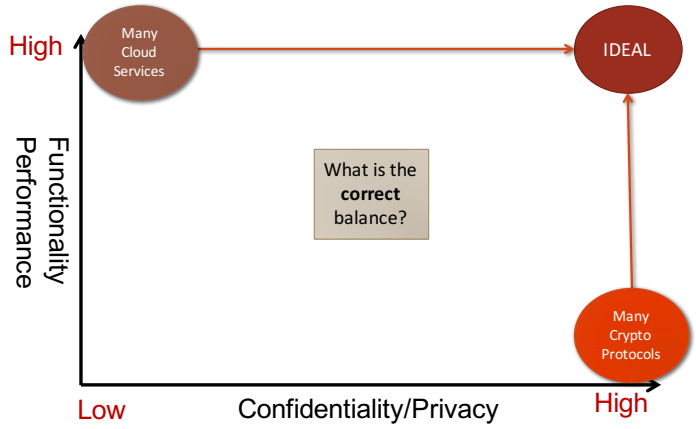


Figure 1.1: Challenge: Conflicting Goals

privacy-preserving and secure data services, but there is no one scheme that solves all the security and privacy problems. Different schemes and models have different security and privacy guarantees, and these protection guarantees come at a cost: decrease in performance and functionality. The ideal scenario of a cloud service would be to ensure very high security/privacy and very high functionality/performance together. However, with the existing cryptographic tools and performance requirements, achieving such a target is not possible. Figure 1.1 presents this impossibility case. There is an obvious trade-off between security/privacy and functionality/performance. Sacrificing functionality and performance completely for the sake of security and privacy makes outsourcing services impractical. Similarly, sacrificing security and privacy for the sake of functionality and performance might cause serious data breaches which is not tolerable. Therefore, any data related service needs to seek a proper balance in the space of security, privacy, functionality and performance.

1.1.1 Confidentiality of Data

The first form of problem for the outsourced databases is to ensure the confidentiality of clear data and storing encrypted data in a hostile environment provides strong data confidentiality. However, the ability to perform practical query processing on encrypted data remains

a major challenge. Various primitive encryption schemes are introduced since they form the building blocks for other system developments. For example, homomorphic encryption provides a desirable and interesting feature which allows computations directly over encrypted data. However, to date, only specific functionality, e.g. aggregation, can be performed efficiently. The need for different encryption schemes for specific tasks has resulted in various proposals such as order preserving encryption [2] and encrypted keyword search [3]. Both the database and the cryptography communities still show great interest in developing more efficient schemes for specific tasks that query encrypted data including keyword search [3, 4], equality queries [5], range queries [6, 7], and order preserving encryption [2, 8]. These methods sacrifice some degree of data confidentiality for more effective querying on encrypted data and provide different levels of security guarantees. Other proposals sacrifice query efficiency for stronger data confidentiality. Examples include homomorphic encryption and predicate encryption, which enable numerical computations on encrypted data without the need for decryption [9–11]. These have been shown to be quite expensive, and thus not practical [12].

1.1.2 Privacy preserving data mining

Although data confidentiality is important, the current cloud services highly depend on mining useful information from data to provide better data analysis, validation and publishing. The data is not useful if there is no information extracted from it. However, it is still crucial to be able to preserve the privacy of data, while exchanging or extracting some information out of it. The goal of privacy preserving algorithms is to allow data mining and analysis to be carried out over the confidential data without revealing sensitive information. The most common approach that seeks to hide sensitive information of data records is referred to as *anonymization* [13, 14]. In this approach, the sensitive data must be retained for analysis while removing explicit identifiers of data records. Even if all explicit identifiers are removed, it is possible

to recover an individual's information by combining and linking distinct data sets [15]. A major challenge in privacy preserving data mining is to preserve both data privacy and information usefulness in the anonymized data. Anonymization can be produced in several ways: generalization [16–18], suppression [16,17,19], anatomization [20], permutation [21], and perturbation [22–24]. *k-Anonymity* [25] and its refinements, *l-diversity* [26] and *t-closeness* [27] have been proposed to prevent privacy leakage. Another privacy model family focuses on how the attacker would change her probabilistic belief on sensitive information of a victim after accessing published data. Dwork proposed ϵ-*Differential privacy* to ensure that the removal or addition of a single record does not significantly affect the outcome of any analysis [28]. ϵ-*Differential privacy* provides a strong notion of privacy and is commonly used for statistical data publishing. Another setting for privacy preserving data publishing, *secure multiparty computation*, is running an algorithm over data which is divided among two or more different parties. The aim is to run a function on the union of the parties' data inventories without allowing any party to reveal another party's private data [29]. After an execution of this function, the parties learn the correct output but nothing else, even if some parties try to obtain more information by colluding. There are well established notions of privacy preserving data mining techniques which have been shown great interest by the researchers in the computer science community. However, the adoption of these techniques by the researchers in other communities and the practitioners in the industry is still very rare.

1.1.3 Access privacy

Although it is necessary, encryption alone is not sufficient to solve all privacy challenges posed by the outsourcing of private data. Indeed, if *access patterns* are *not* hidden from the cloud provider, the provider could detect, for example, whether and when the same data item is repeatedly accessed, even if it does not learn the actual content of the item. This is a real threat

to the privacy of outsourced data, as data access patterns can leak sensitive information using prior knowledge. For example, Islam et al. [30] showed a concrete inference attack against an encrypted e-mail repository exploiting access patterns alone. *Oblivious RAM* (ORAM) – a cryptographic primitive originally proposed by Goldreich and Ostrovsky [31, 32] as a solution for software protection – is the standard approach to make access patterns *oblivious*. ORAM shuffles and re-encrypts data in each data access, making access patterns from any two equally long sequences of read/write operations completely indistinguishable. Hiding access patterns was initially considered in the context of memory access [32]. 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 novel features and fast adoption of the cloud gave impetus to the research community to develop new secure data services in the past several years and many ORAM schemes have been constructed for secure cloud storage systems [33–38]. However, many of these constructions does not capture the entire picture of system service deployment over wide are networks.

1.2 Dissertation Overview

This dissertation focuses on the design, implementation and evaluation of high performance cloud data services without sacrificing the security and privacy requirements by finding the correct balance between security/privacy and fucntionality/performance. The thesis of this dissertation states that *it is possible to deliver reasonably practical and high performance data centric services without compromising any sensitive information with the correct system design and selection of cryptographic and privacy-preserving techniques that provides strong data security and privacy.* Based on this principle, we develop protocols, algorithms and frameworks towards practical privacy preserving systems to address problems. We address these problems

in three parts. The first part solves the problem of range query processing over outsourced encrypted data in a secure and efficient manner. The second part focuses on designing privacy preserving platforms for sustainability metrics that allow environmental researchers and industrial practitioners to perform privacy preserving computations for better environmental decisions and transparency. The last part focuses on ensuring access privacy in the cloud setting.

To solve the problem of range query processing over encrypted data, we advocate a novel approach that brings efficiency without sacrificing sound formal privacy guarantees. We suggest to benefit from the most recent advances in the field of privacy-preserving data publishing by *using cryptography with differential privacy for performing selection range queries*. Our solution proposes to send two complementary data structures to the cloud: an encrypted version of the database (*e.g.,* AES encryption scheme) indexed by a hierarchy of histograms, such that both are perturbed to satisfy differential privacy. Efficiency comes from the disclosure of the index, in the clear, to the cloud, for guiding the query execution strategy. No computation is ever performed on encrypted data. Privacy comes from the differential privacy guarantees of the function that computes the encrypted database and the index. Indeed, the differential privacy model is today's *de facto* standard for protecting personal information that needs to be partially disclosed. It applies to the functions computed on personal data, and defines privacy as a limit on the impact that any possible record may have on their outputs. This new efficiency/privacy tradeoff, however, comes at a cost: the differentially private perturbation makes the index inaccurate. There may be some records that are relevant to the query that will not be retrieved (false negatives), and there may be some irrelevant records that will actually be returned (false positives). After receiving the return set, clients perform post-processing to filter out false positive records. This lower precision is the inevitable cost of increased privacy. A specific query processing strategy must be designed to cope with such inaccuracies, as well as an update management system to maintain the data structures when the original database is

updated (inserts, deletes, or modifications of records) without jeopardizing the privacy guarantees.

This dissertation also explores privacy preserving data mining in the context of environmental science. In an era of ever increasing demands on Earths resources, it is vitally important to have accurate and reliable information about the environmental impacts of industrial activities. The *life cycle inventory* (LCI) data required to make such estimates are vast, varied, and diffuse, known independently by industrial process operators and firms. Contemporary collections of LCI data, on the other hand, are static, centrally maintained, and often mutually inconsistent. Because the data represent operational information about commercial and industrial activities, companies are hesitant to share information with third-party data managers without strict limits on distribution and review. LCI data resources therefore tend also to lack transparency, and validation and interpretation of reported results is challenging. When the objective is to make a quantitative evaluation about the ecological sustainability of a product or service, approaches that consider the full life cycle of the product are often used [39]. This form of analysis, known as life cycle assessment (LCA), is codified in the ISO 14044 standard [40].

Despite the importance of data privacy, the LCA community lacks a formal framework for managing private data, and very limited number of techniques exist for computing sustainability metrics that preserve the privacy of input data. To solve this problem, we formally define the *privacy preserving certification* paradigm along with its goal, security and computation requirements. A certification is a quantitative evaluation of the result of such a computation, or an evaluation of a given contribution with respect to the result. We propose a novel privacy-preserving certification framework that enables an authorized party, referred to as *certifier*, to certify participants based on industrially well agreed on set of criteria or a common function without compromising any sensitive/confidential information to any other parties even in the presence of colluding parties. The framework does not require parties to communicate with each other and aims to minimize the rounds of communication between the parties and the

certifier. We propose efficient algorithms to perform certification operations for the certification problems-mean, quantile- using the proposed framework. We show that the proposed algorithms are correct and secure with the assumption of semi-honest parties. Furthermore, we discuss the efficiency of our algorithms both empirically and analytically.

To solve the problem of publishing more transparent LCA studies, we formulate the LCA computation in a way that allows us to introduce a privacy model, and consider possible threat models and attacks that could result in an adversary learning private data. Our goal in this dissertation is to provide the data security community with a real sense of the challenges faced by practitioners in the field of Industrial Ecology. We explore a particular problem in LCA and explore the privacy issues and possible trade-offs between increase transparency by industrial companies and privacy protection of trade secrets that preserve competitive edge. The results of our attacks justify the concerns over publishing inventory data about industrial processes without securing with any security. To tackle this problem, we apply privacy techniques to LCA computations and illustrate their usage on a specific real life example. Our evaluations on a real life example highlight that it is possible to achieve privacy-preserving LCA publication without losing too much utility on the published data while ensuring privacy with the application of differential privacy.

To prevent leaking sensitive information from outsourced data, it is necessary to make access patterns oblivious. We observe that the earlier constructions have not captured crucial security issues related to asynchronicity in oblivious cloud storage. Therefore, we develop a comprehensive security framework. In addition, in this dissertation, we design and evaluate a new provably secure system, called TaoStore, that fully resists attacks in asynchronous settings and also leverages the benefits of asynchronicity for better performance.

To allow oblivious cloud storage systems to continue operating in the presence of failures, we introduce the first formal study of fault-tolerance for oblivious data storage systems. We develop *generic* fault-tolerance models for a wide class of oblivious cloud systems that con-

sists of a *trusted proxy* and *untrusted cloud storage* for outsourcing the data. For concreteness, we use our oblivious, multi-client cloud storage system, TaoStore. The failure model considers network partitioning and server crash failures. To overcome such failures, we propose *quorum* based replication strategies for three distinct deployment models: 1) a simple storage replication, 2) centralized replication with the help of a coordinator, and 3) fully distributed replication. Quorum based replication strategies have been used to increase the availability of distributed data. Considering the high computational and disk I/O cost of oblivious cloud storage systems, quorum based replication strategies are good fit for oblivious cloud storage systems. However, the selection of the specific quorum model has a direct impact on the sizes of read and write quorums, and the number of tolerated failures. We evaluate each deployment model separately and develop model specific quorum requirements to ensure correctness while hiding access patterns.

1.3 Research Contributions

This dissertation makes several contributions towards practical data security and privacy for outsourced databases in the cloud. In particular, we have made the following contributions:

- A novel privacy preserving range query execution framework, PINED-RQ. This work is the first to construct a differentially private index to an outsourced encrypted dataset. Efficiency is enabled by the fact that the cloud uses a cleartext index structure to perform range query processing. Security relies on both differential privacy (of the index) and semantic security (of the encrypted dataset). PINED-RQ develops algorithms for building and updating the differentially private index while minimizing privacy budget consumption.

- An introduction of a new paradigm, called privacy-preserving certification, that enables

the multi-party computation of sustainability indicators in a privacy-preserving manner, allowing firms to be classified based on their individual performance without revealing sensitive information to the certifier, other parties, or the public. In this work, we describe different variants of the certification problem, highlight the necessary security requirements, and propose a provably-secure novel framework that performs the certification operations under the management of an authorized, yet untrusted, party without compromising confidential information.

- A study of privacy in the context of LCA. The main goal is to explore the privacy challenges in sustainability assessment considering the protection of trade secrets while increasing transparency of industrial activities. To overcome privacy concerns, we apply differential privacy to LCA computations considering the idiosyncratic features of LCA data. We also perform the first formal privacy preserving LCA study on a specific real-life example *distiller grain*.

- The first *formal* study of asynchronicity in oblivious storage systems. We provide security definitions for scenarios where both client requests and network communication are asynchronous (and in fact, even adversarially scheduled). We propose 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.

- Fully implemented version of TaoStore, which was created to be useable not only for research purposes, but also real world applications. As such, this version of TaoStore, which will be made open source. On top of this implementation, we develop *Guess the Access*, an educational game which highlights the important security features of oblivious storage systems, and showcases how TaoStore achieves access pattern privacy, even

in the presence of an adversary, while still delivering high throughput. Our overall goal is to provide the database community with an opportunity to appreciate some of the intricate issues involved in the development and understanding of access security, specifically in a distributed cloud-based data management setting.

- The first formal study of fault-tolerance in oblivious cloud storage systems. Considering privacy as a first class system requirement, this work presents three quorum based replication models, from a centralized replication to a fully distributed one, to tolerate different system component failures for a wide class of oblivious storage systems that depend on a trusted proxy.

1.4 Dissertation Organization

Chapter 2 presents PINED-RQ, privacy-preserving idex for encrypted databases dedicated to range queries, which achieves substantial performance gain compared to the earlier works thanks to the joint use of differential privacy along with semantically secure encryption schemes. Chapter 3 presents a novel framework that allows environmental competitors to benchmark their performances without revealing any individual input. This is followed by the discussion of differentially private LCA computations that allow environmental scientist and industry practitioners to publish more transparent LCA results without revealing any company secrets. Chapter 5 introduces the first formal study of asynchronicity in oblivious cloud storage systems and a novel oblivious cloud storage system TaoStore. In Chapter 6, we introduce the first formal study of fault-tolerance for oblivious data storage systems. Finally, Chapter 7 concludes the dissertation and outlines future research directions.

Part I

Range Query Processing Over Encrypted Data

Chapter 2

PINED-RQ

During the last decade, a large body of academic work has proposed to tackle the problem of outsourcing databases to an untrusted cloud while maintaining both confidentiality and SQL-like querying functionality (at least partially). However, to the best of our knowledge, *performing range queries efficiently* in this context has only been addressed in an unsatisfactory manner. Range queries express a restriction over the retrieved records in the form an upper and a lower boundary. They are fundamental database operations. For example, assume that a university has outsourced its student database to the cloud. A teacher who wants to retrieve the records of students with a grade between A and B could issue the following SQL range query: `SELECT * FROM` students `WHERE` grade ≥ 3.0 `AND` grade ≤ 4.0. Most related work has essentially focused on trading efficiency with security. In particular, they either allow unacceptable security leakage or employ costly cryptographic computations. For example, bucketization techniques [6] do not provide formal privacy guarantees and order preserving encryption schemes (OPE) [2,41] are vulnerable to statistical attacks, while searchable symmetric encryption schemes [42, 43] suffer from execution times that are incompatible with real-world efficiency requirements.

In this dissertation, we advocate a novel approach that brings efficiency without sacrificing sound formal privacy guarantees. We suggest to benefit from the most recent advances in the field of privacy-preserving data publishing by *using cryptography with differential privacy for performing selection range queries*. Our solution proposes to send two complementary data structures to the cloud: an encrypted version of the database (*e.g.,* AES encryption scheme) indexed by a hierarchy of histograms, such that both are perturbed to satisfy differential privacy. Efficiency comes from the disclosure of the index, in the clear, to the cloud, for guiding the query execution strategy. No computation is ever performed on encrypted data. Privacy comes from the differential privacy guarantees of the function that computes the encrypted database and the index. Indeed, the differential privacy model is today's *de facto* standard for protecting personal information that needs to be partially disclosed. It applies to the functions computed on personal data, and defines privacy as a limit on the impact that any possible record may have on their outputs. This new efficiency/privacy tradeoff, however, comes at a cost: the differentially private perturbation makes the index inaccurate. There may be some records that are relevant to the query that will not be retrieved (false negatives), and there may be some irrelevant records that will actually be returned (false positives). After receiving the return set, clients perform post-processing to filter out false positive records. This lower precision is the inevitable cost of increased privacy. A specific query processing strategy must be designed to cope with such inaccuracies, as well as an update management system to maintain the data structures when the original database is updated (inserts, deletes, or modifications of records) without jeopardizing the privacy guarantees.

We propose PINED-RQ (**P**rivacy-preserving **IN**dex for **E**ncrypted **D**atabases dedicated to **R**ange **Q**ueries) which makes the following contributions:

1. A differentially private function computing the encrypted dataset and its hierarchical index.

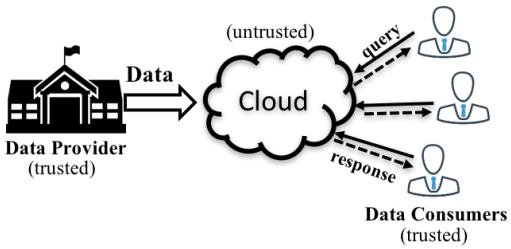


Figure 2.1: System Model

2. An update strategy for managing the insertion, deletion, and modification of records on the cloud.

3. Formal proofs showing the security of PINED-RQ.

4. A thorough empirical evaluation demonstrating the efficiency and quality of the query processing strategy.

To the best of our knowledge, this is the first work that builds, uses, and maintains a differentially private index for performing selection range queries.

2.1 Problem Definition

This section introduces the components of PINED-RQ, and describes the basic data structures and the targeted security model.

2.1.1 Basic Components

Trusted Part. Raw data is produced by the *data provider* and queried by *data consumers*. The data provider encrypts the data and creates a differentially private index structure, both of which are sent to the cloud. Both the data provider and data consumers are part of the trusted part of the system(see Figure 2.1), they are considered to be *honest*. For example, in a

university setting, the data provider would be the information system of the university and the data consumers are the teachers or the administrative staff.

Untrusted Part. The cloud is considered untrusted. It is in charge of storing the data outsourced by the data provider and of processing the queries posed by data consumers. We assume that the cloud follows the *honest-but-curious* attack model: it records every bit of information originating from its exchanges with the trusted part of the architecture and may infer anything that can be inferred in a computationally-feasible way.

2.1.2 Basic Data Structures

The dataset stored by the data provider is a relation $\mathcal{D}(A_1, \ldots, A_d)$ where each A_i is an attribute. Queries are non-agregate single-dimensional *range queries*, over a single attribute A_q, coming from data consumers. A query $\mathcal{Q}$ consists of a set of disjunctions of non-overlapping ranges over A_q : $\mathcal{Q} \leftarrow \phi_1 \vee \ldots \vee \phi_l$ where each ϕ_i is a range defined by a minimum and a maximum value, resp. $\phi_i.min$ and $\phi_i.max$, such that $\cap_{\forall i}\phi_i = \emptyset$. Without loss of generality, we assume a query $\mathcal{Q}$ consists of a single range. Queries are sent *in the clear* to the cloud, without leading to any security breach. The attributes of $\mathcal{D}$ can be of any type, except A_q which must be a totally ordered data type to allow range queries. The set of records in $\mathcal{D}$ that satisfy $\mathcal{Q}$ exactly is called the set of *relevant records* and is denoted $\mathcal{R}$.

In order for the cloud to process queries, the data provider provides the following two data structures to the cloud:

- An encrypted version of the dataset denoted $\overline{\mathcal{D}}$. The encryption $\overline{r}$ of a record $r \in \mathcal{D}$ is performed by concatenating the bits of the attribute values of r and encrypting the resulting bitstring by a *semantically-secure* encryption scheme [44], which means that no probabilistic polynomial-time algorithm is able to gain additional knowledge on a record given its encrypted version and (possibly) auxiliary information (*e.g.,* AES in CBC mode).

Loosely speaking, semantic security implies that no information leaks about a cleartext bitstring given its encrypted value.

- An *index*, denoted $\mathcal{I}(A_q)$, over the queriable attribute of $\mathcal{D}$, A_q, computed from $\mathcal{D}$ but pointing to the encrypted records $\overline{r} \in \overline{\mathcal{D}}$. The index is sent in the clear to the cloud. The information in the nodes of the index structure are randomly perturbed so that *differential privacy* is satisfied.

The differentially private perturbation of the index structure results in an inherent approximation in the set of records that is returned: false positives may be returned while false negatives may be omitted. The recall and precision of an approximate set of records returned by the cloud are defined as follows.

Definition 2.1.1 (Recall and Precision) *Given a query $\mathcal{Q}$, with an exact set of relevant records $\mathcal{R}$ in $\mathcal{D}$, while the set of records returned by the cloud is $\widetilde{\mathcal{R}}$, then the recall* $\mathtt{r}$ *and precision* $\mathtt{p}$ *of $\widetilde{\mathcal{R}}$ are :* $\mathtt{r} = |\mathcal{R} \cap \widetilde{\mathcal{R}}|/|\mathcal{R}|$ *and* $\mathtt{p} = |\mathcal{R} \cap \widetilde{\mathcal{R}}|/|\widetilde{\mathcal{R}}|$.

2.1.3 Privacy

Differential Privacy. The ϵ-differential privacy model [28] requires that whatever the output of an (ϵ-differentially private) function, the probability that any given individual record $r \in \mathcal{D}(A_1, \ldots, A_n)$ is present in the dataset is close to the probability that r is absent by an e^{ϵ} factor. Definition 2.1.2 gives a formal statement of the ϵ-differential privacy criterion.

Definition 2.1.2 (ϵ-differential privacy (from [28,45])) *A randomized function* $\mathtt{f}$ *satisfies ϵ-differential privacy if:*

$$\mathtt{Pr}(\mathtt{f}(\mathcal{D}_1) = \mathcal{O}) \leq e^{\epsilon} \cdot \mathtt{Pr}(\mathtt{f}(\mathcal{D}_2) = \mathcal{O})$$

for any set $\mathcal{O} \in Range(\mathtt{f})$ and any dataset $\mathcal{D}_1$ and $\mathcal{D}_2$ that differ in at most one record.

In our context, the function that must satisfy ϵ-differential privacy is the algorithm that takes a dataset $\mathcal{D}$ as input and outputs the two relevant data structures: the index $\mathcal{I}(A_q)$ and the encrypted dataset $\overline{\mathcal{D}}$. It is based on the well-known Laplace mechanism (Definition 2.1.3), shown to satisfy ϵ-differential privacy [28], and on the composability properties of the ϵ-differential privacy model [46]. Note that we use the ϵ-differential privacy variant of differential privacy without loss of generality.

Definition 2.1.3 (Laplace mechanism (from [47])) *Let D_1 and D_2 be any two datasets such that D_2 can be obtained from D_1 by changing the value of one individual record. Let* $\mathtt{f}$ *be a real-valued function. Let $\mathcal{L}(\lambda)$ denote a random variable which has a Laplace distribution with probability density function $pdf(x, \lambda) = \frac{1}{2\lambda} \cdot e^{-|x|/\lambda}$. The Laplace mechanism consists of adding $\mathcal{L}(\max \|\mathtt{f}(D_1) - \mathtt{f}(D_2)\|_1/\epsilon)$ to the output of* $\mathtt{f}$*, where $\epsilon > 0$.*

Theorem 2.1.4 (Compositions (from [46])) *Let $(\mathtt{f}_1, \ldots, \mathtt{f}_n)$ be a sequence of real-valued functions each satisfying ϵ_i-differential privacy. This sequence of functions satisfies (1) $(\sum_{i=1}^{n} \epsilon_i)$-differential privacy when applied to the same dataset (*sequential*), and (2) $(\max(\epsilon_i))$-differential privacy when applied to disjoint datasets (*parallel*).*

Privacy Model. Our approach is *private* against an honest-but-curious cloud if and only if the cloud learns nothing about the records in $\mathcal{D}$ that do not satisfy ϵ-differential privacy. We formalize below this requirement by intertwining differential privacy with the notion of computational indistinguishability well-known in cryptography [44].

Let π be an instance of PINED-RQ executed on a data provider, a cloud, and a set of data consumers, and Δ be the union of (1) the information output by the differentially private functions run by π (defined later in Section 2.2), and (2) the set of all the information independent from the input original dataset $\mathcal{D}$ produced during the execution of π. Loosely speaking, Δ consists of the information whose disclosure is tolerated. Given $\mathcal{D}$, an attacking cloud $\mathtt{A}$,

and arbitrary background knowledge $\zeta \in \{0,1\}^*$ (*e.g.,* obtained from an other external data source), we define $\texttt{REAL}_{\pi,\texttt{A}(\zeta,\Delta)}(\mathcal{D})$ as the distribution representing the adversarial knowledge over the input dataset in the real setting. Note that for clarity of notation, Δ and π appear separately in the definition of REAL, Δ being an explicit input to the adversary. Similarly, we define $\texttt{IDEAL}_{\texttt{qp},\texttt{A}(\zeta,\Delta)}(\mathcal{D})$ as the distribution representing the adversarial knowledge in an ideal setting where a trusted third party would perform a classical query processing algorithm qp (no untrusted cloud). The ideal setting adversary is abstract. It is simply defined to show that the untrusted cloud in the real setting does not gain additional knowledge about any data-dependent information that would not satisfy differential privacy. This is the reason why the ideal adversary A above also takes Δ as input.

Definition 2.1.5 *We say that* π privately computes qp *iff for every probabilistic polynomial time adversary* $\texttt{A}_r$ *attacking* π*, there exists a probabilistic polynomial time adversary* $\texttt{A}_i$ *for the ideal model so that for every* $\zeta \in \{0,1\}^*$*:*

$$\{\texttt{REAL}_{\pi,\texttt{A}_r(\zeta,\Delta)}(\mathcal{D})\}_{\mathcal{D}} \stackrel{c}{\equiv} \{\texttt{IDEAL}_{\texttt{qp},\texttt{A}_i(\zeta,\Delta)}(\mathcal{D})\}_{\mathcal{D}}$$

In other words, our security model requires that the untrusted cloud does not learn anything about the dataset beyond the information that has explicitly been sent by the data provider after having been perturbed to satisfy differential privacy.

We would like to stress that in this work we do not focus on data access privacy, *i.e.*, protecting the interests of data consumers against information leakage in queries. This issue can be dealt with by using the techniques proposed in this dissertation that will be explained later in Chapter 5.

2.1.4 Problem in a Nutshell

The problem that we address is the design of (1) the differentially private function in charge of computing the two complementary data structures $\mathcal{I}(A_q)$ and $\overline{\mathcal{D}}$ at the data provider to send to the cloud, (2) the query processing strategies of the cloud that use $\mathcal{I}(A_q)$ and $\overline{\mathcal{D}}$ for answering cleartext range queries $\mathcal{Q}$ while ensuring high recall and precision, (3) an update management strategy for handling updates over $\mathcal{D}$, (4) while ensuring that no information that would not have been perturbed by a differentially-private mechanism leaks about the records $r \in \mathcal{D}$.

2.2 Read-Only Context

In this section, we discuss the basic design of the PINED-RQ data structures and query processing strategy in a read-only context. We generalize them in Section 2.3 to tackle a dynamic context where records can be deleted/inserted/modified.

Throughout the section, we refer to the sample example depicted in Figure 2.2. The aim is to publish a student GPA dataset to a cloud server privately. The original dataset has 7 student records with GPAs from 0 to 4. Initially, a clear index is constructed for the dataset where leaf nodes point to a number of tuples corresponding the GPA value. Then, the index entries are perturbed with differentially private noise which results in adding dummy records or deleting some records. In the example, 1 dummy record is inserted to the node with range $[2, 3)$ and 1 dummy record is removed from the node with range $[0, 1)$. After the perturbation, the records are encrypted with a semantically secure encryption scheme. Finally, the cloud is provided the encrypted dataset $\overline{\mathcal{D}}$ (also called encrypted dataset for short) and the cleartext differentially private index $\mathcal{I}(A_q)$ over $\overline{\mathcal{D}}$.

Original schema	
$\mathcal{D}$	Original dataset.
A_q	Queriable attribute.
Perturbed hierarchy of histograms	
$\mathcal{I}(A_q)$	Index.
h	Height of the index.
Φ	Set of ranges in an histogram.
l_i	length of a bin at level i ($0 \leq i \leq h$).
bf	Branching factor.
ϵ_{total}	Privacy budget: total available.
ϵ_i	Privacy budget: for level i in $\mathcal{I}(A_q)$ ($\forall 0 \leq i \leq h$).
Encrypted dataset	
$\overline{\mathcal{D}}$	Encrypted dataset.
χ	Secret encryption/decryption key.

Table 2.1: Notations

2.2.1 Data Structures

We describe below the two data structures provided to the cloud, *i.e.,* $\overline{\mathcal{D}}$ and $\mathcal{I}(A_q)$, showing for each that its computation satisfies ϵ-differential privacy, and demonstrating the end-to-end security of the function that computes them. Table 2.1 summarizes the notations used.

The first data structure computed by the data provider is the differentially private index computed over the dataset. Designing this index is hard because it must address the conflicting goals of allowing the retrieval of the records in a given range query, while satisfying differential privacy. Both false negatives and positives are thus inherent, the challenge lies in reducing them to realistic numbers. To this end, we propose to benefit from two fruitful research tracks: on the one hand, from the B+-Tree family of indices, widely used in traditional databases for supporting range queries over cleartext relational data, and on the other hand, from the more recent differentially private hierarchies of histograms, that have been shown to answer *aggregate* range queries accurately while satisfying differential privacy [48–50]. Index $\mathcal{I}(A_q)$ is essentially a balanced tree over the encrypted dataset $\overline{\mathcal{D}}$ in which each *node* consists of a fixed-size pointers.

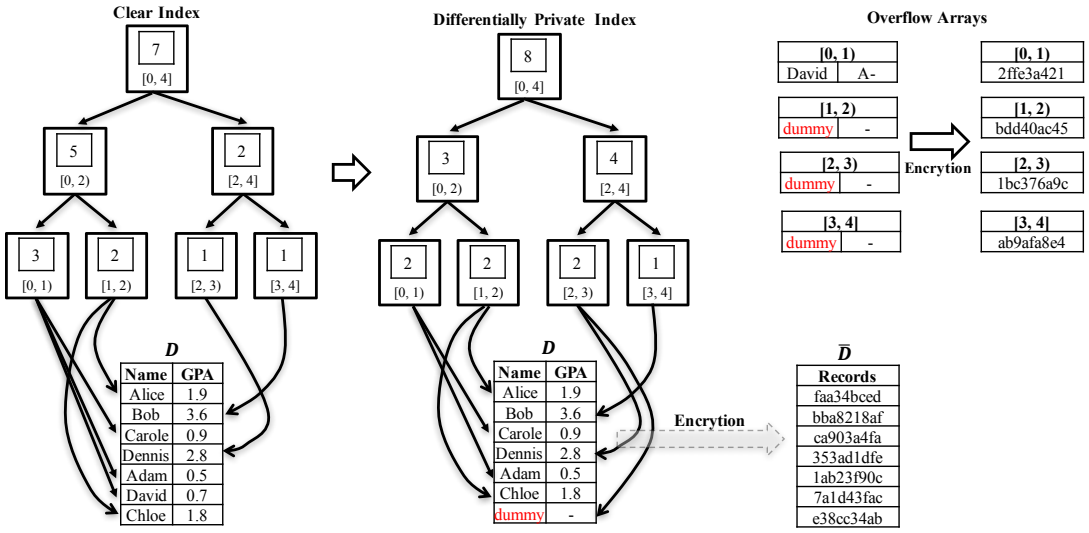


Figure 2.2: Sample Publication

Basics : nodes and histograms In a B+-Tree, a key is a single value (*e.g.,* an integer) that indicates the range within which fall the records that can be accessed while following the pointer associated. In our context, rather than using single values as indications for ranges, we propose to use *histograms*. Indeed, histograms can be used as accurate estimators for ranges through the distributions they disclose. Histograms are defined below in Definition 2.2.1. Histograms are also well known for integrating well with differential privacy [48–50]: the Laplace mechanism and the parallel composition theorem together allow adding a random variable sampled independently in $\mathcal{L}(1/\epsilon)$ to each bin in order to satisfy ϵ-differential privacy.

Definition 2.2.1 (Histogram and Histogram Bins) *Let* $\Phi \leftarrow (\phi_1, \ldots, \phi_k)$ *be a set of non-overlapping ranges that partition the domain of the queriable attribute* A_q*. Each range* $\phi_i \in \Phi$ *is associated to its corresponding* bin b_i *where* b_i *stores the number of data records within the range of* ϕ_i*. Each bin belongs to a unique node (defined in Definition 2.2.2) in the index. A histogram* h *is a complete set of bins over the complete domain.*

Definition 2.2.2 (Node) *A node is a histogram bin/pointer pair, where the pointer is a reference to either a child node or encrypted data records. Each node represents a bin of a histogram.*

Considering the example from Figure 2.2, the index structure has 7 nodes where 4 of them are leaf nodes. The leaf nodes have histogram bins $\{[0,1), [1,2), [2,3), [3,4]\}$. 4 leaf nodes together construct a complete histogram at the leaf level.

Building $\mathcal{I}(A_q)$ The hierarchy of nodes in the index structure is built by following a three-step process. We designed this process so that (1) it requires a single pass over the dataset, and (2) it minimizes the distribution of the privacy budget. The first step computes the clear index, *i.e.,* the nodes and the pointers. It does require a single pass over the clear dataset to associate each data record with the corresponding leaf node. Later, the hierarchy of the index is constructed considering the non-private *branching factor*, a system parameter of the structure. The second step perturbs the nodes of the index based on the Laplace mechanism to satisfy differential privacy and post-process them in order to increase its utility. The last step is encrypting data records with semantically secure encryption scheme after constructing the differentially private index. The dataset is scanned twice: 1) during the first step to construct the leaf nodes, a scan that is both necessary and sufficient for computing the histograms, and 2) during the encryption of data records. We explain now each step.

The first step computes each level of the hierarchy iteratively, starting from the *leaf nodes* at the bottom and ending with the single *root node* at the top. First, the leaf nodes (level 0) are instantiated. The number of leaf nodes is computed from the domain of the queriable attribute A_q and a unit-length interval that defines the length of each histogram bin. For example, if the attribute domain is from 0 to 100 and the unit-length interval is 1, then there are 100 leaf nodes with ranges $\{[0,1), [1,2), .., [99,100]\}$. When the unit-length interval is 2, then there are 50 leaf nodes with ranges of $\{[0,2), [2,4), .., [98,100]\}$. Once the leaf nodes are created, then a scan passes over the cleartext dataset by creating a pointer from the corresponding node to the data records and incrementing each bin's counter by 1. When the scan is completed, the leaf nodes have the correct numbers with associated pointers to the actual data records.

Considering the example in Figure 2.2, the leaf nodes of the clear index have counts $3, 2, 1, 1$ for the corresponding ranges of $\{[0, 1), [1, 2), [2, 3), [3, 4]\}$. This means there are 3 data records whose values are within the range of $[0, 1)$ and the first leaf node has pointers to these records. All nodes together at the same level construct a complete histogram over the domain $[0, 4]$. The nodes of the upper levels are then computed such that each upper node points to a set of children nodes. The number of pointers that map to child nodes is set to the branching factor. The range of the bin of a node is a union of the children nodes' ranges, *i.e.,* given Φ^0 is the range partitioning of the leaf nodes $\Phi^0 \leftarrow (\phi_1^0, \ldots, \phi_k^0)$, the upper level range partitioning of the domain is $\Phi^1 \leftarrow \cup_{l=1}^{m} \left(\cup_{l}^{l+bf-1} \phi_l^0 \right)$. The count of the bin is computed by summing the counts of the child nodes. In our toy example, the upper node for the range $[0, 2)$ has a count of 5, which is the summation of two child nodes $[0, 1)$ and $[1, 2)$. As can be seen from the figure, the branching factor for the example is 2. This process goes on iteratively until a single node remains, which is the root node (highest level).

The second step is to perturb the clear index to satisfy differential privacy. Our approach builds on previous works [48, 51]. The differentially private computation of hierarchies of histograms has been extensively studied in the context of analytical queries. Although our context is different, PINED-RQ can benefit from the strategies proposed in [51, 52] for distributing the privacy budget over the levels. Cormode et al. [52] compare a uniform budget allocation to a geometric budget allocation approach. The uniform budget allocation strategy allocates budgets to each level equally such that if the index has h levels, each level is allocated a budget of ϵ/h. In the geometric budget allocation, the allocated budget increases geometrically from the root to the leaves. The root receives the lowest budget, whereas the leaf nodes receive the highest budget. Cormode et al. demonstrate that geometric budgeting outperforms the uniform strategy as the height of an index increases. However, for shallower indexes ($h \leq 5$), both strategies are competitive. Qardaji et al. [51] also explore the effect of privacy budget allocation and do not recommend optimizing the privacy budget allocation as long as the optimal

branching factor is selected. Our approach also uses a uniform budget allocation strategy. The straightforward approach to apply differential privacy is to sample a random noise from the Laplace distribution for each node bin and add the sampled noise to the original count. The total privacy budget is denoted by ϵ_{total}. Consider the example discussed before (Figure 2.2). The root node of the clear index has a count of 7 (range = $[0, 4]$). After the perturbation, the same node has a bin count of 8. This means the sampled noise for the bin is 1. Clearly, differential privacy causes a loss of utility in the index. Qardaji et al. [51] increase the utility of a histogram by generalizing the constrained inference approach proposed in [48]. The constrained inference provides consistency constraints between parent/children histograms, which has also been adopted by [52]. PINED-RQ also adopts the constrained inference approach to increase utility.

Unlike earlier approaches, we tackle a new problem while satisfying the differential privacy of the index. Earlier approaches target answering analytical range queries where the utility loss occurs due to the sampling of noises. However, the noise itself does not cause any problem since it only changes the count in the bins. This is also the case for the inner nodes in our index and our approach also updates the count of the node bins. However, in our context, the leaf nodes point to the data records which brings a novel challenge regarding noise sampling. We use the Laplace mechanism to sample noises and the sampled noises can be negative or positive as well. Therefore, our index construction handles two cases: 1) sampling of a positive noise, and 2) sampling of a negative noise in a special way. Next, we discuss how the index construction handles these issues.

Positive noise sampling at leaf nodes If the noise v sampled from the Laplace mechanism is positive, v dummy records are inserted at uniformly-random positions in the dataset and v additional pointers from the node to the dummy records are created. Assuming bin b_i has a count of c_i, the updated count will be $c_i + v$. Note that each node in the index is perturbed sep-

arately and such a rule applies for each node that is perturbed with positive noise. Considering the example in Figure 2.2, the actual count of the node with bin range $[2, 3)$ is 1 in the clear index. This means there is only one real record in the real dataset that falls into this range. During the perturbation, the sampled noise is $v = 1$ and the bin count of the same node in the differentially private index is 2. The differentially private index is outsourced to the cloud in clear which means an adversary sitting in the cloud is able to see the counts of the nodes. In case a dummy record is not inserted into the actual dataset, the adversary would be able to see that there is only 1 pointer to the dataset from the node with the range $[2, 3)$ even if the bin count is 2. This would allow an adversary to infer the actual number of records within this range regardless of the node's count information. To hide such information from the adversary, inserting dummy records is necessary. This allows us to ensure that the number of pointers pointing to the real dataset matches the count inside the node. The encryption of data records with semantically secure encryption scheme as will be explained later prevents the adversary from distinguishing real records from the dummy records.

Negative noise sampling at leaf nodes The handling of negative noise during the perturbation is a bit more complicated than the positive case. When the sampled noise v is negative, this requires removing some data records from the dataset in order to ensure differential privacy. If bin b_i has a count of c_i, the updated count is $c_i + v < c_i$. Considering the toy example, the node with a range $[0, 1)$ is perturbed with noise -1 and a uniform-randomly selected single record -$(David, 0.7)$ in this example- in this range is removed from the actual dataset. Completely removing records from the dataset results in missing actual data while querying the index. This is not reasonable as removing some number of records might cause a drastic decrease in recall (very low utility). Although removing these records from the actual dataset is enough to ensure the differential privacy of the index, PINED-RQ introduces a new approach to handle removed records without violating differential privacy to achieve higher performance.

To handle removed records, PINED-RQ creates a fixed-sized *overflow array* for each leaf node. Each leaf node is associated with a single overflow array. Once an actual record is removed from a node, it is removed from the actual dataset and inserted into the corresponding overflow array. Considering the example provided above, the removed record from the leaf node with range $[0, 1)$ is inserted into the overflow array for the same node. During the perturbation, the other two leaf nodes are perturbed with noise 0 and one leaf node is perturbed with noise 1 (handling explained above). Therefore, their overflow arrays do not contain any actual records. However, leaving them empty will reveal to an adversary that the data records contained in the overflow arrays are real data records. To ensure privacy, the number of real records in each array should be indistinguishable. To hide the number of records stored in an overflow array, the empty spaces in the arrays will be padded with dummy data, so each overflow array consists of the same number of records. The size can be selected probabilistically large enough to store any removed records. Since the noise is sampled from the Laplace distribution, it can be selected based on the cumulative distribution function of the Laplace distribution with a high confidence like 99.99%. Later, each overflow array will be encrypted after shuffling. This ensures the confidentiality of the removed data. During query processing, all overflow arrays intersecting with a given range will also be included in the response set.

We now show that PINED-RQ is private as defined in Definition 2.1.5 and satisfies ϵ_{total}-differential privacy.

Theorem 2.2.3 *Index $\mathcal{I}(A_q)$ satisfies ϵ_{total}-differential privacy where ϵ_{total} is the budget dedicated to index perturbation,* i.e, $\epsilon_{total} = \sum_{i=0}^{h} \epsilon_i$.

Proof: We consider neighboring datasets as two datasets, $\mathcal{D}_1$ and $\mathcal{D}_2$, differing in at most one record. Each node represents a bin of a histogram and a difference of a single record affects a maximum of one node: either increment or decrement count by 1. The maximum change 1 is used as a *global sensitivity* to sample noises from the Laplace distribution. Each

bin is perturbed with a noise sampled from the Laplace mechanism, i.e., $\mathcal{L}(1/\epsilon_i)$. At each given level l_i, the set of bins (of histogram) satisfy ϵ_i-differential privacy (parallel composition theorem). Second, any given root-to-leaf path satisfies $\sum_i \epsilon_i$-differential privacy. As a result, thanks to the distribution of the privacy budget, $\sum_i \epsilon_i = \epsilon_{total}$. Moreover, since the composition of any function with a differentially-private function also satisfies differential privacy, the post-processing of the histograms in $\mathcal{I}(A_q)$ also satisfies ϵ_{total}-differential privacy. ■

Once PINED-RQ constructs the differentially private index, it encrypts each data record in $\mathcal{D}$ and overflow arrays with a semantically secure symmetric encryption scheme parameterized with the secret key χ while preserving the pointer relation. The output of the encryption is $\overline{\mathcal{D}}$, a sequence of random bitstrings, and encrypted overflow arrays. Note that there is no way to identify dummy records from the actual ones neither in $\overline{\mathcal{D}}$ nor in encrypted overflow arrays.

Theorem 2.2.4 *Let PINED-RQ build $\mathcal{I}(A_q)$ and encrypt data records with the semantically secure encryption scheme, and then send the encrypted dataset output along with $\mathcal{I}(A_q)$ and encrypted overflow arrays to the cloud. Let π also be an instance of PINED-RQ. Then, π is private and ϵ_{total}-differential privacy is satisfied.*

Proof:

Consider two neighboring datasets $\mathcal{D}_1$ and $\mathcal{D}_2$. Theorem 2.2.3 proves that $\mathcal{I}(A_q)$ is ϵ_{total}-differentially private whether it inputs $\mathcal{D}_1$ or $\mathcal{D}_2$. It is easy to see that an adversary does not infer any information about $\overline{\mathcal{D}}$ since it is encrypted with semantically secure encryption scheme. The adversary cannot distinguish real records from the dummy ones as well. The additional data structures constructed during the index construction are overflow arrays. The size of overflow arrays is fixed and the creation of overflow arrays is independent of the perturbation. Each overflow array has a fixed number of records (either dummy or real) and the number of leaf nodes is independent from the dataset itself. The output is a fix set of encrypted records whose distribution is identical independent of the dataset itself. This does not reveal any information

to an adversary about the dataset. An adversary does not learn anything about the dataset from the published information, therefore, π is private as defined in Definition 2.1.5. ■

Setting the parameters. Our histogram-based index structure shares similarities with the hierarchy of histograms proposed in [51]. Two important parameters, *branching factor* and *number of bins*, have significant impact on the accuracy of the index. Qardaji et al. [51] discuss the computation of the branching factor and point out that when the number of bins is high, the optimal branching factor, which minimizes the mean squared error (MSE), is around 16. The empirical analysis in [51] also highlights that the best performance is delivered when the branching factor is selected between 8 and 16. Therefore, the branching factor is also set to 16 in PINED-RQ.

2.2.2 Query Processing Strategy

PINED-RQ deploys a simple query processing strategy to answer client requests. Given a range query, the query execution starts from the root of the index, and traverses the child of any node that has a non-negative intersection with the provided range. This is repeated recursively until the leaves of the index. In the leaf nodes, if a node has a positive count with the overlapping range query, then PINED-RQ returns the records pointed to by the corresponding node. If a leaf node is reached, independent of the node count, PINED-RQ returns the records in the overflow array for the corresponding node since PINED-RQ prioritizes high recall.

2.3 Updates in a Dynamic Context

This section extends our description of PINED-RQ to support update operations after the initial publication to the cloud server. In our context, updates are either *insert*, *delete* or *modify* operations.

Supporting updates with a differentially private index raises several important issues: 1)

How does the system manage a privacy budget over multiple updates? 2) How are the updates reflected on the published differentially private index and the encrypted storage? 3) How does query processing work after the updates? PINED-RQ supports updates for append-only applications as well as applications with any number of inserts but a finite number of modifications and deletes to the existing records. Append-only data repositories are very popular and have widespread applications. Likewise, many real-world applications have finite updates to existing records while supporting insert operations, for example university and medical databases. In the university case, students in a university have a finite number of updates to their grades, and it is very unlikely to have their grades changed after a student graduates. On the other hand, each year new students register to the university.

The data is published in the cloud with an initial index structure. A simple approach would process updates one by one, but given that the index structure is differentially private and each publication consumes a fraction of the privacy budget, such an approach would exhaust the budget quickly. Therefore, PINED-RQ handles updates in batches. Queries need to be sent to the data provider rather than sending them to the storage server. Hence, the data provider becomes a proxy and mediates the client-storage server communication. Proxy-based secure storage systems have been shown to deliver reasonable performance [53, 54], therefore, utilizing the data provider as a proxy does not cause drastic performance degradation in PINED-RQ.

2.3.1 Processing Updates

Assume that a delete operation on a record r_i is requested. How does PINED-RQ handle this operation without compromising privacy? PINED-RQ does not remove the record from the published index and database - doing so would obviously violate differential privacy (*e.g.,* marking or removing a record directly in the database would reveal that r_i is not a dummy record). Instead, the data provider maintains small databases, called ΔDB and ΔDB_i^j where

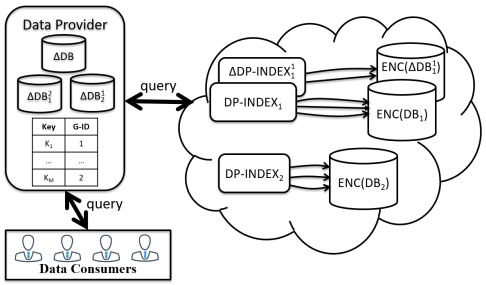


Figure 2.3: Updates in PINED-RQ

i denotes the publication index and j denotes the j^{th} batched modification/delete updates corresponding to the i^{th} publication (depicted in Figure 2.3). PINED-RQ stores new insert operations in ΔDB. ΔDB_i^j are used to store modifications and deletes corresponding to the i^{th} publication, *i.e.,* ΔDB_1^1 and ΔDB_1^2 store updates related to the first publication. When the system is initialized and the initial dataset is published, the data provider stores modifications and deletes in ΔDB_1^1. With incoming modifications and deletes, ΔDB_1^1 grows and the data provider decides to publish ΔDB_1^1 to the cloud at some point. After ΔDB_1^1 is published to the cloud, the data provider stores modifications and deletes related to the first publication set in ΔDB_1^2. Inserts, on the other hand are batched in ΔDB, which also grows until the data provider decides to publish to the cloud as DB_2 with its associated index structure.

In PINED-RQ, each *update* operation is a new insert to either ΔDB or one of the ΔDB_i^j databases. When a new insert operation is requested, the data provider inserts it to ΔDB. Although such behavior is obvious when it is an *insert* operation, *delete* and *modify* queries result in inserting new records to the ΔDB_i^j corresponding to the record being modified/deleted. Each record has an additional attribute which indicates the type of operation, i.e., insert, delete or modify. ΔDB and ΔDB_1^1 are initially empty and after some number of updates, differentially private indexes for them are created. Later, they are published to the server.

Handling Inserts

An interesting feature of inserts that PINED-RQ exploits is that newly inserted records are grouped together and a new group is associated with a new privacy budget. Considering the university example discussed before, each year new students register to the university and they do not have any existing records in the published dataset. Instead of integrating new students to the previous publication, which would imply using the same partially consumed privacy budget, PINED-RQ associates these new students with a new publication and hence with a *full* new privacy budget.

Assume that the initially published dataset contains records with specific key values in attribute A_q, denoted by $\{K_1, ..., K_m\}$. Note that a key might have multiple records in the database, *e.g.,* records r_i and r_j might belong to K_1. Now consider an insert operation related to a new key K_{m+1}. The data provider will note that there is no intersection with earlier publications regarding K_{m+1}. Therefore, this new record can be associated with a new publication set. If there is an intersection with an earlier publication, the data provider would process K_{m+1} as a modify/delete. All keys $K_{1..m}$ are related with the first publication; however, K_{m+1} is mapped to the second publication. The data provider stores new inserts in ΔDB, and, after processing some number of updates, it publishes ΔDB to the cloud by constructing a secure index. Now, the cloud stores two encrypted databases, denoted by DB_1 and DB_2, and two differentially private indexes, DP-INDEX_1 and DP-INDEX_2, which point to DB_1 and DB_2, respectively. After this point, further insert operations will be associated with the third publication. In this way, PINED-RQ can use separate privacy budgets (ϵ_{total}) for each set of publications using parallel composition, which ensures differential privacy.

If the initial publication of a set of inserts consumes the full privacy budget ϵ_{total}, it is not possible to perform further publications associated with that publication set. To allow the system to continue further publications, the initial publication should use an initial privacy

budget denoted by ϵ_{init} such that $\epsilon_{init} < \epsilon_{total}$. The remaining budget is used to perform future publications for the corresponding set of publication.

Handling Modifies and Deletes

Assume that a student objects to her grade after the professor had submitted the grades to the university's grading system. As a consequence of this objection, the professor agrees to change the student's grade. As discussed earlier, a direct modification of a student's grade in the published database violates differential privacy. Therefore, PINED-RQ stores updates related to previous publications in ΔDB_i^j where i corresponds to the publication DB_i that contains the record corresponding to this student, and j refers to the j^{th} update batch of this publication.

Later, the data provider publishes these ΔDB_i^j to the cloud. The main question that needs to be answered is when to perform publications regarding modifications/deletes and how to allocate the privacy budget among multiple publications (publications to the same dataset need to share the budget to satisfy differential privacy). Recall that an initial publication uses a privacy budget of ϵ_{init}. Hence, all future publications have to use the remaining privacy budget, denoted by ϵ_{rem} where $\epsilon_{rem} = \epsilon_{total} - \epsilon_{init}$, to create a differentially private index over ΔDB_i^j and publish to the cloud. As stated earlier, it is possible to perform multiple ΔDB_i^j publications. If this is the case, the remaining budget should also be shared among multiple ΔDB_i^j publications. But the question is *how*? The next sections explain when PINED-RQ decides to publish ΔDB_i^j to the server and how much budget should be allocated to this publication.

When to Publish The straightforward solution would be to perform periodic publications, *i.e.,* after some fixed time or some fixed number of updates. The challenge with this approach is deciding on the parameters. For example, if the data provider publishes the ΔDB_i^j databases too frequently, it will consume the budget too quickly. On the other hand, having rare publications might end up storing too many records at the data provider. It is obvious that there is a trade-off

between executing queries through the data provider and the server. Therefore, we consider the cost in the decision process for performing publications. Storing data at the data provider is more costly than storing at the cloud storage, which is a natural motivation for outsourcing. The cost might depend on many external factors, *e.g.,* location of servers/data provider, bandwidth, etc., but specifying these factors is beyond the scope of this work. We assume that there is a constant cost ratio between storing data at the data provider compared to the server, denoted by α. The decision is made based on the following heuristic approach:

$$\alpha \times Relative\ Size \times \left(1 + \frac{\epsilon_{rem}}{\epsilon_{total}}\right) \geq 2\mu \tag{2.1}$$

where *Relative Size* is the ratio of the data size stored in the data provider compared to the server and μ is a constant threshold parameter. ϵ_{rem} is always less than or equal to ϵ_{total}, *i.e.,* $\epsilon_{rem}/\epsilon_{total} \leq 1$. Thus, the ratio on the left hand side $(1 + \epsilon_{rem}/\epsilon_{total}) \leq 2$. In the best case, this ratio is 2, therefore, μ is multiplied by a constant factor of 2 to match the ratio $(1 + \epsilon_{rem}/\epsilon_{total})$. The right hand side is the minimum threshold that needs to be reached to trigger a publication. Whenever the current ratio at the data provider (the left hand side of the inequality) exceeds the minimum threshold, the data provider constructs an index DP-INDEX$_i^j$ for ΔDB$_i^j$ and publishes the index along with the encrypted ΔDB$_i^j$ to the cloud. A higher μ results in less frequent publications in bigger batches. In our analysis, we found 2 to be a reasonable value for μ, though the system administrator can set the constant to some other value. In addition, higher α triggers more frequent publications, since the execution through the data provider is more costly. Moreover, as ϵ_{rem} decreases, the system tries to delay the publication. When the system is out of budget, no further publications can be performed for the corresponding set of publication, but the system can continue serving modifications/deletes using the data provider.

Allocating the Privacy Budget After deciding to publish ΔDB_i^j to the cloud, PINED-RQ allocates a privacy budget for the publication in a novel way.

Previous work on differentially private updates were proposed in the context of data streams, which have different characteristics than our system. A notable exception is [55] by Zhang et al., which requires a system administrator to decide on the total number of publications k upfront, where each publication consists of updates in batches. The budget is split equally among publications (i.e., each publication receives ϵ/k). However, this deterministic approach does not consider any external factors like dataset sizes. Thus, PINED-RQ uses the perturbed database size as a basis for allocation which considers the relative data storage at the data provider and ϵ_{rem}. Equations 2.2 and 2.3 compute the allocated privacy budget for ΔDB_i^j together.

$$\epsilon_{\Delta DB_i^j} = \epsilon_{rem} \times \frac{Size(\Delta DB_i^j)}{Size(DB_i + \sum_{l=1}^{j} \Delta DB_i^l)} \tag{2.2}$$

As ϵ_{rem} decreases, the system requires a higher ratio between the size of the storage at the data provider versus that in the cloud to perform publication. However, the increase in this ratio results in storing more data in the data provider. As the data stored in the data provider increases, more of the remaining budget is allocated for the publication. Note that it is possible for Equation 2.2 to allocate very small budgets. Therefore, PINED-RQ would allow a system administrator to set a minimum budget threshold for the allocated privacy budget denoted by ϵ_{min}. If $\epsilon_{\Delta DB_i^j}$ is less than ϵ_{min}, the budget for publication $\epsilon_{\Delta DB_i^j}$ is set to ϵ_{min}. Otherwise, the allocated budget is $\epsilon_{\Delta DB_i^j}$. To achieve higher utility, PINED-RQ uses a simple motivation, namely, less budget for bigger datasets, and more budget for smaller datasets.

$$\epsilon_{\Delta DB_i^j} = \max(\epsilon_{\Delta DB_i^j}, \epsilon_{min}) \tag{2.3}$$

Recall that each publication set has its own budget. The data provider performs computations for each set of publication independently to decide on the publication of ΔDB_i^j and its budget $\epsilon_{\Delta DB_i^j}$.

2.3.2 Executing Queries

In the steady state, the server might have multiple indexes, *e.g.,* DP-INDEX$_i$, ΔDP-INDEX$_i^j$, ΔDP-INDEX$_i^{j+1}$, DP-INDEX$_{i+1}$. Clients send queries to the data provider and the data provider partially answers the query based on its local information stored in ΔDB and ΔDB_i^j. In addition, the data provider redirects the query to the server. To retrieve the latest state, the server might need to process a query over all indexes as the data might be deleted or modified. Since each index is independent, parallel execution is possible and existing parallel query execution strategies can be directly applied.

2.4 Performance Evaluation

In this section, we present experimental results that demonstrate the efficiency and practicality of PINED-RQ. We examine the effects of varying system configuration parameters on the overall system performance.

We implemented PINED-RQ in Java. All experiments are conducted on a machine running Windows 7 with i5-2320 3 GHZ CPU and 8 GB memory. We set the branching factor (bf) to 16 and the total privacy budget ϵ_{total} to 1. The domain of A_q is normalized to $[0, 100]$.

Datasets. The experiments were performed with both synthetic and real datasets. To emulate real-world scenarios, the synthetic datasets follow two different distributions: *uniform* or *Zipfian* with a skewness of 1, and contain 0.5 million records. For real datasets, we chose the Gowalla [56], a social networking website where users share their locations by checking-in, and the US Postal Employees [57], called USPS, datasets. The Gowalla dataset consists of

$6,442,890$ check-in records. As a query attribute, we use the 32-bit integer representation of check-in times. The experiments are performed on different sized datasets starting from 0.5 to 5 million records, which are created by choosing records uniformly from the complete Gowalla dataset. The USPS dataset consists of $624,414$ employee records. We use *annual salary* as a query attribute and filtered out employee records that have an hourly payment rate. After filtering, the dataset consists of $394,763$ records. The USPS dataset is highly skewed, whereas Gowalla is relatively uniform.

Query Set. In the experiments, we create various query sets of ranges corresponding to $1\%, 5\%, 10\%,\ 25\%,\ 50\%$, and 75% of the entire domain. For each set of query ranges, we sample 1000 queries uniformly over the domain. Unless stated, all other experiments are conducted using a uniform workload.

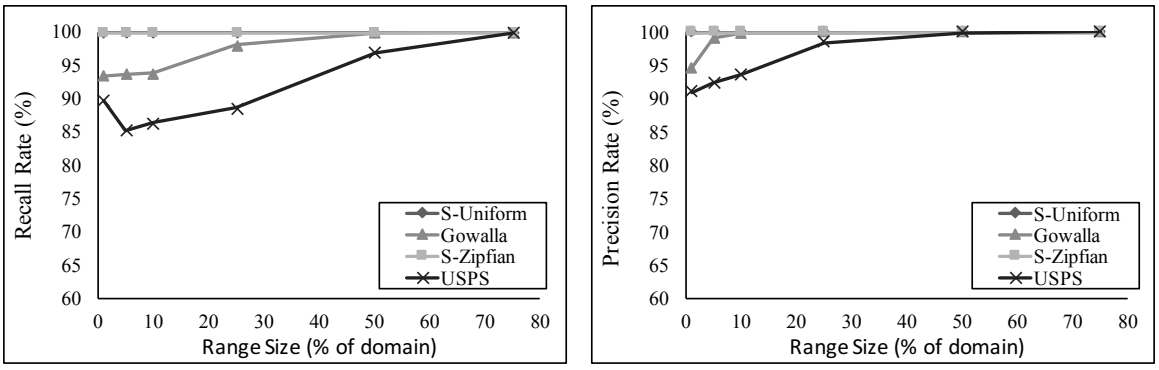
Evaluation metrics. The main metrics to evaluate the performance of PINED-RQ are recall and precision. Note that PINED-RQ constructs a differentially private index; therefore, it is not always possible to return the complete set of true records in a given range. It is also possible to have false records in the returned set. Therefore, the aim of PINED-RQ is to maintain privacy while achieving high recall and keeping the precision as high as possible. In addition to this, we measure the elapsed time for query execution.

2.4.1 Effect of Overflow Arrays

In this set of experiments, we analyze how the index construction mechanism performs with the presence or non-presence of overflow arrays that are constructed to store the removed data records during the perturbation. As discussed earlier, depending on the sampled noise, the recall of a given query might drastically decrease. Our empirical findings validate this claim. These experiments are run on different datasets. The synthetic uniform and skewed datasets are denoted by *S-Uniform* and *S-Zipfian*, respectively. We use a variant of the Gowalla dataset

with 0.5 million records, i.e., similar to the synthetic data sets and approximately equal to the USPS dataset of size $394,763$ records.

Without Overflow Arrays. Figure 2.4 shows the recall and precision results of PINED-RQ over different datasets by varying the query range size without deploying overflow arrays. In this setting, negatively sampled noise might cause the removal of a significant number of data records from the publication. Although PINED-RQ delivers high recall and precision for most of the cases, the performance with lower ranges sizes might not be satisfactory for both real datasets, Gowalla and USPS. For example, PINED-RQ delivers 85.15% recall and 92.26% precision for the USPS dataset when the range size is 5%. This is not the case for the synthetic datasets. During query execution, most of the decisions are made at the leaf level where each node covers 1 unit-length interval. The real datasets do not provide a perfect distribution so some of the nodes might cover very small numbers of records. If such nodes are perturbed with relatively high positive or low negative noises, the returned results consist of some number of false positive records or missed actual records. Therefore, the leaf nodes are more error-prone due to the added differentially private noise if a covered bin has a very low count. This is observed in Gowalla and USPS datasets. Some ranges have very low counts and data removal causes query execution to suffer from low recall (e.g., as low as 85% for 5% range queries on the USPS dataset). This affects the overall recall performance for the USPS and Gowalla datasets. The removal of data records decreases recall for every dataset but this is less observable for the synthetic datasets. The reason is that they follow almost a perfect distribution and leaf node bins have high enough counts(even the skewed one) which make the impact of removed data records negligible. Although, one can argue that PINED-RQ delivers good performance for the synthetic datasets even if there is no overflow array, real world application/datasets usually do not follow perfect data distribution. Therefore, PINED-RQ introduces overflow arrays to further improve the performance.



(a) Recall (b) Precision

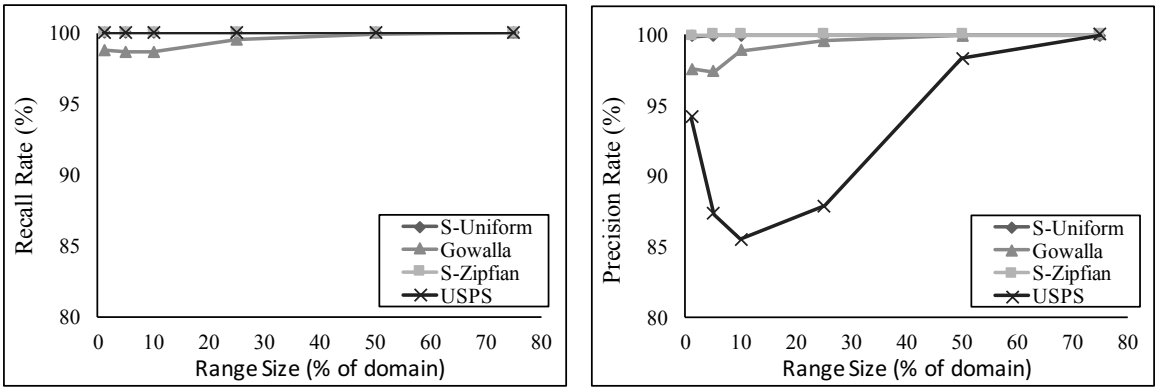
Figure 2.4: PINED-RQ without overflow arrays

With Overflow Arrays. When PINED-RQ deploys overflow arrays, it achieves almost 100% recall for all cases except small ranges executed over Gowalla (see Figure 2.5). Recalls for the small ranges are relatively high ($98.6 - 98.8\%$). PINED-RQ misses some records since the query execution strategy stops traversing the index when overlapped ranges have a negative count. Some nodes have low counts as discussed before and the added differentially private noise misleads the query execution algorithm. Regardless of the dataset and the query range, the deployment of overflow arrays improves the recall performance. This is also true for precision except in the case of USPS. While achieving higher recall, the precision drops to 85.52% from 93.53 when the query size is 10%. Even in the worst case, achieving a 85.52% precision is decent. Note that both recall and precision are important in evaluating the system performance. In this context, recall is more crucial and higher recall is more preferable at a cost of lower precision most of the time.

Please note that the rest of the experiments all use overflow arrays unless otherwise stated.

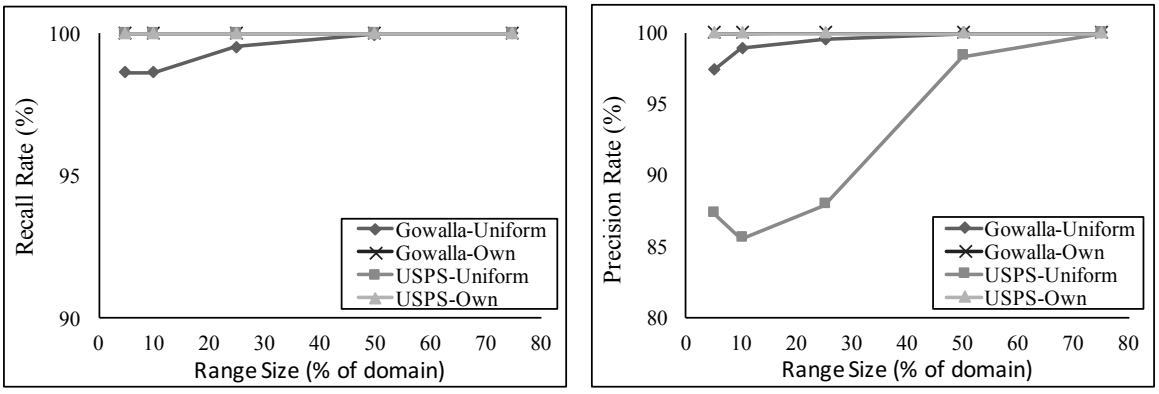
2.4.2 Skewness & Workload

When the data distribution is skewed, the precision performance of PINED-RQ is affected by the workload type (*e.g.,* uniform, skewed). As discussed before, if a query range covers



(a) Recall (b) Precision

Figure 2.5: PINED-RQ with with overflow arrays



(a) Recall (b) Precision

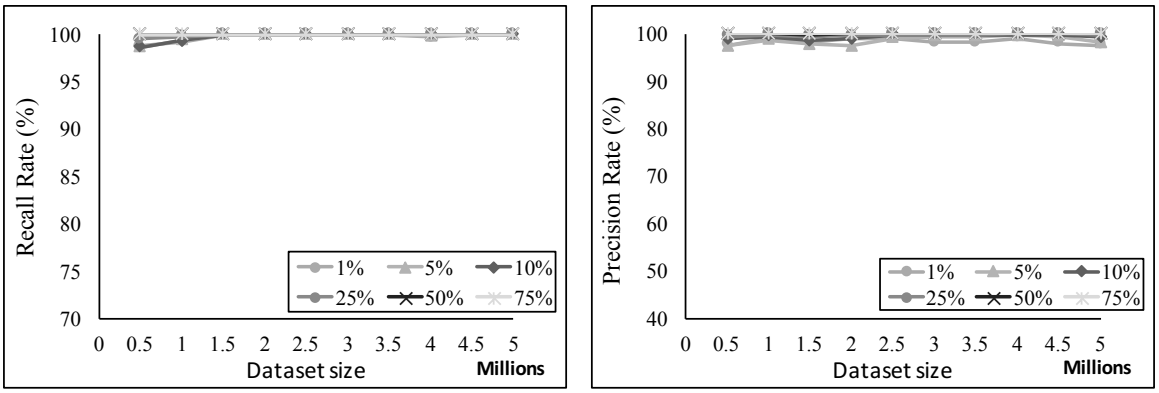
Figure 2.6: Effect of workloads

the skewed area, PINED-RQ delivers high recall and precision. We analyzed this case in more detail and Figure 2.6 presents our findings. In this set of experiments, we use datasets *Gowalla* and *USPS*, and two different workloads: *uniform* or *own*. The workload tag "own" means the query set that is executed over the dataset follows the same distribution as the dataset, and hence is skewed too. This is realistic, as it would be expected that the more dense areas of the data set will be more often queried. The first part of the tag in the legend describes the dataset itself and the second part describes the type of workload that is used in the experiments. For example, USPS-Uniform is a case where a uniform randomly generated workload is executed over the USPS dataset. The executions of uniform randomly selected queries on Gowalla and USPS are also presented in Figure 2.5 and discussed above.

When the workload follows a dataset's distribution, the queries are generated randomly following this distribution. Independent of the underlying dataset and range size, PINED-RQ delivers high recall and precision when the workload follows the dataset's own distribution. In all cases, both recall and precision are very close to 100%. This is not the case with uniform-randomly generated workloads. From our observations, precision and recall are quite high when the query range covers the skewed area and most of the queries in the skewed workloads cover the skewed area. This results in very high recall and precision values. If the workload is uniform, the performance slightly depends on the queries.

2.4.3 Scalability

We use variants of the Gowalla dataset with a scaling factor of 0.5 million to perform the scalability test. Figure 2.7 shows the results with increasing dataset size. Each curve in the graph represents a different query range size. Although there is very small fluctuation in terms of recall and precision from 500 thousands to 1.5 million for small queries 1% and 5%, after 1.5 million records, the recall value does not fluctuate and achieves almost 100% for all



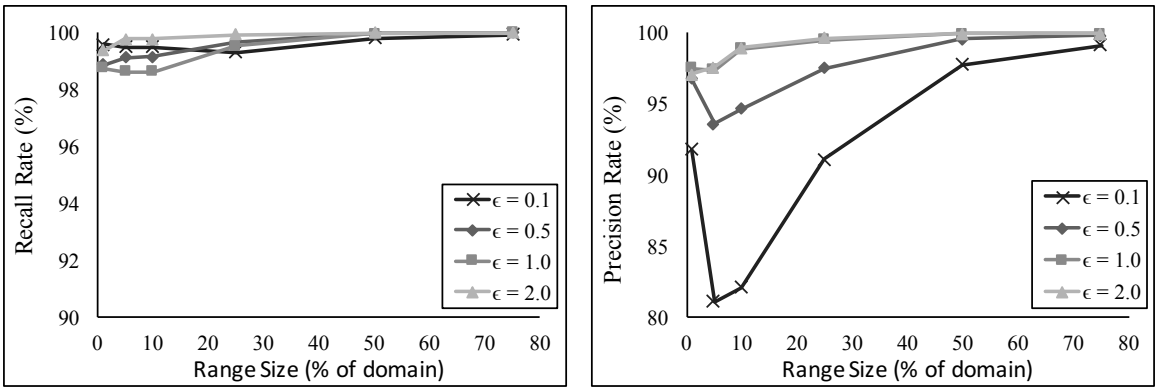
(a) Recall (b) Precision

Figure 2.7: Scalability

cases, which is quite significant. The larger dataset size means higher counts in the nodes. The magnitude of noise sampled from the Laplace mechanism is independent of the dataset size. Therefore, as the dataset size increases, the impact of the perturbation noise becomes negligible. This is a significant design advantage of PINED-RQ. The precision results are also less sensitive to the changes in dataset size. In all cases, PINED-RQ achieves $99 - 100\%$ precision, which is quite good from a system performance point of view. Our empirical findings show that PINED-RQ scales well as dataset size increases.

2.4.4 Effect of Epsilon

The effect of a privacy budget in differentially private publications has been studied by many prior works and it is known that smaller privacy budget provides less utility, since less budget causes higher noise sampling from the Laplace distribution. This hypothesis is also valid in our system and more observable in precision. Thanks to its design, PINED-RQ is capable of delivering high recall even if the privacy budget is small. Moreover, PINED-RQ is expected to have higher precision with larger privacy budgets. In this section, we explore this claim empirically and the results of our experiments, presented in Figure 2.8, verify the claims. In the earlier experimental sections, the privacy budget is set to 1 as explained before. When the



(a) Recall (b) Precision

Figure 2.8: Effect of privacy budget

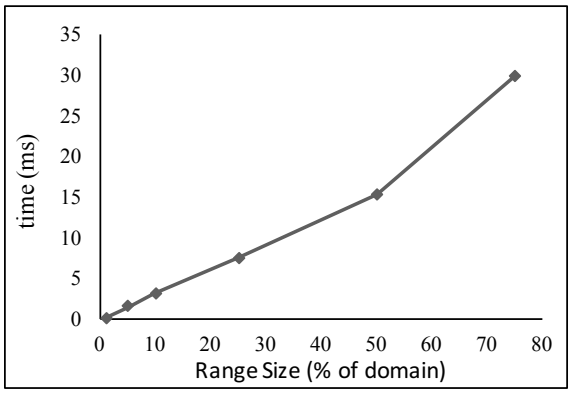


Figure 2.9: Index scan time

budget is dropped to a half or 0.1, the smaller ranges suffer in terms of precision (down to 80% for range queries of size 5% in the worst case for $\epsilon = 0.1$). The recall rate slightly increases as the range size increases, since the added noise is very small compared to the counts at the upper levels of the index. In the same case, the precision also increases. On the other hand, if the privacy budget is doubled, there is a slight improvement in the recall. Though, there is no significant improvement in terms of precision.

2.4.5 Index Scan Timing

PINED-RQ maintains its index in the clear and this ensures fast query processing times. Figure 2.9 shows the average index scan times per query over different range sizes. The scan times are in the order of milliseconds. As the range size increases, the index scan time also increases since the query processing algorithm has to consider more index nodes at the lower levels of the index. Compared to the related work discussed in Section 3.1 that perform heavy cryptographic computations during query executions, which result in query processing times in the order of tens of seconds even for smaller ranges, PINED-RQ is really fast in query processing. Even for the largest sized range 75%, PINED-RQ scans the index in 29.78 **ms**. The execution of similar sized range query over similar dataset takes slightly less than 10^3 **seconds** in [43] when the most secure *Logarithmic-SRC* index model is deployed. [43] guarantees 100% recall - note that precision can also be quite low, e.g., 50% for small to medium sized ranges. In contrast, PINED-RQ achieves approximately 100% recall in almost all cases and in the worst case 85% precision. However, PINED-RQ does not guarantee 100% recall. This is a reasonable performance trade-off given the orders of magnitude improvement in execution times.

2.4.6 Updates

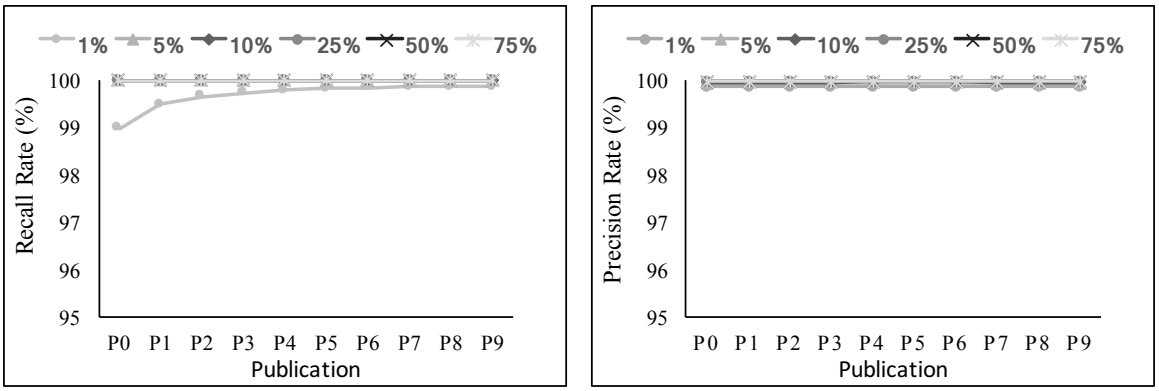
This section evaluates the performance of PINED-RQ's update management system. To simulate updates, we use an update-only uniform workload generator where 80% of the updates are new inserts, whereas the remaining 20% are modifications to earlier records. Due to the lack of space, we do not discuss the behavior of PINED-RQ for different update parameters. Here, we consider a specific scenario which fits well with the targeted application and evaluate PINED-RQ in terms of precision and recall. The results are presented in Figure 2.10. In this setting, we set ϵ_{total} to 1, the initial publication privacy budget ϵ_{init} to 0.7, the minimum publication budget ϵ_{min} to 0.3, the cost for storing data at the data provider α to 5, and the

update frequency parameter μ to 2. The initial publication uses a synthetic uniform dataset of size $500k$ records. The system is tested after each publication to the server. To focus on the performance of indexes, the reported recall and precision rates do not consider the answers returned from the data provider (the results returned from the data provider have 100% recall and precision, which will obviously increase the recall and precision of query answers). The x-axis of the graphs in Figure 2.10 represents the publication id, i.e., the initial publication is denoted by $P0$ and the next publication is denoted by $P1$. There are 10 publications where nine of them are publications of inserts while $P7$ is a special publication consisting solely of modifications to $P0$, with a size of $307,693$ records, on the initially published dataset.

The initial publication uses a budget of 0.7. This is also true for other publications except for $P7$, which uses a budget of 0.3. Thanks to the parallel composition of differential privacy, PINED-RQ maintains its privacy while publishing new datasets with the similar accuracy. Therefore, there is no significant effect on the performance of PINED-RQ while new datasets are published. The later publications only have a positive outcome since recall for the smallest range 1% starts from 99% with the initial publication and slightly increases with the further updates. On the other hand, the precision for the same range is constant over the publications and is not affected by further publications. For all other ranges, PINED-RQ delivers a decent performance with almost 100% recall and precision. Note that for each query, PINED-RQ scans all indexes in parallel, therefore, there is very small overhead in the index scan time compared to a single index case.

2.4.7 Small Datasets

PINED-RQ is devised for handling large datasets which is typical of data stored in the cloud, and our differentially private index structure benefits from this fact. On the other hand, we analyzed the case where very small datasets are outsourced to the cloud using PINED-



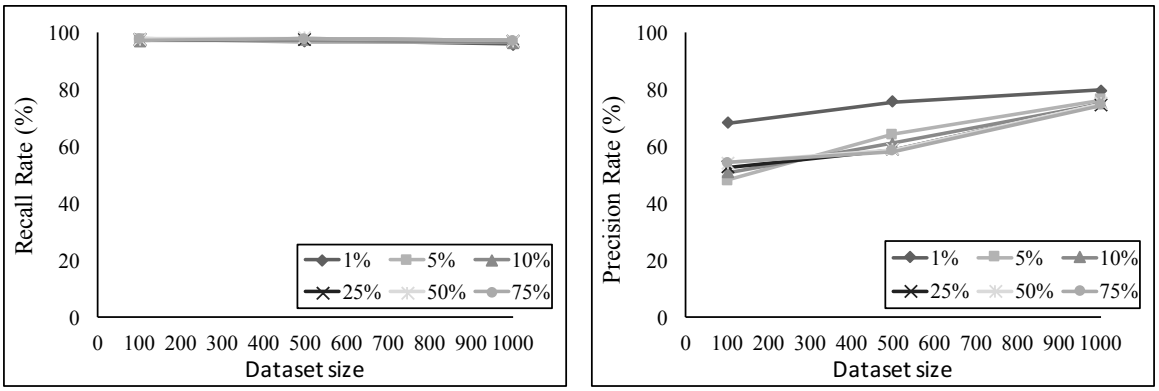
(a) Recall (b) Precision

Figure 2.10: PINED-RQ update performance

RQ. This set of experiments use synthetically generated uniform datasets with a size of 100, 500, and 1000 records. Figure 2.11 presents the experimental results in this setting. PINED-RQ maintains high recall even with small datasets thanks to the usage of overflow arrays. There is a slight decrease in recall compared to the earlier cases but this is expected given the probabilistic nature of the index creation. Note that the impact of noise at the lower levels is higher when the dataset size is small. The upper levels contain aggregate count information and they are less error-prone to the added noise. The effect of size is more visible in precision. Precision drops to 48% when the dataset size is 100 for query range 5%. However, precision increases to $75 - 80\%$ quickly when the dataset size increases from 100 to 100. This behavior is expected since as datasets include more records, the impact of noise on the nodes becomes negligible.

2.5 Related Work

The main challenge for privacy preserving querying has been range queries, so we will focus on them. There have been several research efforts focusing on improving the quality of privacy preserving histograms (for aggregate queries), *e.g.,* [48, 50, 51, 58]. For example, in [51], Qardaji et al. examine the factors affecting the accuracy of hierarchical approaches



(a) Recall (b) Precision

Figure 2.11: PINED-RQ with small datasets

when answering aggregate range queries. Their analysis and experimental results show that combining the choice of a good branching factor with constrained inference can significantly increase the accuracy of the aggregates. In our work, we take advantage of their results to improve the quality of the histograms used in the nodes of the differentially private index.

Bucketization has been used for answering range queries over outsourced data [6, 7, 59, 60]. For example, in [7], Hore et al. use this technique, and propose optimal solutions for distributing the encrypted data of a database to the buckets to guarantee good performance by reducing the number of false positives while preserving a high security level. This work and [59, 60] are complementary to ours; we can use their bucket optimizations for an optimal distribution of the encrypted data in the outsourced database, and then make the index out of it. Unfortunately, bucketization based approaches suffer from the lack of formal security guarantees.

Order preserving encryption (OPE) [2,8,41] has been also used for range query processing. In OPE, the order of the cipher text is the same as the order of the plaintext data. Modular order preserving encryption (MOPE) [41] adds a secret offset to the data before encryption to shift the ciphertext (in a ring) and to hide the real location of the encrypted data in their distribution. In [8], an improved version of MOPE has been proposed. It uses fake queries over the gap

between the maximum and minimum values to improve the security of MOPE against attacks that detect the max/min values. However, OPE, unlike PINED-RQ, reveals the underlying data distribution and is vulnerable to statistical attacks.

In [61], Eu-Jin Goh proposed a secure index that allows a user with a trapdoor for a data x to test if the database contains x or not. The index reveals no information about the data for which the user does not have the trapdoor. The index of [61] can be developed using pseudo-random functions and Bloom filters. The objective of our PINED-RQ index is different from that of [61]. Indeed, we target range queries instead of the exact match queries in [61], and our index allows to return a subset that with a probability matches a given range. In PINED-RQ, the certainty (*i.e.*, probability) about the membership of a data to small ranges is low, and this protects individual data from being revealed.

The recent works by Li et al. [42] and Demertzis et al. [43] rely on *Searchable Symmetric Encryption* (SSE) which has been mainly used for keyword search. Demertiz et al. address several subtle performance and security issues with the solutions proposed in [42] and propose a novel range query solution. The idea is to convert each possible range to a set of keywords, and attach to the keyword all the tuples it contains (*e.g.,* using an index). Then, the problem of range query evaluation is converted to the problem of keyword search. To take advantage of SSE, Demertzis et al. propose three types of indexing approaches with different space requirements in terms of domain size: quadratic, linear and logarithmic. The most secure approach is the quadratic approach. However, its space requirement is very high, *i.e.,* $O(n * m^2)$ where n is the database size and m the domain size. Even if the space requirement is improved, the proposed index scheme suffers from a a high number of false positives along with execution times in the hundreds of seconds, unlike PINED-RQ which has execution times in the hundreds of milliseconds. The objective in [42, 43] is to provide index indistinguishability for the structure and node values in the index. Intuitively, index indistinguishability means that for two datasets with the same size, one cannot distinguish which index has been generated

for which dataset. PINED-RQ differs from this work in its objectives, as we aim at providing index differential privacy.

Supporting updates in differentially private publications is a challenging task as discussed before. The continuous private statistics release with differential privacy has been explored in the context of data streams (e.g., traffic monitoring) [62, 63]. Such systems require repetitive computations and continuous observation. However, these systems have different characteristics than our system in terms of updates. Therefore, direct application of these approaches in our context is not reasonable. Zhang et al. [55] propose differentially private publications of set-valued data release for *count queries* with a limited number of incremental updates, which is the most relevant work to PINED-RQ in terms of updates. The privacy budget is allocated to each publication equally which requires a priori knowledge of total number of updates before system deployment. This work has two main distinctions from PINED-RQ regarding functionality and updates: 1) it is designed for count queries, 2) total number of updates are required upfront without considering any external factor, whereas PINED-RQ publishes updates in batches and the budget allocation and timing of the publication are functions of different factors.

Overall, to the best of our knowledge no previous work proposes an index that guarantees the differential privacy of indexed outsourced data with update support as in PINED-RQ. Joint use of encryption with differential privacy for processing range queries has allowed PINED-RQ to reconcile strong and formal privacy guarantees with efficient range query processing for large datasets.

2.6 Conclusion

PINED-RQ is a highly efficient and differentially private range query execution framework that constructs a novel differentially private index over an outsourced database. Unlike other

differentially private systems, PINED-RQ is extended to support update operations. To the best of our knowledge, PINED-RQ is the first work that builds, uses and maintains a differentially private index for performing *selection range queries*. We have demonstrated the security of PINED-RQ and shown empirically its practicality and efficiency through extensive experiments performed on synthetic and real datasets. Future work includes enlarging the family of indexes that follow similar principles.

Part II

Privacy Preserving Sustainability Metrics

Chapter 3

Privacy Preserving Certification

Organizations are often motivated to make public disclosures about their environmental performance. These motivations may be inspired by regulatory requirements, marketing initiatives, or as part of a broader project of corporate sustainability. The landscape of environmental and sustainability claims is largely standardized, as exemplified by the ISO 14000 series of standards. Often environmental disclosures take the form of certifications, which establish that some agency has reviewed the claim and confirmed its validity. A prominent example is the ISO 14001 certification, which simply establishes that a firm has an established policy to review and correct its environmental performance. When the objective is to make a quantitative evaluation about the ecological sustainability of a product or service, approaches that consider the full life cycle of the product are often used [39]. This form of analysis, known as life cycle assessment (LCA), is codified in the ISO 14044 standard [40].

Sustainability certification has been shown to lead to potentially significant operational improvements in environmental performance [64]. Firms with more significant environmental impacts are more likely to have high-quality environmental management systems [65]. Life cycle approaches can improve the quality of environmental disclosures [66] and also provide a framework for firms to take broader responsibility for the impacts of the products they make

or sell [67].

The ISO 14020 series of standards governs environmental product declarations (EPDs), which include public assertions about the sustainability of products, based on ISO 14044-style life cycle evaluation [68, 69]. EPDs can include both externally certified claims and self-reported results. Certified results can include both "pass-fail" binary assertions about a product or process with regard to a set of criteria, known as "eco-labels," as well as detailed quantitative results [70].

The data sets that provide input to these computations express essential information about the operation of a process or production step [71]. A typical data point could be the quantity of electricity required to output a reference unit of some product. These data are often regarded as confidential and are typically concealed through aggregation with other data sets [72, 73]. Engagement with stakeholders and supply chain partners [74] is often required for effective consideration of life cycle environmental sustainability, which accentuates confidentiality concerns and may limit the scope of information included in the assessment [66].

Despite the importance of data privacy, the LCA community lacks a formal framework for managing private data, and very limited number of techniques exist for computing sustainability metrics that preserve the privacy of input data. In [75], Kerschbaum et al. introduce a framework for sustainability benchmarking with the help of an untrusted third-party, however, the proposed solution has an assumption that the participants do not collude with the third-party or each other which not might be realistic in the LCA community. This can be a big risk to ensure the privacy of individual data since small organizations might be colluding with each other to gain private information against big competitors or vice versa. We seek to apply recent developments in secure multiparty computation (SMC) to the problem of certification of environmental claims even in the presence of colluding parties. Specifically, we aim to confront the following challenges: 1) mutually competitive firms want to gain private knowledge about their environmental performance by benchmarking their environmental impact against a statistical

measurement of its cohort, such as an average or maximum; 2) an association of firms wants to enable its members to make public, validated claims about their individual environmental performance in comparison to a cohort or to the full group, based on private data.

The first of these can be achieved using existing SMC protocols (see Section 3.1). However, SMC has never been applied to the case of sustainability assessment in a completely secure manner. The other use case is novel and has the distinct requirements that parties be provided with certificates validating qualitative assertions about their inputs without the inputs being known, and that parties *not* communicate directly with one another, instead interacting through a certifying authority.

Even if the computation is passed to a certifying authority, it cannot be assumed to act as a trusted, unbiased authority, since the parties may not want to reveal their individual inputs to any other entity, including the certifying authority. In general, the certifying authority might need to perform complex computations and comparisons. It might be possible to perform such computations with an untrusted authority using advanced cryptographic tools like fully homomorphic encryption [10], but such techniques are known to be quite inefficient [12]. An established, computationally efficient approach for performing the complex computations required for certification is to use secure co-processors [76]. A secure co-processor is a tamper-proof hardware, which provides a non-transparent and isolated computation environment. It creates a trusted computing environment in hostile environments and prevents any unauthorized access. Because of these advantages, secure co-processors have been adapted in different contexts such as encrypted database querying [77, 78] and secure multiparty computations [79]. However, such hardware is limited in terms of computational resources and their straightforward deployment does not solve all the problems. The design of a secure and efficient framework is still a challenge.

In this dissertation, we formally define the *privacy preserving certification* paradigm along with its goal, security and computation requirements. A certification is a quantitative eval-

uation of the result of such a computation, or an evaluation of a given contribution with respect to the result. We propose a novel privacy-preserving certification framework that enables an authorized party, referred to as *certifier*, to certify participants based on industrially well agreed on set of criteria or a common function without compromising any sensitive/confidential information to any other parties even in the presence of colluding parties. The framework does not require parties to communicate with each other and aims to minimize the rounds of communication between the parties and the certifier. We propose efficient algorithms to perform certification operations for the certification problems-mean, quantile- using the proposed framework. We show that the proposed algorithms are correct and secure with the assumption of semi-honest parties. Furthermore, we discuss the efficiency of our algorithms both empirically and analytically.

3.1 Related Work

Secure multiparty protocols (SMC) are known for computing functions jointly over a set of inputs without revealing any information about the inputs. In brief, a set of n parties with private inputs $x_1, x_2, \ldots, x_n$ wish to compute a function $f(x_1, x_2, \ldots, x_n)$ jointly without revealing any x_i to any other party. After an execution of this function, the parties learn the correct output but nothing else, even if some parties try to obtain more information by colluding. There are two-party computation protocols that execute generic functions [80, 81], but these constructions rely on heavy cryptographic computations and may not be practical [82]. Privacy-preserving statistics using SMC have been well-studied under the scope of privacy-preserving data mining [29, 83–86]. For example, Rmind [86] is a tool that computes well-known statistics privately such as average, mean, median, while [83] proposes a secure dot product computation using SMC.

Although SMC has a wide spectrum of applications, most applications require interactive

communication among the parties. In certification, this would require all parties to communicate for certification. Such interaction is not realistic nor desirable in the certification model, since the parties might not know each other, and may not want to communicate with each other. Certification on the other hand focuses on a performance evaluation using some statistical analysis. Our protocols differ from existing SMC approaches in that they do not require communication and data exchange among the parties, and instead require the involvement of an authorized (but untrusted) party in the computations to regulate certification policies.

Involvement of an authorized party requires an establishment of a trust between the participants and the authority. Establishing trust on an untrusted party is not a new problem in the literature and several works in different contexts [77, 78, 87–89] rely on trusted hardware based solutions, e.g. Trusted Platform Modules (TPMs) [90] or secure co-processors [76, 91], to establish a trusted computing environment, which are shown to be quite efficient for specific applications [77, 89].

Unlike fully homomorphic encryption, which is computationally quite expensive, partial homomorphc encrytpion has been shown to be relatively efficient. Examples of partial homomorphic encryption are the additive homomorphic Paillier [92] and Quadratic Residues [93] public key cryptosystems and these will be explained in detail later in Section 3.3.2. The central component of our protocols is private comparison, which has been well studied previously [80, 94–99]. Each technique is suitable to different settings. For example, while [98] performs comparison on encrypted data, [96] compares unencrypted values privately. It is important to note that providing a new private comparison technique is not in the scope of this work, it is just one of the main building tools to develop our protocols for the certification problem. We adapted our private comparison protocol from Veugen's protocol [98] as discussed in Section 3.3.3. The recent works [78, 82] also adopt Veugen's protocol to solve different problems. Bost et al. [82] construct machine learning classification protocols over encrypted data. On the other hand, Baldimtsi et al. [78] propose a framework, which also benefits from secure

co-processors, that builds on top of searchable encryption techniques to return ranked results to queries. Our work follows in this tradition, and applies it to an important new domain, namely environmental certification.

To the best of our knowledge, the only closest work to ours is [75]. In this work, Kerschbaum et al. propose a private benchmarking platform for environmental sustainability with the help of an untrusted third party. Although the overall setting seems similar to our setting, there are fundamental differences in two approaches regarding the security of the systems. The assumption in [75] is that the parties do not collude with each other and the untrusted party. However, this is not a realistic assumption given the current competition in the market. The parties might collude with each other or with the untrusted party to gain private knowledge against the competitors. The proposed key management scheme in [75] either allows parties to share the same private key or distribute the private key among k parties which will later require at least t of them to be present to decrypt the output. In the case of key sharing, any party colluding with the untrusted party can reveal the private inputs of other parties. Similarly, in the presence of t colluding parties, it is possible to infer the private inputs of others if the key distribution approach is applied. Our approach is secure against colluding parties. Additionally, the certification process heavily relies on private comparison of inputs. The proposed comparison protocol in [75] relies on [94] which ensures a weaker notion of security due to the usage of multiplicative hiding. Our protocols rely on semantically and cryptographically secure comparison protocols in the certification process.

3.2 Problem Description

3.2.1 Privacy-Preserving Aggregation in LCA

Life Cycle Assessment (LCA) is critical for quantitative evaluations of the ecological sustainability of a product or service. The computation of results in LCA can be described as a series of matrix operations in which possible results are activity or output levels of industrial unit processes, quantities of emissions into the environment resulting from those processes, or measurements of environmental impact scores [100]. The calculation of any one of these values can be described as the inner product of a vector of input data with a weighting vector of environmental characteristics [101]. We formulate the private LCA aggregation problem as an inner product of two vectors:

$$s = \mathbf{w} \cdot \mathbf{x} \tag{3.1}$$

where s is an LCA metric, each element x_i of the input vector $\mathbf{x}$ is one party's private contribution, and the weighting vector $\mathbf{w}$ is determined separately and may be either public or private. In this dissertation, for simplicity, $\mathbf{w}$ will be taken to be $\mathbf{1}$, so that s is the sum of the parties' inputs.

Consider an international trade group in steel manufacturing that wants to issue a report that documents the industry's environmental performance, such as the World Steel Organization's LCA study [102]. Conventionally, such a report can only be prepared if the member firms share their confidential information with the trade group, allowing it to perform the aggregation and report the results. If instead the report were determined using privacy-preserving aggregation, the inputs would remain private, and firms could use the results privately for benchmarking their own performance, or publish the results, individually or together. However, the veracity of the results would be difficult to establish to the public.

We define a new problem, called *private certification*, in which an authorized party, referred

to as *certifier*, can certify the participants' inputs based on a set of criteria or a common function without compromising any sensitive or confidential information. The output of this private computation may be announced by the certifier publicly or held private; however, the certifier cannot learn any sensitive information during its execution. The certifier would need to be "trusted" by the public to compute and report results accurately, but may not be trusted by the parties with respect to the private data. In the private certification framework, unlike in traditional SMC, parties *are not required to communicate with each other*, but only with the certifier.

We introduce two new privacy preserving certification problems, namely *mean* and *quantile* based, which allow firms to make public or private announcements about their inputs to a secure aggregation. Here we describe the constraints and requirements of the two certification methods. The correctness of the certification relies on the correctness of the inputs. As we mentioned earlier, the parties are semi-honest, i.e. they are honest about executing the protocol correctly, but curious to learn other inputs. Hence, we can assume that the provided inputs are correct, which is a standard assumption in the LCA context, since the correctness of inputs are verified via an audit after the computation (e.g. [103, 104]). Please note that in describing the functionality, we use inputs in the clear and ignore cryptographic details. Later in Section 3.4, we will explain how to perform these certifications securely.

3.2.2 Mean Based Certification

In *mean based certification*, the certifier uses private aggregation to compute the average of a set of private inputs. Afterwards, the certifier compares each private input x_i with the average and performs the necessary certification operation, i.e. if a party generates less than the average, it can seek being labeled as more "eco-friendly" than its peers; otherwise it can forgo such labeling.

In mean based certification, the certifier computes the average of n inputs $x_1, x_2, x_3, ..., x_n$, and then certifies the parties either as *below* or *above* by comparing the individual values with the computed average value.

3.2.3 k-Quantile Based Certification

Grouping items into distinct groups based on predefined criteria is a well studied concept in statistics and can be utilized in different contexts. In the context of environmental impact assessment, this grouping technique provides performance information about a specific firm among the set of manufacturers. Being in the top quantile may be regarded as a prestigious certification that manufacturers can use to advertise their products with a greater confidence. By the nature of quantile based computation, the order information among the groups is revealed but it is hard to conclude which party is better inside the same group if the complete ranking information is hidden. It also allows parties increased flexibility to publish top performers' results while keeping others private.

In $k - quantile$ based certification, the certifier partitions the parties into k groups after ranking them based on the provided inputs. A party with the minimum input will be in the first group while a party with the maximum input will be in the k^{th} group.

3.3 System Model and Building Blocks

We now describe the system model and basic building blocks used in this work. We start by explaining the system model and then discuss the cryptosystems and the protocols for computing private comparison over encrypted data.

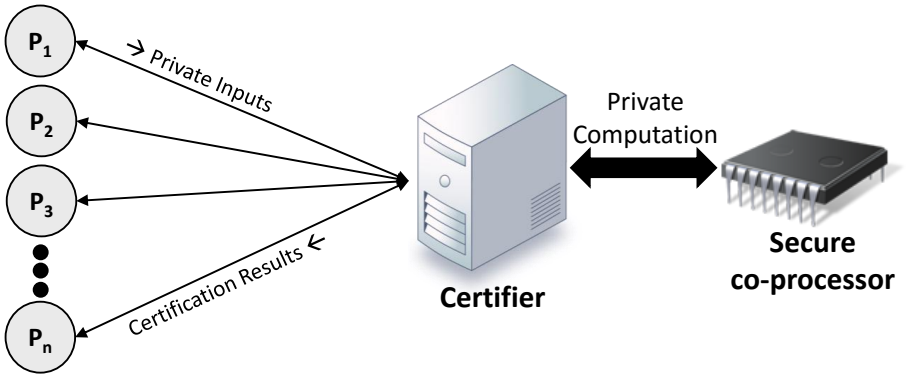


Figure 3.1: Overview of Framework Model

3.3.1 System Model

The proposed framework contains three main entities: *parties*, a *certifier*, and a *computation helper* as illustrated in Figure 3.1.

Parties. Parties are end-users which are the main data (input) providers to the system. In reality, parties are the competitors in manufacturing the same product or providing the same service. To demonstrate the superiority of their product or service, they would like to be certified by an authorized party. Parties are not aware of the other participant parties and do not communicate directly with each other.

Certifier. In this context, the authorized party is called the *certifier*. It is the main computation unit of the framework and it communicates with all registered parties during the computation. Each party has to register through the certifier to be able to join the certification process. The certifier is trusted in performing operations but at the same time it might be curious to learn some information about the parties' data. Therefore, the framework aims to preserve the confidentiality of inputs throughout the computation against the certifier and all other external adversaries. To achieve this goal, the computation is split between two non-colluding computation units: *the certifier itself and a computation helper*.

Computation Helper. The comparison functions can be implemented in other ways using other SMC based solutions, but the proposed comparison protocols, which are explained in Section 3.3.3 in detail, rely on two computation units to satisfy all necessary security and efficiency concerns. Therefore, the certifier needs an additional computation unit to satisfy the constraints. It is called *computation helper*.

The computation helper aids the certifier compute the certification function. The helper and the certifier must not collude, otherwise, they can reveal the secret data. The helper can be a server from a different service provider or a secure, tamper-proof hardware that can be deployed on the certifier site. As depicted in Figure 3.1, the framework deploys a specialized secure co-processor like IBM 4764 PCI-X Cryptographic co-processor [76]. These processors have relatively low resources in terms of memory and computation power, and are invoked to compute relatively small computations. Secure co-processors provide a non-transparent and isolated computation environment which fits directly into our model. We assume that the supplier of the co-processor is different than the certifier and their marketing interests do not intersect. Several privacy preserving solutions using a secure co-processor have already been proposed in different contexts such as encrypted database querying [77, 78] and secure multi-party computations [79]. Our framework requires only one round of communication between the parties and the certifier. Once a party submits a private input to the certifier, all the remaining communication happens between the certifier and the secure co-processor (the computation helper). The availability of a fast network communication between the certifier and the secure co-processor is another advantage of our design. When the secure co-processor is deployed at the certifier's site, it is realistic to assume negligible network latency, since communication usually happens in the order of 1 millisecond.

3.3.2 Cryptosystems

The certifier needs two additively homomorphic cryptosystems: Paillier [92] and Quadratic Residues(QR) [93]. The cryptosystem is called *partially homomorphic* if it supports either addition (additive homomorphic) or multiplication (multiplicative homomorphic). These approaches are considered efficient compared to the fully homomorphic encryption. Both Paillier and QR are additively homomorphic which means given two encrypted ciphertexts, $Enc(m_1)$ and $Enc(m_2)$, the application of the additive homomorphic operation will result in the decryption of $Enc(m_1 + m_2)$.

The Paillier cryptosystem is based on the Decisional Composite Residuosity assumption [92]. We use $[\![m]\!]$ to denote the encryption of message m with the Paillier cryptosystem using a public-secret key pair $K_P = (PK_P, SK_P)$. The plaintext space of Paillier is $\mathbb{Z}_N$ where N is the public modulus of Paillier and its homomorphic property is $[\![m_1]\!].[\![m_2]\!]$ = $[\![m_1 + m_2]\!]$. In addition, the Paillier cryptosystem also supports multiplying ciphertext with a constant, which is actually the homomorphic summation of input with itself by n times. On the other hand, the plaintext space of Quadratic Residues (QR) is bits and $[m]$ denotes the encrypted bit m under QR. The key pair of QR is denoted by $K_{QR} = (PK_{QR}, SK_{QR})$. The homomorphic property of QR is $[m_1].[m_2]= [m_1 \oplus m_2]$.

Basically, Paillier implements the following three functions:

- $K_P(PK_P, SK_P) \leftarrow KEYGEN_{PL}(\lambda)$ generates a key pair. Note that λ is a security parameter.
- $[\![m]\!] \leftarrow$ encPL(m, PK_P) encrypts plaintext m using public key PK_P and outputs encrypted ciphertext $[\![m]\!]$.
- $m \leftarrow$ decPL($[\![m]\!]$, SK_P) decrypts given ciphertext $[\![m]\!]$ using secret key SK_P and outputs m in the clear.

Similarly, QR implements the following functions:

- $K_{QR}(PK_{QR}, SK_{QR}) \leftarrow KEYGEN_{QR}(\lambda)$ generates a key pair.
- $[m] \leftarrow$ encQR(m, PK_{QR}) encrypts clear bit m using public key PK_{QR} and outputs encrypted ciphertext $[m]$.
- $m \leftarrow$ decQR($[m]$, SK_{QR}) decrypts given ciphertext $[m]$ using secret key SK_{QR} and outputs m in the clear.

For the simplicity, we will omit including keys and security parameters in the function parameters in the rest of the chapter.

3.3.3 Comparison of Encrypted Data

A primitive module used by many of the problems addressed in this work is "comparison". Take mean certification as an example. The certifier is able to compute the average using the homomorphic encryption scheme. However, the next step is challenging: the certifier has to compare secret values against the average without learning any information about neither the average nor secret values. There is no efficient and secure way for a certifier to perform the comparison herself. Therefore, we need a *collaboration of two parties* such that both will not know the values, but *together* they will be able to do the comparison. The proposed framework fits this requirement and the certifier is able to perform the comparison protocol with the help of a computation helper.

The certifier has two encrypted numbers $[\![a]\!] \leftarrow$encPL(a) and $[\![b]\!] \leftarrow$encPL(b) of ℓ bits and the computation helper has private keys SK_P and SK_{QR}. Both $[\![a]\!]$ and $[\![b]\!]$ are sent by parties. The goal of the comparison protocol is to decide whether $a \leq b$ without revealing the actual values of a and b to neither the certifier nor the computation helper. Our comparison protocol is adapted from Veugen's [98] protocol. The main idea is to compute $2^{\ell} + b - a$ and check

the most significant bit ($\ell + 1$). If the most significant bit equals 1, then $a \leq b$, otherwise $a > b$. As a result of the protocol, the certifier gets the result of the comparison encrypted and the computation helper never learns the actual results of the inputs. Veugen's protocol has also been adapted and slightly modified by two recent works [78, 82].

To perform certification either with public or private outputs in our certification framework, we introduce two private comparison protocols, namely PRIVATECOMPARE and ENCRYPTEDPCOMPARE. They both takes encrypted inputs but PRIVATECOMPARE announces the output of the comparison publicly, while ENCRYPTEDPCOMPARE keeps the result of the comparison secret. Both protocols require joint computations between the two parties, and both of them are secure under the semi-honest security model.

PRIVATECOMPARE compares two encrypted inputs and announce the result of the comparison publicly. The details of the protocol is summarized in Protocol 1. It is a joint computation of two parties, the certifier and the computation helper. The certifier has two encrypted numbers $[\![a]\!]$ and $[\![b]\!]$ and owns public keys PK_P, PK_{QR} and secret key SK_{QR}. On the other hand, the computation helper owns the secret key for Paillier, SK_P. The certifier initially computes $[\![x]\!] \leftarrow [\![b]\!].[\![2]\!]^{\ell}.[\![a]\!]^{-1}\ mod\ N$ and then hides it with a randomly chosen number, r. r should contain σ more bits than x. Next, the certifier sends $[\![z]\!]$ to the computation helper. Note that unless x was hidden by r, the computation helper could easily learn the comparison result. After receiving $[\![z]\!]$, the computation helper decrypts it and computes $d \leftarrow z\ mod\ 2^{\ell}$. In the meantime, the certifier computes $c \leftarrow r\ mod\ 2^{\ell}$. Then, the certifier and the computation helper cooperate to compare c and d ($t' \equiv d < c$) using a private input comparison protocol. Although Veugen also proposes a private integer comparison protocol in [98], Bost et al. [82] suggest using the DGK protocol [96] for better practicality. This private integer comparison procedure is a sub-procedure in the protocol and either of the proposed protocols can be used in this protocol. After the execution of the private input comparison, the computation helper receives the encrypted bit $[t']$ as a result. Later, the certifier encrypts and sends the $(\ell + 1)^{th}$

Protocol 1 Two party private comparison with Public Output

Input A: $[\![a]\!]$, $[\![b]\!]$, PK_P, PK_{QR}, and SK_{QR}
Input B: SK_P
Output: bit t where $t = a \leq b$

1: **procedure** PRIVATECOMPARE($[\![a]\!]$, $[\![b]\!]$)
2: A: $[\![x]\!] \leftarrow [\![b]\!].[\![2^\ell]\!].[\![a]\!]^{-1}\ mod\ N$ ▷ $x \leftarrow b + 2^\ell - a$
3: A chooses a random number $r \leftarrow \{0,1\}^{\ell+\sigma}$
4: A: $[\![z]\!] \leftarrow [\![x]\!].[\![r]\!]\ mod\ N$
5: A sends $[\![z]\!]$ to B
6: B: $z \leftarrow$ decPL($[\![z]\!]$)
7: A: $c \leftarrow r\ mod\ 2^\ell$
8: B: $d \leftarrow z\ mod\ 2^\ell$
9: A and B privately compute the encrypted bit $[t']$ such that $t' = (d < c)$
10: A: $[r_{\ell+1}] \leftarrow$ encQR($r_{\ell+1}$) and sends $[r_{\ell+1}]$ to B
11: B: $[z_{\ell+1}] \leftarrow$ encQR($z_{\ell+1}$)
12: B: $[t] \leftarrow [z_{\ell+1}].[r_{\ell+1}].[t']$ ▷ $t \leftarrow z_{\ell+1} \oplus r_{\ell+1} \oplus t'$
13: B sends $[t]$ to A
14: A: $t \leftarrow$ decryptQR(t)
15: **return** t

bit of r, $[r_{\ell+1}]$ to the computation helper. Finally, the computation helper computes the most significant bit of z by computing $[t] \leftarrow [z_{\ell+1}].[r_{\ell+1}].[t']$ and sends $[t]$ to the certifier. By using private key SK_{QR}, the certifier decrypts $[t]$ and announces t publicly.

Unlike PRIVATECOMPARE, ENCRYPTEDPCOMPARE aims to return both the comparison result and its negation privately. ENCRYPTEDPCOMPARE is summarized in Protocol 2. As in PRIVATECOMPARE, ENCRYPTEDPCOMPARE also requires the cooperation of both the certification and the computation helper. Although the protocols appear quite similar, they feature crucial differences in terms of the initial setup and the computation. The certifier owns two encrypted numbers-$[\![a]\!]$, $[\![b]\!]$- and public keys for both Paillier and QR cryptosystem, PK_P and PK_{QR}. On the other hand, the computation helper owns private keys for both Paillier and QR, SK_P and SK_{QR}. Until the private integer comparison, both the certifier and the computation helper follow the same procedures as they execute in Protocol 1 (line 2 to 9). Once line 9 is executed, i.e. the certifier and the computation helper have privately

Protocol 2 Two party private comparison with Private Output

Input A: $[\![a]\!]$, $[\![b]\!]$, PK_P, and PK_{QR}
Input B: SK_P and SK_{QR}
Output A: Encrypted Integer $[\![t]\!]$ where $(t = 1) \equiv a \leq b$

9: **procedure** ENCRYPTEDPCOMPARE($[\![a]\!]$, $[\![b]\!]$)
Run the steps 2-9 of Protocol 1
10: A: $[r_{\ell+1}] \leftarrow$ encQR($r_{\ell+1}$)
11: B: $[z_{\ell+1}] \leftarrow$ encQR($z_{\ell+1}$) and sends $[z_{\ell+1}]$ to A
12: A: $[t] \leftarrow [z_{\ell+1}].[r_{\ell+1}].[t']$ $\triangleright\ t \leftarrow z_{\ell+1} \oplus r_{\ell+1} \oplus t'$
Run re-encryption procedure
13: $[\![t]\!]$, $[\![\bar{t}]\!] \leftarrow$ REENCFORPL($[t]$) from Protocol 3

computed the encrypted bit $[t']$ such that $(t' = 1) \equiv (d < c)$, the certifier receives the result of the comparison encrypted $[t']$, and computes $[r_{\ell+1}]$. In the meantime, the computation helper encrypts $[z_{\ell+1}]$ and sends it to the certifier. Finally, the certifier computes $[t] \leftarrow [z_{\ell+1}].[r_{\ell+1}].[t']$ and has the result encrypted. The result is encrypted with the QR cryptosystem. Thus, the certifier and the computation helper jointly run the re-encryption protocol that returns both the resulting bit and its negate to the certifier encrypted under Paillier, i.e. $(t = 1 \equiv a \leq b) \iff$ $[\![t]\!]$=$[\![1]\!]$ and $[\![\bar{t}]\!]$=$[\![0]\!]$.

3.3.4 Re-encryption From QR to Paillier

PRIVATECOMPARE generates the result of the comparison encrypted under the QR cryptosystem (line 12 of Protocol 1). The plaintext space of QR is a bit, i.e. the result is either the encryption of 0 or 1. Although it is enough for learning the result of the comparison, to rank the inputs privately, our quantile based certification protocol needs to keep counters for comparison results without actually knowing the result. Therefore, we need to re-encrypt the resulting comparison bit to a corresponding integer value which is encrypted with Paillier. Re-encryption from the QR scheme to Paillier is performed such that the value of an encrypted bit is not revealed to any of the parties. Our implementation is adapted from [78] and slightly modified to meet the additional requirements. As presented in Protocol 3, to re-encrypt en-

crypted bit $[m]$, the certifier selects a random secret bit r, and then computes $[s_r]$=$[m].[0]$ and $[s_{1-r}]$=$[m].[1]$. The certifier sends $[s_r]$ and $[s_{1-r}]$ to the computation helper, thus, independently of the value of m, the computation helper receives the encryption of 0 and 1 every time. Then, the computation helper decrypts s_r and s_{1-r} under Paillier encryption, and sends $[\![s_r]\!]$ and $[\![s_{1-r}]\!]$ back together with their negates $[\![\overline{s_r}]\!]$, $[\![\overline{s_{1-r}}]\!]$ to the certifier in the same order as it received them. Since the certifier knows r, it uses $[\![s_r]\!]$ and $[\![\overline{s_r}]\!]$. $[\![\overline{s_{1-r}}]\!]$ and $[\![\overline{s_{1-r}}]\!]$ are disregarded.

Protocol 3 Re-encrypt from QR to Paillier

Input A: $[m]$, PK_P, and PK_{QR}
Input B: SK_P and SK_{QR}
Output: $[\![m]\!]$ where $[\![m]\!]$ = $[\![1]\!]$ if $m \equiv 1$. Else, $[\![m]\!]$ = $[\![0]\!]$.

1: **procedure** REENCFORPL($[m]$)
2: A chooses a random bit $r \leftarrow \{0, 1\}$
3: A: $[s_r] \leftarrow [m].[0]$ ▷ $s_r \leftarrow m \oplus 0$
4: A: $[s_{1-r}] \leftarrow [m].[1]$ ▷ $s_{1-r} \leftarrow m \oplus 1$
5: A sends $[s_0]$ and $[s_1]$ to B
6: B: $s_0 \leftarrow$ decrQR($[s_0]$)
7: B: $[\![s_0]\!] \leftarrow$ encPL(s_0), $[\![\overline{s_0}]\!] \leftarrow$ encPL($s_0 \oplus 1$)
8: B: $s_1 \leftarrow$ decQR($[s_1]$)
9: B: $[\![s_1]\!] \leftarrow$ encPL(s_1), $[\![\overline{s_1}]\!] \leftarrow$ encPL($s_1 \oplus 1$)
10: B sends $[\![s_0]\!]$, $[\![s_1]\!]$, and their negates to A in the same order as received, i.e ($[\![s_0]\!]$, $[\![s_1]\!]$, $[\![\overline{s_0}]\!]$, $[\![\overline{s_1}]\!]$)
11: A: $[\![m]\!] \leftarrow [\![s_r]\!]$ and $[\![\overline{m}]\!] \leftarrow [\![\overline{s_r}]\!]$

Note that our comparison and re-encryption protocols are correct and secure. The correctness and the security can be found in [78, 98].

3.4 Certification Protocols

This section outlines how to deploy and perform the certification operations described in Section 3.2 in a privacy-preserving manner on top of the proposed framework model. Basically, n parties want to be certified through a certifier. To satisfy security guarantees, the computation

helper, an on-site secure co-processor, helps the certifier execute protocols securely. Note that each certification problem has its own computation and security requirements, and these are highlighted explicitly. For simplicity, we assume all n parties join the computation.

3.4.1 Private Mean Based Certification

To perform *mean based certification*, the certifier needs to overcome two main challenges: (1) computing the average of n encrypted ciphertexts, (2) comparing each private input with the computed average privately.

Initialization. The secure co-processor executes the $K_P \leftarrow KEYGEN_P$ function to generate a key pair for the Paillier cryptosystem and shares public key PK_P with the certifier. Then, the certifier executes the key generation algorithm for QR, $K_{QR} \leftarrow KEYGEN_{QR}$. After key generation, the certifier sends PK_P to all parties and sends PK_{QR} to the secure co-processor.

Security Requirements. The individual inputs x_i will be kept confidential throughout the certification. In addition to this, the average value of the provided inputs must also be hidden from both the certifier and the secure co-processor. The final result of the computation will be made public. The system should also be secure against the existence of colluding parties.

Protocol. Parties encrypt their inputs with the Paillier cryptosystem using public key PK_P, $[\![x_i]\!]\leftarrow$encPL(x_i), and send encrypted ciptertexts to the certifier. After receiving n inputs $[\![x_1]\!]$, $[\![x_2]\!]$, ..., $[\![x_n]\!]$, the certifier executes the MEAN-CERTIFY algorithm which is presented in Protocol 4. The protocol starts by computing the summation of the private inputs. Using the homomorphic property of Paillier, the certifier computes $[\![s]\!] \leftarrow [\![s]\!].X[i] \ mod \ N$. This operation yields $s \leftarrow s + x_i$ and after this is executed on all inputs, the resulting computation will be the summation of all inputs encrypted with Paillier, $[\![s]\!]$. The Paillier cryptosystem does not support a division operation. Rather than computing the average, i.e. dividing the

Protocol 4 Mean Certification

Input Party$_i$: x_i and PK_P
Input Certifier: SK_{QR}, and PK_P
Input Secure Coprocessor: SK_P and PK_{QR}
Output: c_i (certification result for each party)

1: **procedure** SENDTOCERTIFIER(x_i)
2: $[\![x_i]\!] \leftarrow$ encPL(x_i)
3: Send $[\![x_i]\!]$ to the certifier
4: Each party executes SENDTOCERTIFIER(x_i)

After receiving all inputs, $X[1..n] = \{[\![x_1]\!], [\![x_2]\!],...,[\![x_n]\!]\}$, the certifier executes the following procedure.

5: **procedure** MEAN-CERTIFY($X[1..n]$)
Compute the sum of inputs
6: $[\![s]\!] \leftarrow X[1]$
7: **for** $i \leftarrow 2$ to n **do**
8: $[\![s]\!] \leftarrow [\![s]\!].X[i]\ mod\ N$ ▷ $s \leftarrow s + x_i$
Note that $s = \sum_{i=1}^{n} x_i$
9: **for** $i \leftarrow 1$ to n **do**
10: $[\![\tilde{x_i}]\!] \leftarrow X[i]^n\ mod\ N$ ▷ $\tilde{x_i} \leftarrow n \times x_i$
11: $t_i \leftarrow$ PRIVATECOMPARE($[\![s]\!]$, $[\![\tilde{x_i}]\!]$)
12: **if** $t_i == 1$ **then**
13: $c_i \leftarrow$ Above
14: **else**
15: $c_i \leftarrow$ Below
16: Sends c_i to P_i

summation by n, the certifier normalizes the inputs by multiplying them by the number of participants, i.e. $[\![\tilde{x_i}]\!] \leftarrow [\![x_i]\!]^n \ mod \ N$. Recall that our main goal is to compare x_i with the average, ie, $x_i \leq sum/n$. The basic idea for this comparison is $x_i \leq \frac{sum}{n} \equiv x_i * n \leq sum$ where $\frac{sum}{n}$ is the average. Recall that the Paillier cryptosystem supports multiplying ciphertext with a constant. After normalizing the input, the certifier and the secure co-processor jointly execute the PRIVATECOMPARE function, introduced in Section 3.3.3, to compare $[\![s]\!]$ with the normalized input $[\![\tilde{x_i}]\!]$. The result of this comparison is known by the certifier in the clear and the certification is completed by labeling the party with input x_i as *above* or *below*.

Correctness. The certifier computes the summation of n private inputs using the additive homomorphic operation of Paillier which executes the summation operation over ciphertexts. Because of the homomorphic property of Paillier, lines 3 through 5 of Protocol 4 compute the encrypted summation of n inputs. Each encrypted input $[\![x_i]\!]$ is normalized by taking the power of n under modular arithmetic, which is equivalent to $[\![\tilde{x_i}]\!] \leftarrow [\![x_i \times n]\!]$ due to the Paillier properties. The comparison of each private input with the average of n private input is equal to the comparison of normalized input with the summation of n inputs, i.e. $\frac{sum}{n} \leq x_i \equiv sum \leq x_i \times n$ where $sum \leftarrow \sum_{i=1}^{n} x_i$. After executing the private comparison, a party is certified as *above* if $sum \leq x_i * n$. Otherwise, the label is *below*.

Intuition of Security Proof. The certifier receives the inputs encrypted with Paillier from the parties. Since, it does not own the secret key SK_P, it cannot decrypt and learn the actual inputs. The homomorphic addition is semantically secure due to the Paillier cryptosystem, and the certifier computes the summation encrypted under Paillier. The comparison protocol is already proved secure [98] and does not reveal any information. Recall that the only restriction on collusion is between the certifier and the helper. Hence, we need to prove that collusion between the certifier and any number of parties will not reveal any private parties. Assume $n-1$ parties collude with the certifier except party P_1. In such a case, the certifier has $x_2, x_3, ..., x_n$ in the clear and x_1 encrypted, i.e. $[\![x_1]\!]$. Throughout the computation, the certifier computes sum

encrypted which is denoted as $[\![s]\!]$ in Protocol 4. The certifier can compute $C_{sum} = \sum_{i=2}^{n} x_i$ in the clear. If sum was in clear, knowing C_{sum} would help computing $s - C_{sum} \equiv x_1$. However, since s is encrypted, the subtraction results in $[\![s]\!].[\![C_{sum}]\!]^{-1} \equiv [\![s - C_{sum}]\!] \equiv [\![x_1]\!]$. As it can be easily inferred from the result, the colluding parties do not provide any useful information to the certifier to reveal x_1. Hence, our private mean based certification protocol is secure even under the existence of colluding parties, since neither the certifier nor the secure co-processor learn any intermediary results throughout the computation.

3.4.2 Private k-Quantile Certification

To split parties into distinct groups privately, the quantile based certification is performed. Although the quantile computation is directly related to the ranking of a set of inputs, the certifier computes the k-quantiles privately without learning the ordering of the private inputs.

Initialization. The secure co-processor generates key pairs, K_P and K_{QR}, for both Paillier and QR. It owns *both* private keys SK_P and SK_{QR}, and sends the public keys, PK_P and PK_{QR}, to the certifier. Then, the certifier shares the public key for Paillier, PK_P, with the parties. We assume that the certifier knows the parameter k.

Security Requirements. Throughout the certification process, the individual inputs should be kept secret as in prior certifications. By the nature of quantile computations, the order information among different groups, i.e. the order of the parties in different groups, will be revealed. However, the ordering information of parties inside the same quantile group should not be revealed. We assume that the parties do not collude in this certification method[1].

Protocol. Parties encrypt their inputs with Paillier, $[\![x_i]\!]$, and send them to the certifier. After receiving n inputs, the certifier executes the QUANTILE-CERTIFY algorithm which

[1]This is a natural problem of quantile based private grouping. If the parties from neighbor groups collude with each other, due to the ordering, it might be possible to reveal the input of non-colluding party in one of these groups.

Protocol 5 k-Quantile Certification

Input Party$_i$: x_i and PK_P
Input Certifier: k, PK_P, and PK_{QR}
Input Secure Coprocessor: SK_{QR}, SK_P and PK_{QR}
Output: c_i (certification result for each party)

Each party executes SENDTOCERTIFIER(x_i) from Protocol 4
After receiving all inputs, $X[1..n] = \{[\![x_1]\!],...,[\![x_n]\!]\}$, the certifier executes the following.
procedure QUANTILE-CERTIFY($X[1..n]$, k)
Private pairwise comparisons of inputs
    $C[1..n][1..n] \leftarrow$empty
    for $i \leftarrow 1$ to n **do**
        for $j \leftarrow i+1$ to n **do**
            $[t_{ij}] \leftarrow$ ENCRYPTEDPCOMPARE($X[i]$, $X[j]$) ▷ $t_{ij} \leftarrow x_i \leq x_j$
            $[\![t_{ij}]\!]$, $[\![\overline{t_{ij}}]\!] \leftarrow$ REENCFORPL($[t_{ij}]$)
            $C[i][j] \leftarrow [\![t_{ij}]\!]$ and $C[j][i] \leftarrow [\![\overline{t_{ij}}]\!]$
    $c[1..n] \leftarrow$ empty
    for $j \leftarrow 1$ to n **do**
        $[\![sum]\!] \leftarrow [\![0]\!]$
        for $i \leftarrow 1$ to n **do**
            if $i \neq j$ **then**
                $[\![sum]\!] \leftarrow [\![sum]\!].C[i][j]\ mod\ N$
        $c[j] \leftarrow [\![sum]\!]$
    Choose a random permutation π over $\{1, .., n\}$
    for $i \leftarrow 1$ to n **do**
        $c_\pi[i] \leftarrow c[\pi(i)]$
    $R[1..n] \leftarrow$COMPUTEBIN($c_\pi[]$, n, k)
    for $i \leftarrow 1$ to n **do**
        $j \leftarrow \pi^{-1}(i)$
        $c_j \leftarrow R[i]$
        Send c_j to party P_j
procedure COMPUTEBIN($V[1..n]$, n, k)
    $R[1..n] \leftarrow$ empty
    for $i \leftarrow 1$ to n **do**
        $v \leftarrow$ decPL($V[i]$)
        $R[i] \leftarrow \lceil \frac{n-v}{n/k} \rceil$
    return R

is presented in Protocol 5. At a high-level, the protocol privately compares each possible pair securely. The aim is to construct a private comparison matrix, where the pairwise comparison is hidden from the certifier using Paillier. Later, by computing the sum of each column in the comparison matrix, the certifier figures out the rank of the corresponding parties, which will be used later to split parties into k groups.

The public pairwise comparisons of all pairs reveal the order of the inputs, which obviously violates the security constraint. Therefore, the protocol initially performs private pairwise comparison for all pairs using ENCRYPTEDPCOMPARE procedure introduced earlier in Section 3.3.3 which returns the resulting bit $[t_{ij}]$ of comparison $x_i \leq x_j$ encrypted. Using the re-encryption function, the resulting bit is transformed to a Paillier scheme. Additionally, the negation of the result is also provided to the certifier. This will allow the certifier to construct a private comparison matrix C_{ij} where $i, j \in \{1, .., n\}$ as shown is Figure 3.2. In brief, if $x_i \leq x_j$, then the comparison will return $[\![t_{ij}]\!] = 1$ and $[\![\overline{t_{ij}}]\!] = 0$. These results are stored in the indexes of C_{ij} and C_{ji}. Consider an example in Figure 3.2 where $x_1 = 3$ and $x_3 = 5$. The comparison of x_1 and x_3 is $x_1 \leq x_3 \equiv 3 \leq 5 \equiv 1$. Hence, $C_{13} = 1$ and $C_{31} = 0$. After comparing all pairs, the certifier computes the columnwise summation of all entries in the comparison matrix using homomorphic addition. The columnwise summation will give the number of ones, i.e. the input is greater than or equal to how many other inputs. Therefore, the summed values in the resulting vector show the ranking among n parties, i.e. if the entry in index i of the resulting vector is 0, that means the input x_i is the minimum input. If it is $n-1$, that means x_i is greater than all other inputs and it is the maximum. Consider the example in Figure 3.2. After the columnwise summation, the resulting vector is $< 1, 0, 2 >$, which means x_i is greater than one input, x_2 is not greater than or equal to any of the other inputs, and x_3 is greater than equal to two parties. Hence, the resulting vector shows the ranking of the corresponding inputs. Recall the certifier does not own SK_P; thus, it cannot learn the ordering information. To prevent the secure co-processor from learning the order of the values in the resulting vector

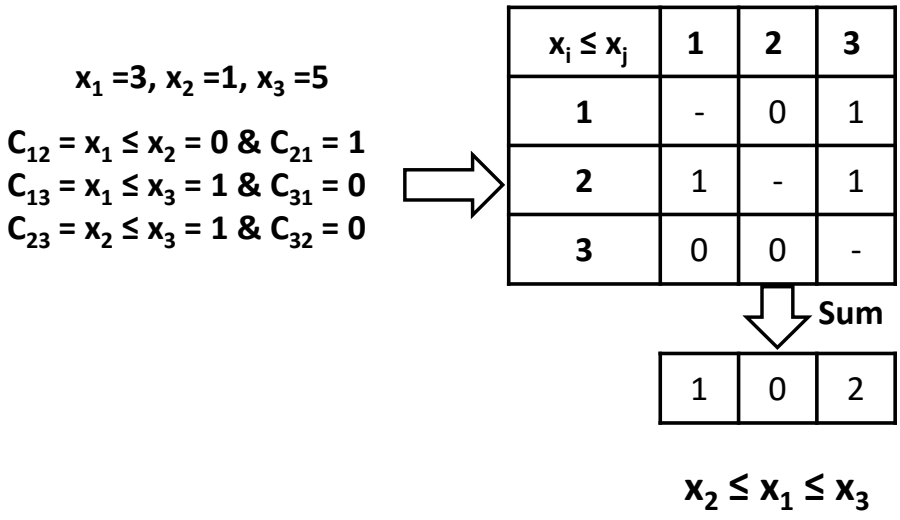


Figure 3.2: Private Comparison for Ordering

$c[1..n]$, the certifier applies a random permutation π. The i^{th} element of c is stored at index $\pi(i)$, $c_\pi[i] \leftarrow c[\pi(i)]$. Then, the permuted result vector is sent to the secure co-processor. The secure co-processor decrypts the entries in the permuted resulting vector, and computes the group of the inputs. After computing groups for all inputs, the secure co-processor returns the group vector, R, to the certifier. The certifier can compute $j \leftarrow \pi^{-1}(i)$ which represents the j^{th} index in the unpermuted order. After unpermuting the orders, the certifier returns the corresponding results to the parties, $c_j \leftarrow R[j]$.

Correctness. Th certifier first compares all pairs and constructs a comparison matrix such that $\forall i, j x_i \leq x_j \Leftrightarrow C_{ij} \leftarrow 1$ and $C_{ji} \leftarrow 0$ where $i \neq j$. The comparison can be one of the followings: (1) $x_i < x_j$, (2) $x_i = x_j$, and (3) $x_i > x_j$. For cases 1 and 3, the numbers are distinct, and the output of comparisons are $C_{ij} \leftarrow 1$ and $C_{ij} \leftarrow 0$, respectively. In case 2, the numbers are equal and it returns $C_{ij} \leftarrow 1$. In this case, $C_{ji} \leftarrow 0$. This means x_i is not greater than x_j. Although $x_i = x_j$, the comparison selects x_j greater and ranks it higher. The columnwise summation of the comparison matrix will form a resulting vector which shows the ranking of the inputs among all n parties. The smallest input will have an entry of 0 and the

maximum input will have an entry of $n-1$ which says this input is greater than or equal to $n-1$ other entries. Thus, the resulting vector will have entries from 0 to $n-1$ which are the ranks of the inputs. The correctness of the rest of the protocol is straightforward. The resulting vector has entries $0, 1, ..., n-1$ in some order. The secure co-processor decrypts the entries and split inputs into k groups (quantiles) based on their order among the n parties. For example, the inputs with entries $0, 1, ..., k-1$ will be in the first group.

Intuition of Security Proof. The certifier receives the inputs encrypted with Paillier. The certifier initially compares all pairs using the function ENCRYPTEDPCOMPARE which is followed by the execution of the re-encryption function. The private comparison and re-encryption functions are already proved secure in [78] and they do not reveal any information. The results of the pairwise comparisons are encrypted with Paillier and the certifier cannot decrypt the results due to its lack of knowledge of the private key, SK_P. To rank the inputs, the certifier computes the columnwise summation of the comparison matrix using the additive homomorphic properties of Paillier. Therefore, it does not learn any information about the inputs and the pairwise comparisons. On the other hand, the secure co-processor receives the resulting vector permuted. Although it decrypts entries in the permuted vector, it cannot infer any information about the relationship between the results and the parties, since it does not know the permutation. At the end of the certification, groups(quantiles) of parties are public, but neither the certifier nor the secure co-processor learn any information about the ordering of parties inside the same group. Thus, the quantile based certification is secure.

3.4.3 Private Certification with Private Outputs

Till now, the certification results are made public. As was discussed earlier, mutually competitive firms might want to gain private knowledge about their performances without revealing the result of the certification to the certifier, the computation helper and other parties. We now

describe necessary modifications to perform such certifications with private outputs on top of the proposed framework.

Mean Based Certifications with Private Outputs

The framework initializes the same setup as in the corresponding certification with public outputs except that a key pair K_{QR} is generated by the secure co-processor. SK_{QR} is only owned by the secure co-processor and PK_{QR} is shared with the certifier. To compare the private input with the encrypted threshold value, the certifier invokes the ENCRYPTEDPCOMPARE function from Protocol 2 until line 12 instead of the PRIVATECOMPARE function inside the MEAN-CERTIFY function. Line 12 from the ENCRYPTEDPCOMPARE function returns the result of the comparison encrypted with QR, $[t]$, to the certifier. Since, the certifier does not own SK_{QR}, it cannot decrypt and learn the result of the comparison. The certifier sends the resulting bits to the parties encrypted. The parties do not own the secret key SK_{QR}, thus, they need help from the secure co-processor to learn the actual results. To prevent the certifier and the secure co-processor from learning the actual results, the parties randomize their inputs by applying the same logic as in Protocol 3. In brief, each party chooses a random bit, r, and then computes $s_r \leftarrow [t].[0]$ and $s_{1-r} \leftarrow [t].[1]$. Both s_r and s_{1-r} are independent from the value of the resulting bit t. Each party sends their s_r and s_{1-r} to the certifier and the certifier sends them to the secure co-processor. The secure co-processor decrypts both of them and returns the unencrypted results to the certifier in the order received. The certifier also does the same and sends the unencrypted s_r and s_{1-r} to the corresponding party. Since the party knows r, it selects the correct result. If the result is 1, the party knows the label is *above*; otherwise, it is *below*.

Quantile based Certification with Private Outputs

The framework uses the same setup introduced in Section 3.4.2. The certifier executes the QUANTILE-CERTIFY functions as it is until line 21 in Protocol 5, where the secure co-processor computes the groups (quantiles) of inputs based on their order. After the secure co-processor computes the groups, it encrypts the entries of R using the COMPUTEBIN function, which are the group numbers (quantiles) of the inputs, with Paillier. Then, the secure co-processor returns R to the certifier. Since the certifier does not own SK_P, it cannot decrypt and learn which party is placed in which group. The certifier executes the rest of the protocol as is and sends the results to the parties encrypted. The parties do not have the secret key SK_P. Therefore, they need help from the secure co-processor. To hide the real results, each party selects a large enough random number r, and executes $[\![s]\!] \leftarrow [\![c]\!].[\![r]\!]$ which is equivalent to $[\![s]\!] \leftarrow [\![c+r]\!]$. Then, each party sends their inputs to the certifier and the certifier also sends these inputs to the secure co-processor. After decrypting $[\![s]\!]$, the secure co-processor sends s to the certifier in the clear. Note that since the random number r is hidden from both the certifier and the secure co-processor, they cannot learn the actual group number of the party. The certifier sends s back to the corresponding party. Upon receiving s, a party executes $s \leftarrow s - r$ and learns the group of the party.

3.5 Performance

Although the certification process is typically performed off-line, and hence might not require strict time constraints to complete the certification process, other applications might require instant feedback or certification based on the input. For example, a privacy preserving online auction system has to compare a private bid with the maximum provided private bid and announce the result quickly. In such cases, the efficiency and practicality of the proposed

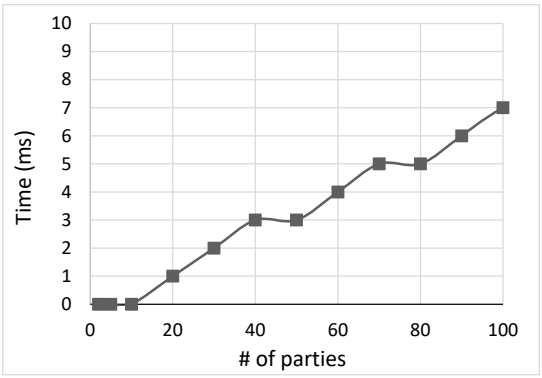
system really matter. Even if the addressed scenario does not prioritize fast certification, the proposed algorithms are secure and efficient. To show the performance analysis of our framework and algorithms, in this section, we present both empirical and complexity analysis for both the mean and k-quantile certifications.

3.5.1 Complexity Analysis

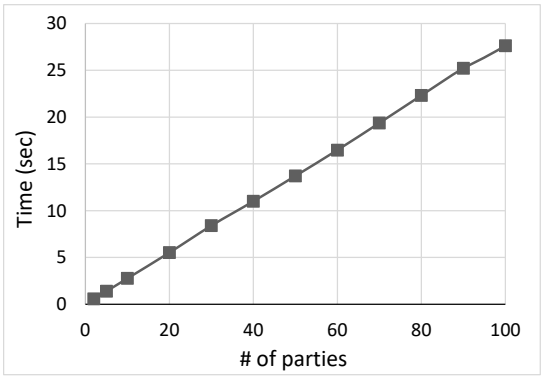
Mean and quantile based certifications rely on comparing encrypted data. This dissertation proposes two comparisons protocols, PRIVATECOMPARE and ENCRYPTEDPCOMPARE, which are adapted from Veugen's [98] protocol. Veugen discusses the complexity analysis of the encrypted comparison protocol and shows that encrypted comparison has a very low computation complexity. The main computation complexity occurs while two private integers are being compared. In the same paper, Veugen proposes a Lightweight Secure Integer Comparison (LSIC) which requires l rounds of communications plus half a round at the beginning. Our prototype also implements the LSIC algorithm to compare two integers privately. Both PRIVATECOMPARE and ENCRYPTEDPCOMPARE have one more round for transferring z and $[t]$. Therefore, our comparison protocols require $l + 1.5$ rounds of communications between the certifier and the computation helper (e.g. assuming a 32-bit integer domain: 33.5). In addition, the re-encryption procedure requires one round of communications.

3.5.2 Empirical Analysis

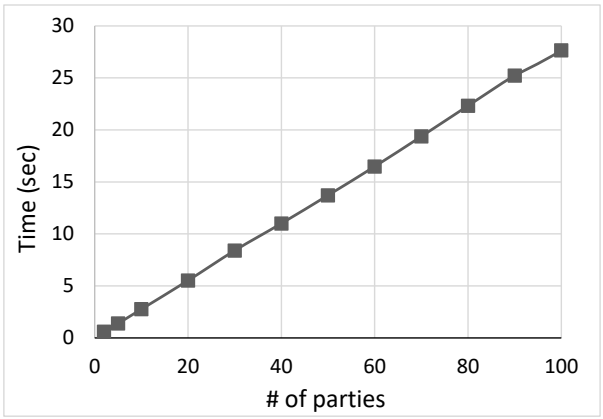
We implemented a prototype of the proposed framework in Java. The certifier is run on a Windows machine with i5-2320 3 GHZ CPU and 8 GB memory. On the other hand, the computation helper is run on a machine running Linux with Intel Xeon(R) E31235 3.20 GHZ CPU and 32 GB memory. Both machines are on the same network and the average latency between them is 0.1 ms. The parties are run on the same machine with the certifier. The data



(a) Homomorphic sum



(b) Time spent in comparisons



(c) Total execution time

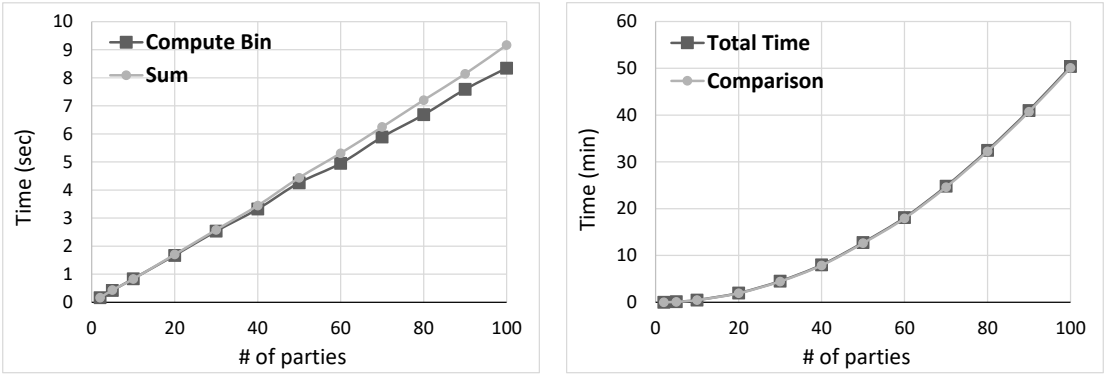
Figure 3.3: Results of Private Mean Certification

domain is 32-bit integers. The conducted experiments measure the execution time to evaluate system performance by varying the number of participating parties. The size of the keys for both the Paillier and the QR cryptosystems are set to 2048 bits.

Mean Certification

The mean based certification initially computes the average of inputs, and then compares each encrypted input with the average. Figure 3.3(a) and 3.3(c) present the execution times for homomorphic summation and total certification times, respectively.

Homomorphic summation is performed with modular multiplication. It is cheaper compared to encryption and decryption and this is also validated in our experiments. For very



(a) Homomorphic Sum and Bin Computation (b) Time spent in comparisons

Figure 3.4: Results of Private Quantile Certification

small number of parties, the average computation is performed in 0 or 1 ms. In the worst case, the homomorphic summation takes 7 ms (number of parties = 100). These results are very promising for other privacy-preserving database applications which need to perform aggregate operations as part of query executions.

The execution time of the mean based certification is dominated by the comparisons with the average (Figure 3.3(b)). The mean certification requires n comparisons against the computed average. In our implementation, the comparisons are sequential, therefore, both total execution time and time spent in comparisons have linear behavior. As the number of participating parties increases, the total execution time also increases. It is possible to perform comparisons in parallel which will decrease the total execution time, though this chapter does not discuss and implement parallelism. Even without such an optimization, the total certification times take seconds, with a maximum of 27.6 seconds when 100 parties participate. This is still well below a minute, and hence for many applications, especially environmental certification, is very reasonable.

4-Quantile Certification

The quantile based certification requires pair-wise comparison of each input data, which requires $(n^2 - n)/2$ comparisons. This quadratic behavior causes longer certification times as the number of participants increases as depicted in Figure 3.4(b). The other important sub-procedures inside the quantile certification protocol are the homomorphic summation of comparison values and the grouping computations. In our experiments, we set k to 4, that means the parties are split into 4 groups. The computation helper maps parties into groups in linear time. On the other hand, to get the final scores encrypted, the certifier performs $n^2 - n$ homomorphic summations before computing the bins. The quantile based certification of 20 participants is performed within 2 minutes though it takes slightly more than 50 minutes when there are 100 participants. However, it is not realistic to have hundreds of participants in our application scenario. It is expected to have 20-30 participants most of the time and the quantile based certification can be done within a few minutes in such settings, which is pretty efficient.

Discussion. There is a clear trade-off between the security and the performance/functionality. Our algorithms and framework enable achieving significant functionality with reasonable computation performance without sacrificing any performance. Our framework benefits from recent cryptographic tools to perform operations fast. Our evaluations show the advantage of the usage of secure co-processors as a computation helper on site. Recall that the average network latency between the certifier and the computation helper is 0.1 ms in our experiments, which makes the cost of rounds of interactions among two parties negligible compared to the computation cost. An on site secure co-processor also makes the network transmission time negligible. Recall that the encrypted comparison operations require $l + 1.5$ rounds of communication and the mean certification requires n comparisons while the k-quantile comparison requires $(n^2 - n)/2$ comparisons, which makes $n(l + 1.5)$ and $(n^2 - n)(l + 1.5)/2$ rounds of communications, respectively. A setting where there is a non-negligible latency between the

certifier and the computation helper will result in drastic performance degradation. Therefore, a secure co-processor perfectly fits the proposed model.

3.6 Conclusion

In this chapter, we formally define the *privacy preserving certification* paradigm to evaluate the environmental impacts of industrial processes privately and propose solutions for two certification problems-mean, quantile. To perform privacy preserving certifications without compromising any sensitive information, we propose a framework, which considers a realistic network communication model for the certification model, which enables a certifier to certify parties based on a well agreed upon set of criteria. The chapter also presents efficient and provably secure algorithms for the certification problems. Our simulation/prototype demonstrate that the proposed approach is not only secure but also efficient and practical.

Chapter 4

Differentially Private LCA Computations

One of the greatest challenges facing global society is to ensure that the industrial goods and services required by a growing and modernizing population can be met sustainably and equitably [105]. Industrial Ecology (IE) is the study of resource requirements and the social and ecological implications of industrial activities. Its primary utility is to inform consumers, businesses, and policy makers about the magnitude and significance of *material flows* through the economy that supports specific products, technologies, or systems [106]. One primary technique in IE is life cycle assessment (LCA), a standardized methodology for estimating the total environmental implications of products or services [107, 108]. The core methodology of LCA is governed by a set of international standards [40] and is widely applied to evaluate the potential ecological consequences of consumption decisions.

Preparing an LCA requires access to a database of information about the inventory requirements and environmental emissions of industrial processes, called a life cycle inventory (LCI) database. Preparing an accurate and comprehensive LCI database is a tremendous task and the development and maintenance of these resources is an ongoing challenge [73]. Because industrial processes are typically undertaken in a competitive economic context, the operators of these processes would like to prevent potential competitors from learning sensitive informa-

tion about their activities. Information that may be valuable to a competitor is often termed *confidential business information*. Inventory data about industrial processes is usually considered to be confidential, and therefore is often not available freely. This type of information is nonetheless required in order to accurately assess environmental impact. As a consequence, the historical development of LCA has long been intimately bound to questions of confidentiality [109, 110].

Despite its centrality to LCA, data privacy in the LCA domain has not been formally considered. In particular, methods for privacy-preserving data publication in LCA have not been well-developed. The guiding principle behind privacy protection in LCA database preparation is that data that are regarded as secret by the owners can be concealed through aggregation with other data sets and with data sets extracted from LCI background databases (see [73], ch. 3).

In this dissertation, we formulate the LCA computation in a way that allows us to introduce a privacy model, and consider possible threat models and attacks that could result in an adversary learning private data. Our goal in this work is to provide the data security community with a real sense of the challenges faced by practitioners in the field of Industrial Ecology. We explore a particular problem in LCA and explore the privacy issues and possible trade-offs between increase transparency by industrial companies and privacy protection of trade secrets that preserve competitive edge. The results of our attacks justify the concerns over publishing inventory data about industrial processes without securing with any security. To tackle this problem, we apply privacy techniques to LCA computations and illustrate their usage on a specific real life example. Our evaluations over a real life example highlight that it is possible to achieve privacy-preserving LCA publication without losing too much utility on the published data while ensuring privacy with the application of differential privacy. A straightforward optimization such as normalization, considering the idiosyncratic features of LCA data, delivers a reasonable improvement in the publication quality without sacrificing the privacy.

The followings summarize our contributions in a nutshell:

- The first formal privacy-preserving LCA computation formulation while providing more transparency.
- Verify privacy concerns of LCA practitioners by developing an attack.
- Develop a differentially private matrix multiplication that is particularly efficient in the LCA context.
- Evaluate the proposed privacy-preserving publications and propose optimization to improve publication utility.

The rest of the chapter is organized as follows. The next section formulates the LCA aggregation problem and explains current practice along with privacy concerns. Section 4.2 investigates the validity of privacy concerns in LCA publications. Differentially private LCA publication techniques are presented in Section 4.3. The following section presents experimental evaluation. The final section concludes the chapter.

4.1 Formulating the LCA Aggregation Problem

4.1.1 LCA Basics

LCA following the ISO standards describes the delivery of a product or service as a network of industrial *unit processes* whose outputs are required in order to provide a *functional unit* of utility to a user. Each unit process represents one form of industrial activity. Each edge in the network indicates a *flow* from one process to another, or between one process and the environment. Flows between processes are called *intermediate* flows, and flows between a process and the environment are called *elementary* flows. Only elementary flows may generate environmental impacts [40, 100].

LCA studies distinguish between a *foreground model*, which represents the activities under scrutiny, and a *background model*, which represents the operations of the broader economy [111]. Private data are typically contained in the foreground model. The preparation of a background database is outside the scope of an individual study, and background databases are provided and maintained by dedicated research [112] or commercial [113] organizations. Although background databases are subject to licensing restrictions, in this study they are regarded as publicly available because any party who purchases a license may inspect them freely. Background databases are assumed to be available in an aggregated form in which the relations among the different processes are not known.

An *LCA aggregation study* can be described as three sequential matrix multiplications with respect to a background database B_x [101]. B_x is an $m \times n$ matrix that maps a set of n background processes to a set of m elementary flows. The foreground model is made up of a set of p foreground processes, each of which is defined by its dependencies on the n background processes. These are described in an $n \times p$ dependency matrix A_d, which comprises the study's private input data. Here w is a p-element weighting vector that specifies the relative significance of the different foreground processes. The first multiplication aggregates the foreground model into a weighted dependency vector a_p:

$$a_p = A_d \cdot w \tag{4.1}$$

The dependency vector a_p is then applied to the background database to determine an emission vector b:

$$b = B_x \cdot a_p \tag{4.2}$$

The vector b, also called a *life cycle inventory*, reports the aggregate amounts of different emissions released into the environment throughout the life cycle of the product system speci-

fied. The results of the inventory computation must be characterized with respect to a set of t environmental impact categories, represented by multiplication with a $t \times m$ characterization matrix E.

$$s = E \cdot b \tag{4.3}$$

This multiplication results in a set of t impact scores s , which are the final results of the study. The impact scores in s provide a basis to compare different product systems with equivalent functional units on the basis of their potential environmental impacts.

4.1.2 Privacy Concerns and Current Practice

The current practice in the Industrial Ecology community is to make the result of the study s (Equation 4.3) publicly available, so that the product system they represent can be compared to other product systems. However, it is difficult to evaluate the significance of the elements of s without knowing something about b. For instance, an independent researcher making a critical evaluation of s may wish to know whether a given environmental emission was included in b with a significant value. Alternatively, a practitioner may require further knowledge about the flows in b, such as their geographic or temporal scope. Some research questions may require a practitioner to supply her own E matrix, which is not possible if b is not disclosed.

On the other hand, these requirements raise several privacy concerns over the data in A_d, for which a_p is a proxy. In the absence of a formal understanding of the privacy implications of disclosing b, it is common practice in the community to withhold b and only publish s. As mentioned earlier, B_x can be regarded as public, and so there is conceivable risk that a_p could be back-computed from b if it is fully released. On the other hand, the release of an obfuscated form of b may permit certain research questions to be answered while still ensuring privacy. In order to support the needs of the sustainability research community, it is necessary

to understand the relationship between disclosure of b and exposure of elements of a_p.

4.2 Confidentiality & Privacy Issues

As explained in Section 4.1, b is an emission vector which reports the amount of exchange for each emission during a production or a service. b contains important information both for environmental analysis and marketing decisions. However, LCA practitioners are hesitant to publish b due to their fear of information leakage concerning details of a_p, and hence potentially trade secrets that give a specific company a competitive edge over its competition. The question is whether the practitioners are right or not in their concerns. Here, we investigate the possible information leakage out of the publication of b. In other words, how much of a_p can be recovered when b is published, given that B_x is public and b is derived from the factorization of B_x and a_p as shown in Equation 4.2?

4.2.1 Industrial Ecology Privacy Concerns

The operations of an LCA aggregation study is sequential matrix multiplications. If B_x is a nonsingular (invertible) matrix, there exists a unique inverse denoted by B_x^{-1}, i. e., $B_x \cdot B_x^{-1} = B_x^{-1} \cdot B_x = I$. Then, Equation 4.2 has a unique solution, $a_p = B_x^{-1} \cdot b$. This might be seen as a justification of the concern not to publish b along with impact scores, s. However, B_x in LCA is a singular matrix most of the time, which means it is not invertible and a_p cannot be solved directly from Equation 4.2. Is this enough to ensure security guarantees?

The answer to this question is unclear. The concept of Moore-Penrose pseudoinverse of matrices [114], generalizes the notion of a nonsingular (invertible) matrix and makes it applicable to singular matrices. This concept is useful when someone searches for an optimal approximation of a set of linear equation solutions like $A \cdot x = y$, where A is a known $m \times n$ matrix, y is a column vector with m components and x is an unknown column vector. x is the

solution for the linear system, which usually leads to the minimum *least square* of $(A \cdot x - y)$. A common approach to compute the pseudoinverse is to use the Singular Values Decomposition (SVD) [115]. This approach can be directly applied in the LCA study to reveal the secret a_p vector with some approximation. The Moore-Penrose pseudoinverse has already been employed to solve different problems like digital imaging methods [116, 117] and astronomical data analysis [118]. A key question is to what extent the $\overline{a}_p$ vector computed from the pseudoinverse allows an attacker to reconstruct a_p. The next section investigates the power of the pseudoinverse technique to reveal industry secrets.

4.2.2 Revealing Industry Secrets using Moore-Penrose Pseudoinverse

This section briefly explains the features of the Moore-Penrose pseudoinverse [114] in terms of its capabilities and limitations. The pseudoinverse of a matrix A is denoted by A^+. For any matrix A, it is known that there exists only one Moore-Penrose inverse A^+, i. e., uniqueness. The general psudoinverse solution to a linear system $A \cdot x = y$ is:

$$x = A^+ \cdot y + (I - A^+ \cdot A) \cdot q \tag{4.4}$$

where q is an arbitrary vector of appropriate order. Since q is arbitrary, there exists an infinite number of solutions when $(I - A^+ \cdot A) \neq 0$. A natural question is whether there is a case where $(I - A^+ \cdot A) = 0$. The answer is in the affirmative when A has a full column rank [119], $A^+ = (A^T \cdot A)^{-1} \cdot A^T$. Having a full column rank guarantees a unique solution to

x as seen from the following derivation:

$$\begin{aligned} x &= A^{+} \cdot y + (I - A^{+} \cdot A) \cdot q \\ &= A^{+} \cdot y + (I - (A^{T} \cdot A)^{-1} \cdot A^{T} \cdot A) \cdot q \\ &= A^{+} \cdot y + (I - I) \cdot q = A^{+} \cdot y \end{aligned} \tag{4.5}$$

In the context of LCA, to the best of our knowledge, having a full column rank in B_x matrix is very rare. The columns are not completely independent from each other which leads to having an infinite number of solutions for the linear system. One can claim that having an infinite number of solutions for x will create enough ambiguity and an adversary will not be able to distinguish which x is close to the original one. However, our empirical studies over a real LCA study disprove this and show that one can solve the linear system approximately close enough using the Moore-Penrose pseudoinverse as we will explain in detail later in Section 4.4. Therefore, we need to ensure the security of publication which prevents an adversary from recovering the solution even with the usage of Moore-Penrose inverse. In the context of privacy-preserving data publication, differential privacy becomes a canonical technique due to its strong privacy guarantees and capability to release useful aggregation information. Given that an LCA study is an aggregation problem, we propose differentially private LCA publications. The next section explains differential privacy and its usage in the context of LCA publication in detail.

4.3 Achieving LCA Privacy

4.3.1 Background: Differential Privacy

Differential privacy provides a strong notion of privacy and is commonly used for statistical data publication [28]. It ensures that the removal or addition of a single record does not significantly affect the outcome of any analysis. It quantitatively bounds how much a single record can contribute to a public output. The formal definition of differential privacy is [28]:

Definition 4.3.1 *A random mechanism M gives ϵ-differential privacy if for any neighboring data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(M)$,*

$$Pr[M(D_1) \in S] \leq e^{\epsilon} \cdot Pr[M(D_2) \in S] \tag{4.6}$$

Differential privacy can be achieved by the addition of random noise. The magnitude of the noise is chosen based on the sensitivity of a query function which considers the largest change in the output of the function with a change of a single record. Such a change is referred to as the *global sensitivity* of a function [28].

Definition 4.3.2 *For any function $f\colon D^n \rightarrow \mathbb{R}^d$, the sensitivity of f is:*

$$\Delta f = \max_{D_1, D_2 \in D^n} \parallel f(D_1) - f(D_2) \parallel_1 \tag{4.7}$$

for all D_1, D_2 differing in at most one element.

For example, for counting queries, the global sensitivity of a function is 1, since inclusion or exclusion of a single record changes the output of a function by at most 1.

Dwork [28] suggests using the Laplace mechanism to add noise to achieve differential privacy and this has become a canonical approach for differentially private systems. Here, we revisit the differentially private Laplace mechanism.

Theorem 4.3.3 *The randomized mechanism M_F for a query function $f : D^n \rightarrow \mathbb{R}^d$, computes $f(x)$ and adds a noise sampled from the Laplace distribution to each of the d outputs satisfies ϵ-differential privacy [47]. For such a function, the Laplace mechanism is defined by*

$$M_F(x) = f(x) + (Y_1, Y_2, ..., Y_d) \tag{4.8}$$

where Y_i is drawn from the Laplace function $Lap(\Delta f/\epsilon)$.

A relaxed form of differential privacy, called *approximate differential privacy* or (ϵ, δ)-*differential privacy* for short, is introduced by Dwork et al. [120]. The approximate differential privacy can be achieved using Gaussian noise calibrated to the L_2 sensitivity.

Definition 4.3.4 *L_2 sensitivity of a real valued query function g: $D^n \rightarrow \mathbb{R}$:*

$$\Delta g = \max_{D_1, D_2 \in D^n} \parallel g(D_1) - g(D_2) \parallel_2 \tag{4.9}$$

for all D_1, D_2 differing in at most one element.

Theorem 4.3.5 *The randomized mechanism M_G for a query function g, computes $g(X)$ and adds a noise sampled from the normal distribution $N(\mu,\ \sigma^2)$ where μ and σ^2 are mean and variance, respectively. For such a function, the Gaussian mechanism is defined by*

$$M_G(D) = g(D) + N(0, \sigma^2) \tag{4.10}$$

where $\sigma = \Delta g \sqrt{2ln(2/\delta)}/\epsilon$. M_G provides (ϵ, δ)-differential privacy.

4.3.2 Differential Privacy for LCA Computation

The main motivation of this work is to perform differentially private LCA matrix multiplication in the form of Equation 4.2, where no adversary is able to recover a_p from the published

b vector. Recall that B_x is a publicly known matrix. In this section, we develop two differentially private matrix multiplication mechanisms that will be used later to achieve differentially private publication for LCA computations.

Each element in the a_p vector represents a background process that is included in the production. The privacy goal is to make a publication such that either inclusion or exclusion of a specific background process from the computation has a negligible effect on the output, which is vector b. To achieve this goal, differential privacy might be applied by either perturbing the input or the output.

Input Perturbation

The initial way to achieve differential privacy is to add noise to the input data itself. In the LCA context, the a_p vector contains sensitive information. To achieve ϵ-differentially private computation, the straightforward approach is to generate a differentially private version of a_p, and then perform matrix factorization. Similarly, the (ϵ, δ)-differentially private a_p vector can be published using the Gaussian mechanism, and then it is used in the matrix computation.

In this case, the global sensitivity of the publication considers the maximum change in all possible neighboring vectors.

Definition 4.3.6 *Let $\mathbb{R}$ denote the set of real numbers. For $x_1,\ x_2 \in \mathbb{R}^d$, the sensitivity of the publication:*

$$\Delta f_1 = \max \parallel x_1 - x_2 \parallel_1 \tag{4.11}$$

for all x_1, x_2 differing in at most one element in the vector.

Assume $x_1^1, x_1^2, .., x_1^d$ are the elements of x_1 and $x_2^1, x_2^2, .., x_2^d$ are the elements of x_2 such that $\forall i, j \in [1, d]$, $x_1^i, x_2^j \in [0, N]$. If x_1 and x_2 differ in one element, the maximum change in the publication (global sensitivity) will be N.

Although having a data independent sensitivity computation is a desired feature in differentially private publications, the sensitivity computation in our context is data dependent. In theory, the sensitivity is unbounded and can be infinity. Given this fact, differential privacy might be considered as an inappropriate methodology for differentially private LCA computations. However, this is not the case. LCA data modeling has its own characteristics like sparsity, data distribution, which make differential privacy work in the practice for the LCA computations. Later, in this section we will develop a probabilistic estimated variance formulation which is a measurement of utility of an LCA publication.

Now, we can formally define our differentially private vector publication mechanism.

Proposition 4.3.7 *The randomized mechanism M_K that outputs the following vector is ϵ-differentially private:*

$$M_K(x) = x + k \tag{4.12}$$

where k is a vector consisting of n independent samples drawn from the Laplace distribution function with a scale $\Delta f_1/\epsilon$, i. e., $Lap(\Delta f_1/\epsilon)$.

Proof: Recall that x is a vector consisting of the true answers. M_K mechanism adds independent Laplace noise to each element of x. Thus, the output of M_K is a vector of length d containing a noisy answer for each element in x. The M_K mechanism incorporates the features of Theorem 4.3.3, hence, satisfies ϵ-differential privacy. ■

Recall that, our motivation is to publish vector b in LCA computation, not a_p. Using the M_K mechanism, it is possible to publish ϵ-differentially private a_p. Now, the differentially private version of a_p will be used to compute resulting b vector.

Proposition 4.3.8 *Given a public $A \in \mathbb{R}^{m \times n}$ and private $x \in \mathbb{R}^n$, the randomized mechanism M_{F_1} that performs the following operation ensures ϵ-differentially privacy for x:*

$$M_{F_1}(A, x) = A \cdot M_K(x) \tag{4.13}$$

Proof: M_{F_1} uses a differentially private (obfuscated) version of x, generated by mechanism M_K, in the matrix factorization. A is known by the public and transforming to the LCA computation A stands for a B_x matrix where the rows are emissions and the columns are processes that are included in the study. The mechanism has to ensure that either the inclusion or removal of a process should not reveal any information about the process. It is already proven that the M_K mechanism ensures differential privacy. The factorization of $A.M_K(x)$ is a post processing over the differentially private x. The factorization does not have any access to the original x matrix, hence it does not violate differential privacy. Although the mechanism outputs only the result of the factorization, assume that an adversary tries to find the original x vector by solving the linear system. In the best case, the adversary will get $M_K(x)$ by solving the linear system which is already proven ϵ-differentially private. Therefore, M_{F_1} mechanism ensures ϵ-differential privacy for x. ■

Expected Variance of Error. To measure the utility, we analyze the accuracy of resulting vector. Let y denote the factorization of $A.M_K(x)$ where $y_1, y_2, .., y_m$ are the elements of y. We use y_i to denote the correct value, $\widehat{y_i}$ to denote differentially private result, and $E_{M_{F_1}}(y_i)$ to denote the absolute error for y_i with M_{F_1} mechanism such that:

$$E_{M_{F_1}}(y_i) = |y_i - \widehat{y_i}| \tag{4.14}$$

Given each y_i is randomized, $E_{M_{F_1}}(y_i)$ is a random variable. Since y has m elements, the average variance of error (the mean squared error) of M_{F_1} is:

$$\mathrm{Var}_{\mathrm{avg}}(M_{F_1}) = \frac{\sum_{k=1}^{m}(E_{M_{F_1}}(y_k))^2}{m} \tag{4.15}$$

In the M_{F_1} mechanism, each element of x is added a noise sampled from the Laplace distribution $Lap(\Delta f_1/\epsilon)$. The variance at each element, therefore, $\mathrm{Var_e}$ = $2.(\frac{\Delta f_1}{\epsilon})^2$. Note that the sampled random variables are uncorrelated. In the factorization, for each row, the j^{th}

element of A is multiplied by the j^{th} element of obfuscated x.

$$\mathrm{Var_i}(\mathrm{E}_{\mathrm{M}_{\mathrm{F}_1}}(\mathrm{y_i})) = \sum_{k=1}^{n} A_{ik}^2.\mathrm{Var_e} \tag{4.16}$$

In the factorization, each element of A is a weighting constant. It corresponds to the B_x matrix in LCA computations. Without modeling the LCA study completely, it is possible to estimate A_{ik} if the underlying data distribution of A is known. A consists of $m \times n$ discrete values, we can define the probability density function $g(z_i)$, such that for any z_i, which is a value that Z can take, g gives the probability that the random variable Z equals z_i:

$$\begin{aligned} P(Z = z_i) &= g(z_i) \quad i = 1, 2, ... \\ g(z_i) &\geq 0, \sum_i g(z_i) = 1 \end{aligned} \tag{4.17}$$

Then, the expected value for Z is:

$$\mathrm{Ep}(Z) = \sum_z z.g(z) \tag{4.18}$$

Using the expected value for any entry in A, we can compute the expected error variance of y's elements in the following way.

$$\mathrm{Ep}(\mathrm{Var_i}(\mathrm{E}_{\mathrm{M}_{\mathrm{F}_1}}(\mathrm{y_i}))) = n.\mathrm{Ep}(Z)^2.\mathrm{Var_e} \tag{4.19}$$

The final step is to compute the expected average error variance for the M_{F_1} mechanism.

$$\mathrm{Ep_{avg}}(\mathrm{Var_{avg}}(M_{F_1})) = \frac{n^2.\mathrm{Ep}(Z)^4.\mathrm{Var_e}}{m} \tag{4.20}$$

The expected average error variance depends on the data distribution of A, and there is no

boundary for the error. However, in the LCA context, the B_x matrix is sparse most of the time and most of the entries are either zero (0) or close to zero, which makes the expected average error variance low. Although unbounded sensitivity is a problem, the characteristics of LCA publications enable differentially private publication to deliver significant utility, which will later be discussed and verified in Section 4.4.2.

Output Perturbation

To achieve differential privacy by perturbing the output, the desired differentially private mechanism initially computes the function, and then adds noise to each element of the computed output to obtain differentially private publication. Similar to the previous setting, A is a public matrix and x is a private vector which we want to preserve its privacy.

Definition 4.3.9 *Let $\mathbb{R}$ denote the set of real numbers where $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$. A matrix multiplication function f:$\mathbb{R}^{m\times n} \times \mathbb{R}^n \to \mathbb{R}^m$ is defined by:*

$$f(A, x) = A \cdot x \tag{4.21}$$

The output of f is an m-dimensional vector. To achieve differentially private matrix multiplication, the noise should be generated based on the sensitivity of f. The sensitivity of f considers the maximum change in the output with a single change in the vector x. The defined function is a matrix multiplication, thus, a single change in x will result in changes in every entry of the output. We consider the maximum change as a sensitivity with a single change.

Definition 4.3.10 *For $x_1, x_2 \in \mathbb{R}^n$, $A_1, A_2 \in \mathbb{R}^{m\times n}$, the sensitivity of $f(A, x)$:*

$$\Delta f_2 = \max \parallel f(A_1, x_1) - f(A_2, x_2) \parallel_1 \tag{4.22}$$

for all x_1, x_2 differing in at most one element.

When an element is excluded from the x vector, the corresponding column is also excluded from A to perform matrix multiplication. For example, if the second entry from the x vector is excluded, the second column of matrix A should also be removed from A to perform multiplication operation consistently. Basically, this is an exclusion of a process from an LCA model and observation of its effect.

In the proposition below, we define a differentially private matrix multiplication mechanism.

Proposition 4.3.11 *Given a matrix multiplication function $f(A, x)$, the randomized mechanism M_{F_2} that outputs the following vector is ϵ-differentially private:*

$$M_{F_2}(A, x) = f(A, x) + k \tag{4.23}$$

where k is a vector consisting of m independent samples drawn from the Laplace distribution function with a scale $\Delta f_2/\epsilon$, i. e., $Lap(\Delta f_2/\epsilon)$.

M_{F_2} initially executes $f(A, x)$ which outputs the multiplication of A with x. Then, the mechanism adds a randomly sampled vector k to the result of the multiplication to obfuscate it.

Proof: M_{F_2} incorporates the features of Theorem 4.3.3, which states that a random mechanism satisfies ϵ-differential privacy *iff* each output of a function is added a noise sampled from the Laplace distribution. M_{F_2} initially, performs matrix multiplication, and then adds a noise to each element in the resulting vector. Therefore, M_{F_2} is ϵ-differentially private. ■

Expected Variance of Error. Let y denote the result of $M_{F_2}(A, x)$. The absolute error is caused only by the addition of random noises sampled from the Laplace distribution. There-

fore, the error variance of y's entries:

$$\mathrm{Var_i}(\mathrm{E_{M_{F_2}}(y_i)}) = \mathrm{Var_e} \tag{4.24}$$

where $\mathrm{Var_e} = 2.(\frac{\Delta f_2}{\epsilon})^2$ for M_{F_2}.

Since all noises are independently generated and have the same variance, the average error variance is:

$$\mathrm{Var_{avg}}(\mathrm{E_{M_{F_2}}(y_i)}) = \frac{\mathrm{Var_e}}{m} \tag{4.25}$$

The average error variance is again data dependent, but as it will be verified with a real life example in the next section, it is likely for LCA computations to preserve the utility of differential privacy.

4.4 Evaluation of Privacy-Preserving LCA computation

To evaluate the security concerns and challenges of an LCA publication and the effects of differential privacy, we conducted experiments over a real LCA study for *distillers grain*. Using U.S. Life Cycle Inventory (USLCI) [121], we design and build a case study for distillers grain.

Data sets: The distillers grain study contains 39 background processes and 378 elementary flows. Therefore, a_p is a 39-dimensional vector and B_x is a 378×39 matrix. The distinctive property of this data set is having a very broad range of numbers. The entries in the matrices range from 10^{-15} to 10^3. We will later explain the effects of having numbers from such a wide range.

This section initially presents attacks to demonstrate whether there is a need for privacy preserving publication in reality, given that the only motivation is to make b public. Due to

the special properties of the LCA data, the answer to this inquiry is affirmative. Therefore, the section continues with the detailed guideline on applying differential privacy to an LCA computation where the aim is to make the publication useful (high utility) while still preserving the privacy.

4.4.1 Attack against LCA publication with Public b

The attack is formed to understand the security and privacy breaches in LCA publication. Suppose an LCA practitioner wants to publish b. She computes b using Equation 4.2 and makes it publicly available while not providing any information about a_p. As stated before, B_x is publicly known. The LCA practitioner thinks the computation is secure, since B_x is a singular matrix and there is no way to recover a_p. An adversary, on the other hand, is interested in learning information about a_p, since this vector contains confidential information regarding production processes which could be used to gain financial benefits.

Attack with Pseudoinverse: The attacker develops its attack by computing the Moore-Penrose pseudoinverse of B_x which is covered in Section 4.2.2. The rank of B_x is 29 -not a full column rank-. This means the solution to the $B_x \cdot a_p = b$ linear system is not unique. The common approach to resolve this issue uses the least square approach to optimize the approximation for a_p. This will output an approximate solution that is denoted by $\widehat{a_p}$. There are variety of ways to measure the distance between two vectors all of which might provide different results. To measure the closeness of the output, the Euclidean distance is used in this study. Additionally, the computations provide details about how many entries in the vectors are close within a given threshold. We use *close enough* as a term to express that the distance between the approximated value and the actual value is less than a provided threshold, which basically means an adversary approximate enough to recover the actual value. For example, consider a scenario where the first entry in a_p is 3 and the attacker finds the first entry of $\widehat{a_p}$ to be 2. If the threshold is 0.5,

the comparison indicates that the outputs are far away from each other and this is a failure for the attacker. However, if the threshold is 2, the attacker recovers the entry approximate enough and this is a success.

When the attack is executed[1] using the distillers LCA data, the distance between the actual a_p and the computed vector $\widehat{a_p}$, i. e., $\| a_p - \widehat{a_p} \|_2$, is 0.6558. Although the distance seems close, the attacker is able to approximate only 3 processes out of 39 close enough when the threshold is 10^{-10}. However, this is still a good source of information to claim that a practitioner is not able to secure a_p completely and the data is breached.

Furthermore, in a piratical setting, many entries of the a_p vector are 0 for the distillers grain data set. It is reasonable to assume that an expert in the field has enough background knowledge to estimate which processes are included in the computation pretty well. In such a case, the expert can develop a stronger attack against the publication of b and hence knowledge of a_p by removing all zero entries from ap which will result in the removal of the corresponding columns in B_x. In our study, 19 entries of ap are 0. When these entries are removed from the computation, the attacker has 20 entries to estimate. The new B_x matrix does not have a full column rank (it is 17). When the attacker solves the linear system using the pseudoinverse technique, the distance $\| a_p - \widehat{a_p} \|_2$ equals 0.15559. Compared to the initial case, it is a more powerful attack and the attacker is able to approximate 13 processes out of 20 when the threshold is 10^{-10}. Given that the attacker already knows the zero entries, she manages to recover almost 82.05% of a_p. The conducted experiments outline the power of pseudoinverse approach in the context of LCA domain. The important reasoning for such a good approximation is the domain range of the LCA data. The case study contains many very small numbers and this helps in approaximating the result better.

These attacks show that publishing b without securing with any privacy technique has severe security issues and the concerns over making b public in the LCA community are justified.

[1]The Singular Value Decomposition technique is used to compute the pseudoinverse.

Therefore, the publication should be made privacy-preserving. Next, this work applies different differential privacy techniques to secure the publication. The publications are again attacked by the adversaries to test the security of the publication in practice.

4.4.2 Differentially Private LCA Computation

In this section, we explain how to perform differentially private LCA computation efficiently by using the mechanisms that are introduced in Section 4.3.2 and evaluate the efficiency of the publication in terms of utility and security. The main metric to provide comparison is again the *Euclidean distance* of matrices for both utility and security. To ensure better utility, it is better to have a smaller distance between the original and the computed matrices. However, it is desired to have a larger distance between matrices to achieve better security.

Given B_x and a_p, the randomized mechanism $M_{F_1}(B_x, a_p)$ ensures ϵ-differentially private matrix multiplication by perturbing a_p first and then factorizing it with B_x (Proposition 4.3.8). $\widehat{b}$ denotes the obfuscated version of b vector. The LCA practitioner publishes the obfuscated version and keeps any version of a_p private. If an adversary solves the linear system of $B_x \cdot \widehat{a_p} = \widehat{b}$ perfectly, she ends up having $\widehat{a_p}$ int the best case. Since $\widehat{a_p}$ is ϵ-differentially private, privacy is still guaranteed.

Table 4.1 presents the results of privacy-preserving LCA computation with the M_{F_1} mechanism. The experiments are conducted by varying the ϵ security parameter. Δb denotes the Euclidean distance between the original b vector and $\widehat{b}$. $\Delta\widehat{a_p}$ measures the distance between a_p and $\widehat{a_p}$ where $\widehat{a_p}$ is the output of the M_K mechanism (Proposition 4.3.7). This explicitly depicts the effect of random noise addition. Assume that an adversary finds an approximate solution, denoted by $\overline{a}_p$, to $B_x \cdot \widehat{a_p} = \widehat{b}$ using the pseudoinverse approach. Δa_p is defined as the Euclidean distance between a_p and $\overline{a}_p$.

It is a well-known fact that when ϵ is small, the amount of noise addition is larger but en-

Table 4.1: M_{F_1} Mechanism For Matrix Factorization

ϵ	Δb	$\Delta \widehat{a_p}$	Δa_p
0.01	180.1E3	498.934	447.95
0.05	94216.22	94.86	91.449
0.1	128531.08	32.099	29.876
0.5	14580.31	5.348	5.144
1	5393.87	7.66	7.247
2	3840.31	2.03	1.783
10	2727.62	0.464	0.44
100	247.28	0.044	0.162

sures more security [122]. As ϵ increases, less noise is added which results in more utility. Finding the correct ϵ value for differentially private systems is a well studied research problem [122, 123]. $\ln 2$ and $\ln 3$ are widely used ϵ values for differentially private applications. This suggestions are also applicable in our context. We assume $\epsilon \approx 1$ is an ideal setting in our context.

When ϵ is 0.01, the distance between differentially private a_p and the original a_p is maximum, 498.934. When ϵ is 1, this distance is 7.66, which is also not very small. It is easy to infer that the noises are sampled with a large scale from the Laplace distribution. The main reason for this is that the values of a_p range from 6.48×10^{-8} to 0.7. In order to hide the existence of a single record, the differential privacy mechanism adds large noises since the sensitivity is too high.

The change in b is relatively large as a result of the M_{F_1} mechanism. It seems that the small perturbations in a_p introduce large perturbations in b. Such a system is referred to as *ill-conditioned* [124]. When ϵ is 1, Δb is 5393.87. This can be inferred as too much utility loss. However, when the result of the computation is analyzed in detail, 165 elements out of 378 (44%) are approximately close within the threshold of 10^{-10} when ϵ is 1. If an analyst wants to make a study for individual emissions, such a publication is very useful. The other important feature of this publication is its privacy. When an attacker executes the attack described before,

Table 4.2: M_{F_2} Mechanism For Matrix Factorization

ϵ	Δb	Δa_p
0.01	3311.5	9.91E8
0.05	697.663	8.23E7
0.1	309.63	8.98E7
0.5	69.421	1.11E7
1	40.601	222.4E4
2	17.467	509.8E4
10	2.865	634.2E3
100	0.339	102.1E3

she cannot recover a_p at all. The attacker computes $\overline{a}_p$ which has a distance of 7.247 from a_p. More importantly, even if she knows the location of all zero elements in the vector, she cannot approximate even 1 element out of 20 within a threshold of 10^{-10}. This validates the strong privacy guarantee of the M_{F_1} mechanism.

To achieve a differentially private LCA publication with the output perturbation, the M_{F_2} mechanism is proposed (Proposition 4.3.11). This approach initially computes b by multiplying B_x with a_p, and then obfuscates b by adding a random noise vector. a_p is again kept secret and the obfuscated emission vector $\widehat{b}$ is made public.

Table 4.2 presents the experimental results of privacy-preserving LCA computation with the M_{F_2} mechanism. This kind of publication reduces utility less compared to the earlier publication with M_{F_1} when Δb results are considered. When ϵ is 1, Δb equals 40.601 when M_{F_2} is used. It is 5393.87 when the publication is done with the M_{F_1} mechanism for the same ϵ. However, when the results are analyzed in detail, none of the entries in $\widehat{b}$ is close enough to the entries in b within the threshold of 10^{-10}. As explained before, in a similar setting, the M_{F_1} outputs 44% of the entries close enough. The trade-off between M_{F_1} and M_{F_2} can easily be seen by considering the empirical studies. The M_{F_1} delivers better utility for an analysis of individual emissions. On the other hand, M_{F_2} delivers better utility if an analysis contains aggregate computation, e. g., "What is the summation of emissions $(b_i, b_j, b_k, ..., b_n)$

Table 4.3: M_{F_1} Mechanism For Matrix Factorization with Normalization

ϵ	Δb	$\Delta \widehat{a_p}$	Δa_p
0.01	122057.076	337.99	303.475
0.05	63824.391	64.355	62.082
0.1	87070.216	21.647	20.13
0.5	9877.376	5.176	4.901
1	3653.79	3.677	3.538
2	2601.958	1.427	1.271
10	1848.137	0.367	0.375
100	167.92	0.259	0.299

in the distillers grain study?".

In terms of privacy, M_{F_2} achieves a strong privacy as presented in Table 4.2. When ϵ is 1, Δa_p equals 222.4×10^4. When an attacker tries to solve the system with the pseudoinverse approach, the computed $\overline{a}_p$ has a distance of 222.4×10^4 to the original a_p. The attacker is not able to approximate any entries in a_p.

Both M_{F_1} and M_{F_2} ensures strong privacy. This is very positive and convincing findings in the context of LCA publication. The practitioners can feel confident about publishing b. Although the current techniques deliver reasonable utility, the question remains whether there is a way to improve utility without sacrificing the privacy guarantees in such *ill-conditioned* systems.

To answer this inquiry, this study explores a normalization technique to decrease the utility loss in the LCA computation. The motivation for applying normalization is narrowing down the range of numbers that data sets have, since such a wide range causes differentially private systems to inject more noise to the system.

Table 4.3 presents the results when M_{F_1} is executed with the normalized version of a_p, denoted by $\tilde{a}_p$. In this computation, $\tilde{a}_p$ is provided as an input instead of a_p. As seen from the results, using $\tilde{a}_p$ instead of a_p decreases the distance between the published $\widehat{b}$ and b from 5393.87 to 3653.79, since $\widehat{a_p}$ contains less noise compared to the non-normalized computation.

Table 4.4: M_{F_2} Mechanism For Matrix Factorization with Normalization

ϵ	Δb	Δa_p
0.01	1609.384	4.8E8
0.05	339.06	4.0E7
0.1	150.48	4.36E7
0.5	33.738	5.41E6
1	19.73	1.08E6
2	8.489	2.47E6
10	1.392	3.08E5
100	0.164	4.9E4

In addition to that 167 entries of $\widehat{b}$ are approximate enough to the entries of b within a threshold of 10^{-10}. It is 165 if the non-normalized input is used in the computation. Therefore, it is reasonable to state that using the normalized input increases the utility of the M_{F_1} mechanism.

To apply a similar approach to M_{F_2}, the normalization operation is performed on B_x, where the normalized version is denoted by $\tilde{B}_x$. $\tilde{B}_x$ and a_p are inputs to the M_{F_2} mechanism. The results of the publications are presented in Table 4.4.

The normalization approach also has a positive effect on the M_{F_2} mechanism in terms of utility. Since the system is *ill-conditioned*, narrowing down the range of numbers results in adding less noise to the output of the publication. When ϵ equals 1, Δb is 19.73, in contrast to 40.601 when normalization is not employed. This is a huge gain in the utility. However, the normalization technique does not improve the utility of the M_{F_2} mechanism in terms of individual emission analysis. None of the emissions is close enough within the threshold of 10^{-10} to the original emissions.

The normalization does not have any negative impact on the privacy for both M_{F_1} and M_{F_2}. An adversary cannot approximate any element in ap.

Considering the overall empirical study, differentially private LCA computation can be achieved with either M_{F_1} or M_{F_2} without sacrificing security. Although M_{F_1} is useful for an individual emission analysis, M_{F_2} delivers good utility for aggregate analysis on b. The

straightforward application of normalization increases the utility.

4.5 Conclusion

In this chapter, we present a comprehensive study to explore the privacy concerns over publicizing the industrial activities in the form of LCA computations. Accurate and high quality sustainability assessment requires detailed information about industrial activities; however such information is considered confidential. This paper initiates a study to explore privacy and security challenges that prevent organizations from making public disclosures about their activities. Our empirical studies show that the application of privacy-preserving techniques is required to preserve the privacy of private data. Otherwise, it is possible to expose the private data by reverse-computing from the publication. To support the needs of the sustainability research community, this paper proposes differentially private LCA computations and explains how to achieve it for LCA computations by either perturbing the input data or the output data. Our evaluations on a real LCA example from a distillers grain study demonstrates that the use of differential privacy to publish more detailed information ensures strong privacy while revealing useful information for analysts.

Part III

Oblivious Cloud Storage

Chapter 5

TaoStore: Overcoming Asynchronicity in Oblivious Data Storage

Outsourcing data to cloud storage has become increasingly popular and attractive. However, confidentiality concerns [1] make potential users skeptical about joining the cloud. Encryption alone is not sufficient to solve all privacy challenges. Typically, the *access patterns* are *not* hidden from the cloud provider, i.e., it can for example detect 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 e.g. in the setting of searchable symmetric encryption [30, 125].

This work targets *cloud storage* where multiple users from a trusted group (e.g., employees within 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) – a cryptographic primitive originally proposed by Goldreich and Ostrovsky [32] for software protection – is the standard approach to make access patterns *oblivious*. Most ORAM solutions [38, 126–128] are not suitable for our multi-user scenario,

as they handle operation requests *sequentially*, i.e., a new request is not processed until a prior ongoing request is completed, thus creating a bottleneck under concurrent loads. To date, only a handful of solutions leverage parallelism to increase throughput [35,37,38,129]. PrivateFS [35] is based on hierarchical ORAM and supports parallel accesses from a limited number of clients. ObliviStore [37] (which is based on SSS-ORAM [36]) was the first work to consider the proxy model we also assume in this work. ObliviStore was recently revisited by Bindschaedler et al. [38], who proposed a new system called CURIOUS fixing a subtle (yet serious) security flaw arising in concurrent environments.

Our contributions, in a nutshell Motivated by [38], this work initiates a comprehensive study of asynchronicity in oblivious storage. 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 develop a comprehensive security framework, and present an attack showing that *access patterns in CURIOUS are not oblivious in an asynchronous environment* as captured by our model.

2. We design and evaluate a new provably secure system, called TaoStore, that fully resists attacks in asynchronous settings and also leverages the benefits of asynchronicity for better performance. Our system follows a completely different paradigm than previous works – in particular it departs from the SSS framework and is completely tree based – with substantial gains in simplicity, flexibility, and efficiency.

Asynchronicity vs Security

Asynchronicity is an important variable in the design of secure storage systems, and there are at least two ways in which it can affect them:

- *Asynchronous client requests.* Multiple client requests can come at any time point in time (either from the same client or from different ones), and should be answered independently of each other, possibly as soon as the data item is retrieved from the server in order not to slow down the applications requiring these accesses.

- *Asynchronous network communication.* The communication between the clients and the proxy, and the communication between the proxy and the server, is in general asynchronous.

Needless to say, we would like our systems to be secure in such asynchronous environments. The first question we thus ask is:

> ***Are existing systems secure under arbitrary scheduling of communication and operations?***

The answer is negative: *all* existing approaches of handling concurrent requests on the *same* data item can leak substantial information under asynchronous scheduling. The authors of CURIOUS [38] have already shown that the sequentialization of accesses to the same block in ObliviStore renders the system insecure. We will go one step further, and show that CURIOUS itself has not completely resolved the issue, and is also insecure when operations are scheduled concurrently and communication is asynchronous.

Our attack assumes that the adversary learns the timings of the proxy's answers back to the client. We find this assumption reasonable. For example, the attacker may observe (encrypted) network traffic between the proxy and the clients, and moreover, a client may only schedule a new access (or perform some other noticeable action) when a previous access terminates. These timings were however kept secret in the original security definition of [37], also used in [38]. Therefore, our attack does not invalidate any of the claims from [37]. Still, it motivates us to develop a definitional security framework for asynchronous oblivious storage systems, which we believe to be of independent interest.

Asynchronicity vs Efficiency

Our security assessment calls for a system which is fully secure in an asynchronous environment. Instead of simply fixing existing approaches (e.g., CURIOUS), we first take the chance to address the following question:

> ***How well do existing systems leverage parallelism to handle concurrent asynchronous requests?***

Indeed, existing systems have some undesirable features. CURIOUS relies on data partitioning, and accesses to the same *partition* are sequentialized. In contrast, here, we would like to develop a system which is "natively" concurrent – we would like our system to achieve high throughput even when using a single partition. ObliviStore achieves higher concurrency on individual partitions, yet, as pointed out in [38], the system relies on a fairly complex background shuffling process which is responsible for writing data back to the server and which significantly affects performance of the system.

TaoStore Motivated by the above concerns, we develop and evaluate TaoStore, a fully-concurrent provably secure multi-user oblivious data store. TaoStore departs from the traditional partition-based SSS approach [36] used in current systems. Instead, it relies on a *tree based* ORAM scheme aimed at fully concurrent data access. Tree-based ORAMs 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 scheme features a novel approach to manage *multiple* paths fetched concurrently from the server. In particular, the write back of updated path contents to the server occurs in an entirely *non-blocking* way, i.e., new paths *overlapping* with paths being written back can still be retrieved and updated *while* the write back operation is under way.

TaoStore is substantially simpler than ObliviStore and enables better concurrency than CU-

RIOUS. We can in particular dispense with running the expensive background shuffling process from the former, and different from the latter, operations can be executed concurrently even on individual partitions.

Security and correctness We prove the ORAM scheme underlying TaoStore secure using our new security framework, which guarantees security against adversaries which can schedule both operations and network messages. In particular, a key contribution of our construction is the introduction of a *sequencer* module aimed at preventing our attacks affecting other systems. 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. Our concurrency handling calls for a rigorous proof of correctness, which was not necessary in previous systems due to simpler approaches to accessing shared objects.

Evaluation We present two different evaluations of TaoStore: (1) A local evaluation (with the same experimental setup as in [37]) 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. That is, the frequency of write backs does not substantially slow down the system.

We emphasize that we *do not* implement recent bandwidth-reducing techniques using server-side computation [130–132] – we *explicitly* target usage on a simple storage server which only allows for read-write access and no computation (except for basic time-stamping), and these newer schemes – while extremely promising – are not relevant for our setting.

Partitioning Previous works use data partitioning in a fundamental way. In particular, CURIOUS [38] relies on data partitioning to ensure concurrency (access to the same partition are sequentialized). TaoStore does not rely on partitioning – indeed, the performance of our system is competitive even without it – yet there are scenarios where partitioning is desirable, as it can help overcome storage, bandwidth, and I/O limitations. If desired, our tree-based approach enables partitioning as a simple add-in – one breaks up the tree into a forest of sub-trees, maintaining the tree-top in the proxy.

Overview of TaoStore

Developing an ORAM scheme for a concurrent setting is indeed far from obvious. To see why this is the case, we first review the main ideas behind tree-based ORAM schemes, such as Path ORAM by Stefanov et al. [126].

These schemes have their storage space organized as a tree, with each node containing a certain number of (encrypted) blocks. A single client keeps a position map mapping each (real) block address to a path from the root to a leaf in the tree, together with some local memory containing a (usually small) number of overflowing blocks, called the *stash*. To achieve correctness, the ORAM client maintains a *block-path invariant* ensuring that at each point in time, a block is either on its assigned path *or* in the stash. Under this invariant, processing each access (either read or write) for a block involves three operations—*read-path, flushing and write-back*. First, the ORAM client fetches the path P assigned to the block, and uses it together with the stash to answer the request. To maintain obliviousness, the block is immediately assigned to a new random path in the position map, so that a future access for the same block would fetch an independent random path (hiding repetition in accesses). Next, the contents of the path P and stash are re-arranged so that every block ends up at the lowest possible node on P and also on its assigned path; only blocks that do not fit remain in the stash. This re-arrangement

is referred to as *flushing* and is crucial for ensuring that the stash never "overflows". Finally, a re-encrypted version of P is written back to the server, keeping the server up-to-date.

How do we make Path ORAM concurrent and asynchronous, while retaining both security and correctness? Even after a first glance, several issues immediately arise. First off, multiple paths may need to be retrieved simultaneously, as one request may be made while a path is being retrieved from the server – however, what if the requests are for the same item? Second, every path needs to be written back to the server, but what if just after the contents of a path have been sent back to the server, one of the items contained in this path needs to be updated? Finally, if the attacker can observe when the clients receive responses to their requests, how does one ensure that the timing of these responses are oblivious? All of this must be considered in a truly asynchronous setting, where we do not want to make any timing assumptions on the communication between the proxy and the server.

Our ORAM scheme – TaORAM – resembles Path ORAM, but allows multiple paths to be retrieved *concurrently*, without waiting for on-going flush and write-back operations to complete. All operations are done asynchronously:

- At the arrival of a request for a certain block, the appropriate read-path request is sent immediately to the server.
- 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.
- 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).

Here, we highlight the fundamentals of our approach, and how we address the challenges outlined above; see Section 5.3 for more details.

Consider 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 concurrent requests for the same block (via a data structure called request map) so that at each point, only one request triggers reading the actual assigned path, whereas all others trigger fake reads for a random path.

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. Several technical challenges need to be addressed to maintain the invariant, as the local subtree and the server are constantly concurrently updated, and read-path and write-back operations are completely asynchronous.

The stash size analysis of Path ORAM breaks down when operations are asynchronous. Nevertheless, we show that the stash size of TaORAM running with a sequence of requests is the same as that of Path ORAM running with a different but related sequence of requests, which is permuted from the actual sequence according to the timing of flushing.

Further background and related works

It is impossible to cover the huge body of previous works on ORAM, and its applications. We have already discussed works implementing multi-client systems and in particular Oblivi-

Store and PrivateFS – here, we give a short overview of other works.

Hierarchical ORAMs *Hierarchical* ORAMs were first proposed by Goldreich and Ostrovsky [32] (referred to as the GO-ORAM henceforth), to store N elements. Hierarchical ORAMs organize the memory in $\log N$ many levels, consisting of increasingly many 2^i buckets. At any time point, each logical block is assigned to one random bucket per level, and stored in exactly one of them. Hierarchical ORAMs require a regular shuffling operation to deal with overflowing levels after oblivious re-insertion of items into the hierarchical data structure. Subsequent hierarchical ORAMs improve different aspects of GO-ORAM, such as reduced overhead [133–136, 136, 137], faster shuffling [134, 136, 138, 139], and de-amortizing shuffling [33, 35, 134, 140, 141].

Tree ORAMs Tree ORAMs have been proposed relatively recently, first by Shi et al. [142] and then soon extended in a number of works [126–128, 143]. The current state-of-the-art construction is Path ORAM [126] which was briefly reviewed above and will be reviewed in detail below. Other tree ORAMs share the same overall structure but differ in important details, for instance, the absence of stash in [127, 142, 143], varying heights and degrees of the tree in [128, 143], applying flushing on randomly chosen paths in [127, 128], or on paths in a fixed deterministic order [130, 143], reducing the frequency of flushing and changing the tree bucket structure [130], varying the size of the blocks [126, 144], and achieving constant communication size by moving computation to the server [131, 132].

Recent practical constructions In the past several years, many practical ORAM schemes have been constructed and implemented for real-world applications, like secure (co-)processor prototypes [34, 145–147] and secure cloud storage systems [33–37, 129, 148]. While classical ORAM schemes with small client memory apply directly to the former setting, in cloud applications where a client wishes to outsource the storage of a large dataset to a remote server and

later access it in an oblivious way, the client typically has more storage space, capable of storing $O(\sqrt{N})$ blocks or even some per-block meta-data of total size $O(N \log N)$. The availability of large client storage enables significantly reducing the computation overhead of ORAM to $O(\log N)$ [35–37, 136, 137, 149, 150], and furthermore, reduces the number of client-server interactions per access to $O(1)$ (instead of $O(\log N)$).

Other works on multi-client ORAM A problem superficially related to ours (but technically different), is that of Oblivious Parallel RAM (OPRAM), recently introduced by Boyle, Chung, and Pass [151]. Even though Path ORAM-like OPRAM schemes have also been proposed [152], OPRAM clients coordinate their access to the server *without* a proxy. To achieve this, they can communicate *synchronously* with each other. The resulting schemes are however fairly unpractical.

A recent work by Maffei et al. [153] also considers ORAM in conjunction with multi-user access, developing a new primitive called *Group ORAM*. Their work considers a scenario where a data owner enforces access-control restrictions on data, whereas we consider a common address space which can be accessed by a group of mutually-trusting users. The efficiency of their solution compares to that of single-client, sequential, ORAM schemes (like Path ORAM), and they do not address efficient, high-throughput, concurrent access, which is the focus of our work.

5.1 Asynchronous ORAM Schemes: Definitions and Attacks

This section addresses the security of ORAM schemes in asynchronous settings. We give both a formal security model, and attacks against existing implementations.

5.1.1 Security Model

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. Note that the definition of [37] (which is also used in [38]) *does* consider asychronicity, but it is inherently *non-adaptive* and, even more importantly, does not reveal response times.

We give an informal (yet self-contained) overview of the definition – further formal details are deferred to Appendix A.1. We stress that we do not differentiate, at the formal level, between multi- and single-client scenarios – an ORAM scheme is what is run by the proxy in our application scenario, but we think more generally this of it as a single "client" answering asynchronous requests. Whether these come from multiple parties or not is orthogonal to our treatment.

ORAM Schemes We think of an asynchronous ORAM scheme as a pair $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$, where Encode takes an initial data set D of N items with a certain block size B, and produces an encrypted version $\hat{D}$ to initialize an *untrusted storage* sever SS, together with a corresponding secret key K. In particular, SS gives basic read/write access to a client accessing it, together with timestamping, i.e., writing a new item in some location on SS overwrites the current item only if the timestamp of the new item is larger. $\mathsf{OClient}$ is the actual (stateful) client algorithm which is given K, and can be invoked at any time with requests for read-/write operations, and eventually answers these requests, after interacting with SS. Concretely, $\mathsf{OClient}$ processes ***read requests*** for a certain block address $\mathsf{bid} \in [N]$ to retrieve the value stored in this block, and ***write requests*** to overwrite the value of a certain block bid (and possi-

bly retrieve the old value). These requests are denoted as $(\mathsf{op}, \mathsf{bid}, v)$ where $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$ and $v = \perp$ when $\mathsf{op} = \mathsf{read}$. Every such request is *terminated* at the point in time by either returning the retrieved value or (for write operations) simply an acknowledgement to the caller, and possibly the value which was overwritten.

Security definition We now proceed with our definition of aaob security, which is an indistinguishability based security notion. Given an attacker $\mathcal{A}$ and an ORAM scheme $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$, we consider an experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ where OClient accesses a storage server SS via an asynchronous link. The experiment initially samples a random challenge bit $b \xleftarrow{\$} \{0, 1\}$, and then proceeds as follows:

- The attacker $\mathcal{A}$ initially chooses two equally large data sets D_0, D_1. Then, the game runs $(\hat{D}_b, K) \xleftarrow{\$} \mathsf{Encode}(D_b)$. As a result, $\hat{D}_b$ is stored on SS, and the key K is given to OClient.

- The attacker $\mathcal{A}$ can, at any point in time, invoke 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 OClient. When the operation completes, the adversary $\mathcal{A}$ is notified, yet it is not told the *actual* value returned by this operation.[1]

- When processing operation requests, OClient communicates with SS over a channel whose scheduling is controlled by $\mathcal{A}$. Concretely, when ORAM sends a read or write request to SS, $\mathcal{A}$ is notified (and given the message contents), and $\mathcal{A}$ can decide to deliver this message to SS at any point in time. Similarly, $\mathcal{A}$ controls the scheduling of the messages sent back from SS to ORAM, and also learns their contents. There are no ordering constraints – $\mathcal{A}$ can deliver messages completely out of order, and even drop messages.

[1] This restriction is necessary, for otherwise an adversary $\mathcal{A}$ could easily guess the value of b.

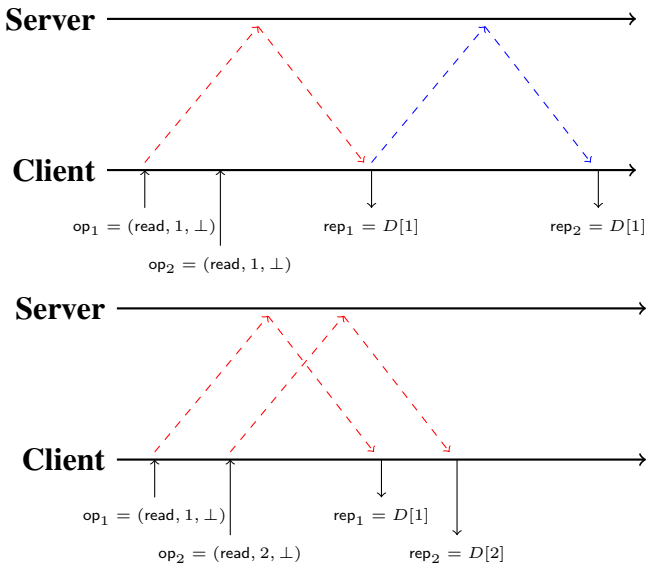


Figure 5.1: **Attack against ObliviStore.** Comparison of event timing for repeated access (above) and distinct accesses (below). Here, we assume constant delays in delivering messages.

- Finally, when the adversary $\mathcal{A}$ is ready, it outputs a guess b' for b, and the experiment terminates. In particular, if $b = b'$, we way that the experiments outputs `true`, and otherwise it outputs `false`.

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

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

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).

Remarks One key point of our definition is that the adversary learns the response times – this was not the case in [37]. This information is crucial, and in particular it is very hard to argue an adversary has no access to it. Not only in our deployment scenario this information is visible by a potential network intruder (the actual ORAM client is run by a proxy with net-

work connectivity to its users), but also ORAM users will most likely have different behaviors triggered by these responses.

We also note that (out of formal necessity) we do not leak the *contents* of operation responses, and only their timing. Otherwise, $\mathcal{A}$ can easily recover the challenge bit b. In the full version, we discuss stronger simulation-based security notions allowing this information to be revealed.

Correctness The above discussion did not address the issue of correctness of the scheme, which is quite subtle given the concurrent nature of the system. Following the classical literature on distributed systems, Appendix A.3 defines *atomic semantics* for an asynchronous ORAM scheme as our target correctness notion. This in particular means that operations appear to take place atomically at some point between their invocation and their response.

5.1.2 Attacks

We present two attacks – one against ObliviStore, one against CURIOUS – breaking their aaob-security. We note that the former attack is just a re-iteration of the key idea presented in [38]. In contrast, our second attack is novel. We give a high-level explanation of the attacks, but a formalization in our framework (given an appropriate formalization of the scheme) can be obtained easily.

Attack against ObliviStore An attack against ObliviStore can be derived from the weakness already observed in [38]. In particular, ObliviStore sequentializes accesses on the same item, and thus an adversary requesting the same item *twice* (e.g., issuing two subsequent requests $\mathsf{op}_{1,0} = \mathsf{op}_{2,0} = (\mathsf{read}, 1, \perp)$) will see only one request being made to the storage server, with a second request being scheduled only *after* the response to the first one returns to the client. In contrast, scheduling requests $\mathsf{op}_{1,1} = (\mathsf{read}, 1, \perp)$ and $\mathsf{op}_{2,1} = (\mathsf{read}, 2, \perp)$ for two different

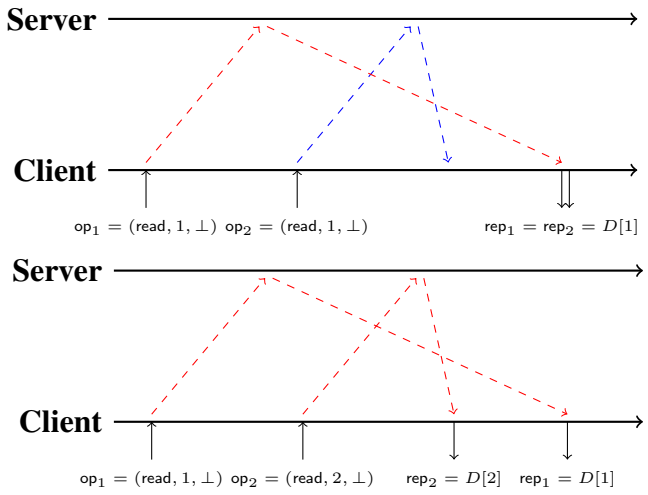


Figure 5.2: **Attack against CURIOUS's fake-read logic:** The upper figure represents the timing of the communication between the client and the server when accessing the same item twice, and the second access is a "fake read" (in blue). The figure below represents the execution when the accesses are for two distinct items (both "real reads"). The timings of the responses differ, as in the above case, the client needs to wait for the actual value to arrive.

addresses will have the adversary see the client immediately schedule two requests to retrieve information from the server. This leads to easy distinguishing. Figure 5.1 gives two diagrams presenting the two situations in detail.

We note two things. First off, this attack breaks ObliviStore even in the model in which it was claimed to be secure, as response times are not needed to distinguish between the repeated-access scenario. Also, the attack does not require the network to be asynchronous – only the ability to schedule overlapping operations. Second, if response times can be measured, then the attack is very easy to mount: An independent experimental validation (with the ObliviStore implementation provided to us) shows that repeatedly accessing the same item over and over leads to a performance degradation of up to 50% compared to accessing well-spread loads.

Attack against CURIOUS The overcome this, [38] suggested an alternative approach based on the idea that a concurrent operation on the same item should trigger a "fake read". We show

that this idea, by itself, is not sufficient to achieve aaob-security. We note that our attack does not contradict security claims in [38], since the model of [37] is used, which does not leak the timing of responses. (As argued above, we believe that it is extremely hard to hide these timings in actual deployment.)

To start with, recall that when two concurrent requests for the same item are made in CURIOUS (think of these as read requests for simplicity), 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 (a so-called "fake read") to hide the repeated access. This "fake read" looks like an access to an unrelated, independent item (the details are irrelevant).

The key issue – ultimately allowing us to distinguish – concerns the *timings* of the responses given by the ORAM client. When the fake read operation terminates (i.e., the corresponding data is received by the ORAM client from the server), the client always returns the item fetched in the real read *if it is available*. If the item is not available, then it needs to wait for the real read to terminate. Note that in the asynchronous setting, the latter situation *can* occur – we have no guarantee whatsoever that the real read terminates before the fake read.[2] This is in contrast to the case where the reads are for two distinct items (and hence both "real"), and the second request can be answered right away even if the client has not received the data from the server associated with the second request.

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 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

[2] CURIOUS in fact envisions the fake read going with high probability to a partition different than the real read – this partition may even be on a different machine, and thus out-of-order responses are quite likely.

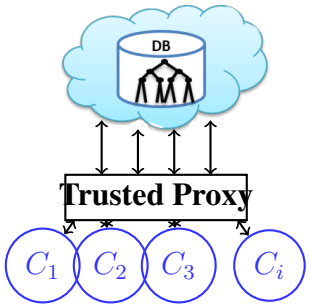


Figure 5.3: Deployment model of TaoStore

back to OClient immediately. If this results in an answer to the second operation being triggered immediately, the attacker guesses $b = 1$, otherwise it guesses $b = 0$. The outcome of the attack is depicted in Figure 5.2.

Remarks We note that to prevent the same attack affecting CURIOUS, our system TaoStore will introduce the notion of an *operation sequencer*, a module catching out-of-order early replies from the ORAM client back to the caller, for instance by ensuring that in our attack scenario from above, *also* in the setting with two real reads, the final response to the second real read will not be sent before the response to the first real read. In other words, we will not happen to modify the fake-read logic. Rather, we make sure that real reads have response timings consistent with the behavior one would observe if some of these are fake.

5.2 Overview of TaoStore

This section provides a high-level overview of TaoStore and its goals, including the deployment scenario and architecture of our system.

High-level goal The goal of TaoStore is to allow multiple clients (or users) to securely and obliviously access their shared data on an untrusted storage server (a "public cloud"). Infor-

mally, the security guarantee is that the contents of the shared data and of the accesses from the multiple clients are kept hidden against any honest-but-curious entity[3] observing traffic to and from the server and being able to schedule messages. This is formalized via the notion of aaob security introduced above.

Concretely, users issue *read requests* for a certain block address bid to retrieve the value stored in this block, and *write requests* to overwrite the value of a certain block bid (and possibly retrieve the old value). These requests are denoted as $(\mathsf{type}, \mathsf{bid}, v)$ where $\mathsf{type} \in \{\mathsf{read}, \mathsf{write}\}$ and $v = \perp$ when $\mathsf{type} = \mathsf{read}$. The block address bid belongs to some logical address space $\{1, \ldots, N\}$, and blocks have some fixed size B. (In our system, $B = 4$ KB.) Every such request is *invoked* at some point in time by a client process, and *terminates* at the point in time by either returning the retrieved value or (for write operations) simply an acknowledgement to the caller.

System architecture As in previous works [37, 38], TaoStore relies on a *trusted proxy*, who acts as a middle layer between users and the untrusted storage. (See Figure 5.3 for an illustration of the architecture.) The proxy coordinates accesses from multiple users to the untrusted storage, which it makes oblivious, and stores locally secret key material used to encrypt and decrypt the data stored in the cloud. We also assume that the communication between users and the proxy is protected by end-to-end encryption. This is often referred to as the "hybrid cloud" model [37].

TaoStore's proxy will effectively run the Oblivious RAM scheme, TaORAM (briefly discussed above in the introduction and presented below in Section 5.3), which is particularly well suited at processing requests in a highly concurrent way, as opposed to traditional ORAM schemes which would force request processing to be entirely sequential.[4] We assume that net-

[3]While not addressed in this work, enhancing security to an actively malicious server can be achieved via fairly standard techniques.

[4]The number of clients is irrelevant for our system, as all clients are allowed to access the same data and each

work communication, most importantly between the proxy and the untrusted storage, is completely *asynchronous*. Furthermore, in contrast to classical applications, the ORAM scheme here can effectively use large memory on the proxy, even up to $N \log N$ (e.g., to store a full position map). (Large proxy memory was also exploited in ObliviStore already.)

5.3 Our Asynchronous ORAM

In this section, we present the asynchronous ORAM scheme underlying TaoStore – which we refer to as TaORAM. In particular, TaORAM is run by the trusted proxy, which acts as the "single client" interacting with the storage server, handling queries concurrently. Therefore, in the following, we refer to the entity running the ORAM algorithm (the trusted proxy here) as the *ORAM client*.

TaORAM is based on the non-recursive version of Path ORAM, but processes client requests *concurrently and asynchronously*. We focus on the non-recursive version, since in our deployment model the trusted proxy has reasonably large memory, able to hold some meta-data for each data block. (The same recursive technique as in Path ORAM can be applied to reduce the memory overhead if needed.) Below, we first briefly review Path ORAM, and then describe TaORAM.

5.3.1 A Review of Path ORAM

To implement a (logical) storage space for N *data blocks* (stored in encrypted form) the basic Path ORAM scheme organizes the storage space virtually as a complete binary tree with at least N leaves, where each node of the tree is a small storage bucket that fits $Z = 4$ data blocks. To hide the logical access pattern, each data block is assigned to a random path pid

client can issue multiple queries concurrently, and thus effectively an arbitrary number of clients can be seen as one single client accessing the proxy without loss of generality.

from the root to the leaf (so we can equivalently think of pid as being the identifier of a leaf, or of such a path) and stored at some node on this path; the assignment is "refreshed" after each access for this block (either for a read or for a write operation) to a new random path pid' to hide future accesses to the same block. The ORAM client keeps track of the current assignment of paths to blocks using a *position map*, pos.map, of size $O(N \log N)$ bits,[5] overflowing blocks (see below) in an additional data structure, called the *stash*, and denoted stash, of fixed a-priori bounded size (the size can be set to some function of the order $\omega(\log N)$, even only slightly super-logarithmic).

For each client request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ with $\mathsf{type}_i = \mathsf{read}/\mathsf{write}$, Path ORAM performs the following operations:

1. **Request Processing (Read-Path):** Upon receiving the request, Path ORAM sends a read request to the server for the path $\mathsf{pid} = \mathsf{pos.map}[\mathsf{bid}]$ assigned to block bid. When the path is retrieved, it decrypts the path and finds block bid on the path or in stash, and either returns its value if $\mathsf{type}_i = \mathsf{read}$, or updates it to v_i if $\mathsf{type}_i = \mathsf{write}$. Path ORAM then assigns block bid to a new random path pid' and updates $\mathsf{pos.map}[\mathsf{bid}] = \mathsf{pid}'$ accordingly.

2. **Flushing:** In a second phase, it iterates over each block bid on the path pid or in the stash, and inserts it into the lowest non-full node (i.e., containing less than Z nodes) on pid that intersects with its assigned path $\mathsf{pos.map}[\mathsf{bid}]$. If no such node is found, the block is placed into the stash.

3. **Writing-back:** Then Path ORAM encrypts the path with fresh randomness, and writes path pid back to the server.

Initializing the remote storage. To initialize the contents of the remote storage server, the ORAM client can simply run the ORAM algorithm locally, inserting elements one by one.

[5] The full Path ORAM scheme recursively outsources the position map to the server to reduce the ORAM client's local storage to $\mathrm{poly}\log(N)$.

The resulting storage tree can be safely sent to the server to store and accessed later. Since this approach can be applied universally to any ORAM scheme, we omit a discussion on encoding the initial data set below.

5.3.2 TaORAM

TaORAM internally runs two modules, the *Processor* and the *Sequencer*. (See Figure 5.4 for an illustration.) The Processor interacts with the server, prepares answers to all logical requests, and returns answers to the Sequencer. The Sequencer merely forwards logical requests to the Processor, and when receiving the answers, enforces that they are returned in the same order as the requests arrive, as we explain in more detail below.

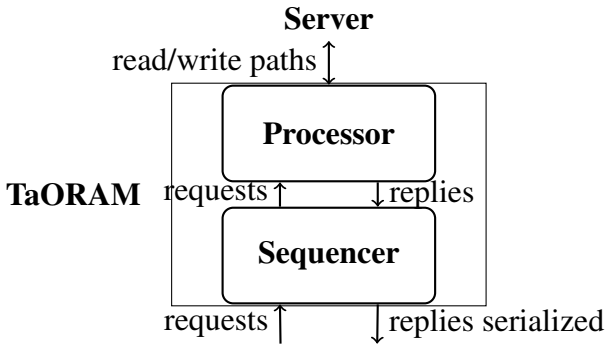


Figure 5.4: TaORAM Structure

We present TaORAM in steps. Step 1-3 describe the design of the Processor, each step enabling a higher degree of concurrency. In this description, when obliviousness is concerned, it is convenient to focus only on the communication between the Processor and the server. Then, in Step 4, we show how to prevent additional information leakage through the timing of replies, and in particular explain the functionality of the Sequencer. A complete pseudocode description of TaORAM is provided in Figures 5.5, 5.6, and 5.7.

Module Sequencer:

Global Data: A sequencer.queue and a sequencer.map.
Sequencer reacts to the following events:

- Upon receiving request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$, do:
  - Create entry sequencer.map$[(\mathsf{type}_i, \mathsf{bid}_i, v_i)] \leftarrow \perp$.
  - Push request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ into sequencer.queue.
  - Send request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ to **Processor**.
- Upon receiving response w_i for request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ from **Processor**, set sequencer.map$[(\mathsf{type}_i, \mathsf{bid}_i, v_i)] \leftarrow w_i$.
- Run on a separate thread the Serialization Procedure that keeps doing the following:
  - When sequencer.queue is non-empty, pop a request $(\mathsf{type}, \mathsf{bid}, v)$ from sequencer.queue.
  - Wait until entry sequencer.map$[(\mathsf{type}, \mathsf{bid}, v)]$ is updated to a value $w \neq \perp$.
  - Return w as a response to request $(\mathsf{type}, \mathsf{bid}, v)$, and remove entry sequencer.map$[(\mathsf{type}, \mathsf{bid}, v)]$.

Module Processor:

Global Data: A secret (encryption) key key, a stash, a request.map, a response.map, a PathReqMultiSet, a subtree, a counter #paths and a write.queue.
Processor reacts to the following events:

- Upon receiving a logical request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ from **Sequencer**, start a new thread doing the following and then terminate.
  - (pid, P, fake.read) $\leftarrow$ READ-PATH($\mathsf{type}_i, \mathsf{bid}_i, v_i$);
  - *Lock* subtree;
  - ANSWER-REQUEST(type_i, bid_i, v_i, pid, P, fake.read);
  - FLUSH(pid);
  - *Unlock* subtree;
- Whenever #paths turns a multiple of k, $c \cdot k$, start a new thread running WRITE-BACK(c);

Figure 5.5: Pseudocode description of TaORAM.

READ-PATH($\mathsf{type}_i, \mathsf{bid}_i, v_i$):

1. Create entry response.map$[(\mathsf{type}_i, \mathsf{bid}_i, v_i)] \leftarrow (\mathsf{false}, \perp)$.
2. Insert $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ into queue request.map$[\mathsf{bid}_i]$.
   - If the queue was previously empty, set fake.read $\leftarrow 0$ and pid $\leftarrow$ pos.map$[\mathsf{bid}_i]$;
   - Else, set fake.read $\leftarrow 1$, and sample pid $\xleftarrow{\$} \{0,1\}^D$.
3. Read-path pid from server and insert pid to PathReqMultiSet. Wait for response.
4. Upon waking up with the server response, remove (one occurrence of) pid from PathReqMultiSet.
5. Decrypt the response with key to obtain the content of path pid, denoted as P, and return (pid, P, fake.read).

ANSWER-REQUEST($\mathsf{type}_i, \mathsf{bid}_i, v_i$, pid, P, fake.read):

1. Syncing procedure: Insert every node w on path P that is currently not in subtree into subtree.
2. Update entry response.map$[(\mathsf{type}_i, \mathsf{bid}_i, v_i)]$ from (b, x) to (true, x). If $x \neq \perp$, reply value x for the request $(\mathsf{type}_i, \mathsf{bid}_i, v_i)$ to **Sequencer**, and delete the entry.
3. If fake.read $= 0$, find block bid_i in subtree, and create responses to requests in queue request.map$[\mathsf{bid}_i]$ as follows:
   - Pop a request $(\mathsf{type}, \mathsf{bid}_i, v)$ from the queue.
   - Let w be the current value of block bid_i.
   - If type $=$ write, set the value of bid_i to v.
   - If entry response.map$[(\mathsf{type}, \mathsf{bid}_i, v)] = (\mathsf{true}, \perp)$, reply value w for the request $(\mathsf{type}, \mathsf{bid}_i, v)$ to **Sequencer**, and delete the entry.
   - Else, if response.map$[(\mathsf{type}, \mathsf{bid}_i, v)] = (\mathsf{false}, \perp)$, set the entry to (false, w).

   Repeat the above steps until request.map$[\mathsf{bid}_i]$ is empty.
4. If fake.read $= 0$, assign block bid_i a new random path pos.map$[\mathsf{bid}_i] \xleftarrow{\$} \{0,1\}^D$.

Figure 5.6: Pseudocode description of TaORAM.

FLUSH(pid):

1. For every block bid' on path pid in subtree and stash, do:
   - Push block bid' to the lowest node in the intersection of path pid and $\mathsf{pos.map}[\mathsf{bid}']$ that has less than Z blocks in it. If no such node exists, keep block bid' in stash.
2. Increment $\#\mathsf{paths}$ and push pid into queue write.queue.
3. For every node that has been updated, add (local) timestamp $t = \#\mathsf{paths}$.

WRITE-BACK(c):

1. Pop out k paths $\mathsf{pid}_1, \cdots \mathsf{pid}_k$ from write.queue.
2. Copy these k paths in subtree to a temporary space S.
3. Encrypt paths in S using secret key key.
4. Write-back the encrypted paths in S to the server with (server) timestamp c. Wait for response.
5. Upon waking up with write confirmation, delete nodes in subtree that are on paths $\mathsf{pid}_1, \cdots \mathsf{pid}_k$, with (local) timestamp smaller than or equal to $c \cdot k$, and are *not* on any path in PathReqMultiSet.

Figure 5.7: Pseudocode description of TaORAM.

Step 1 – Partially Concurrent Requests

For any $k \geq 1$, Path ORAM can naturally be adapted to support partial "k-way" concurrent processing of logical requests when the k logical requests are *non-repetitive* (i.e., accessing distinct blocks).[6] In this case, the Processor implement a variant of Path ORAM to first (1') simultaneously fetch k paths from the server to find the requested blocks, and store all paths in local memory, forming a subtree we refer to as subtree; after assigning these k blocks to k new random paths, (2') it flushes along the subtree, and (3') writes back the entire subtree to the server. Note that since the server is not updated during step (1'), the read-path requests for the k

[6] A similar observation was made for hierarchical ORAMs in the design of PrivateFS [35], which supports partial concurrent processing of requests from multiple clients.

logical requests can be issued concurrently and asynchronously, without further coordination. Furthermore, when logical requests are for distinct blocks, the k paths fetched in step (1') are independent and random, and this ensures obliviousness.

However, when there are repetitive logical requests, obliviousness no longer holds. This is because multiple accesses to the same block cause the Processor to fetch the same path multiple times, leaking the existence of repetition. To solve this issue, TaORAM maintains a **request map**, denoted as request.map, that maps each block bid to a queue, request.map[bid], of (unanswered) logical requests for this block. To avoid leaking repetitions, only the *first* logical request in the queue triggers reading the actual assigned path—termed a "real read", whereas all following requests trigger reading a random path—termed a "fake read". Later, when the assigned path is retrieved, responses to all requests in request.map[bid] are created in sequence to ensure logical consistency. (See Step 2 in algorithm READ-PATH and Step 3 in algorithm ANSWER-REQUEST in Figure 5.6.)

Step 2 – Fully Concurrent Request Processing

In the above scheme, flush and write-back operations (i.e., Step 2' and 3') implicitly "block" the processing new requests, imposing an undesirable slow down. In the following, we enhance the Processor to enable *fully* concurrent processing: Each incoming request is immediately inserted into the request map and the appropriate path is fetched from the server, *even if flushing and writing back of previously retrieved paths are in progress.*

Such modification brings a number of challenges for ensuring correctness. For example, before a write-back operation is completed, part of the contents on the server are potentially stale, and hence reading a path from the server at the same time may lead to an incorrect answer to some logical request. To ensure correctness, TaORAM will maintain the following,

Fresh-Subtree Invariant: *The blocks in the local* subtree *and* stash *are always*

up-to-date, whereas the tree at the server contains the most up-to-date contents for the remaining blocks.

The invariant is strongly coupled with our subtree **synching procedure**: Whenever the Processor retrieves a path from the server, it discards the part that intersects with the local subtree, and only inserts the rest of the nodes into subtree. Under the fresh-subtree invariant, after "synching", the path in subtree is guaranteed to be up-to-date, and can safely be used to answer logical requests. (See Step 1 of algorithm ANSWER-REQUEST in Figure 5.6.)

Maintaining the invariant is, however, subtle, and one of the core technical challenges in our algorithm. If nodes in subtree were never deleted, the invariant would be trivially maintained, as all updates are first performed on subtree. But, this eventually leads to a huge subtree. Therefore, whenever the server confirms that some k paths has been written back, the Processor deletes some nodes from subtree.

Unfortunately, naively deleting the entire k paths would violate the fresh-subtree invariant. This is because between the time t_1 when the write-back operation starts and t_2 when it completes (receiving confirmation from the server), the subtree is potentially updated. Hence, at t_2, the Processor must keep all nodes updated after t_1, or else new contents would be lost. Another issue is that between t_1 and t_2, new logical requests may trigger reading a path pid from the server; to ensure that when the path is retrieved (after t_2), it can be correctly "synched" with subtree, the Processor must keep all nodes on path pid (for the content retrieved from the server may be stale since the path is requested before t_2).

In summary, the Processor must not delete any nodes that have been more recently (than t_1) updated or requested. To ensure the former, we timestamp every node in subtree (locally) to record when it is last updated. (See Step 3 of Algorithm FLUSH in Figure 5.7, and note that this timestamp is different from the version number used as a server timestamp.) To ensure the latter, the Processor maintains a multi-set PathReqMultiSet that tracks the set of paths

requested but not yet returned.[7] (See Step 3 and 4 of algorithm READ-PATH in Figure 5.6 and Step 6 of WRITE-BACK in Figure 5.7.)

Step 3 – Non-Blocking Flushing

So far, though requests are concurrently processed at their arrival, the flush and write-back operations are still done sequentially, in the same order their corresponding logical requests arrive (in batches of k). We further remove this synchronization.

First, we decouple the order in which paths are flushed from the order in which logical requests arrive: As soon as a path is retrieved ("synched" with the subtree, and used for answering client request), the Processor flushes the path immediately, even if the paths for some previous requests have not yet been returned (remember that they could well be late due to the asynchronous nature of the network). Furthermore, we make write-back operations asynchronous: As soon as k new paths are inserted into subtree and flushed, the Processor writes-back these k paths to the server, irrespective of the status of any other operations (e.g., some previous write-back requests may still be pending)— therefore, in the rest of the chapter, we call k the *write-back threshold*. In summary, flush and write-back operations are performed as soon as they are *ready* to be performed. (See the pseudocode of Module Processor.)

This brings two challenges. First, since paths may be flushed in an order different from that they were requested, it is no longer clear whether the stash size is bounded (at least the analysis of Path ORAM does not directly apply as a black box). We show that this is indeed the case, and provide the proof below.

Lemma 5.3.1 *The stash size of TaORAM is bounded by any function $R(N) = \omega(\log N)$ (e.g. $R(N) = (\log N) \cdot (\log \log \log N)$), except with negligible probability in N.*[8]

[7] We remark that PathReqMultiSet must be necessarily a multi-set, as the same path may be requested more than once.

[8] In fact, the statement can be made more concrete, as the probability of overflowing is roughly c^{-R} for some constant c and stash size R.

The second challenge is ensuring server consistency when multiple write-back operations end up being concurrent. In an asynchronous network, these requests may arrive at the server out-of-order, causing the server to be updated incorrectly. To address this problem, we mark each node stored at the server, as well as each write-back request, with a version number (or "server timestamp"), and the server can only overwrite a node if the write-back request is of a newer version. (See Step 4 of WRITE-BACK; we omit the server algorithm due to lack of space.)

Proof: [Proof of Lemma 5.3.1] We only give a proof sketch. A more formal proof is rather tedious and requires repeating many of the technical steps in the stash analysis of Path ORAM with little change.

We show that given any execution trace T of TaORAM with a sequence of logical requests $r_1, r_2, \cdots$, one could come up with another sequence $r'_1, r'_2, \cdots$ of the same length (modified and permuted from the original sequence based on the execution trace) which when fed to Path ORAM sequentially yields the same stash.

By design of TaORAM, whenever the Processor receives a request $r_i = (\mathsf{type}_i, \mathsf{bid}_i, v_i)$ with $\mathsf{type}_i = \mathsf{read}/\mathsf{write}$, it immediately issues a path-read request to the server, fetching either the path $\ell_i = \mathsf{pos.map}(\mathsf{bid}_i)$ assigned to block bid_i (in the case of real read), or a randomly chosen path $\ell_i \xleftarrow{\$} U$ (in the case of fake read). Furthermore, upon receiving the path ℓ_j corresponding to request r_j from the server, the Processor flushes the path immediately. The execution trace T contains the time t_j at which each path ℓ_j corresponding to request r_j is flushed. Order the time points chronologically $t_{j_1} < t_{j_2} < \cdots$. We observe that the contents of the stash are determined by the sequence of events of flushing over paths $\ell_{j_1}, \ell_{j_2}, \cdots$, where if the j_k'th request corresponds to a real read, then the block bid_{j_k} is assigned to a new path, and if the j_k'th request corresponds to a fake read, no new assignment occurs.

Suppose we execute Path ORAM with a sequence of requests $r'_1, r'_2, \cdots$ sequentially, where $r'_k = r_{j_k}$ if the j_k'th request corresponds to a real read, and otherwise r'_k is a "special request"

for flushing path ℓ_{j_k} without assigning new paths to any blocks, (and suppose that the same random coins are used for assigning new paths as in execution trace T). At any point, the contents of the stash is identical to that of TaORAM with execution trace T.

It was shown in [126] that the stash size of Path ORAM when executed without "special requests" is bounded by any function $R(N) = \omega(\log N)$ with overwhelming probability. Since the "special requests" only involve flushing a path without assignment new paths (in other words, they only put blocks at lower positions on the path), the probability that the stash size exceeds $R(N)$ decreases. Therefore, the stash size of TaORAM is also bounded by $R(N)$ with overwhelming probability. ■

Step 4 – Response Timing and Sequencer

The above description considers only the obliviousness of the communication between the server and the Processor. Indeed, by the use of "fake reads", every read-path request to the server fetches an independent random path. Their timing, as well as that of the write-back requests, are completely determined by the timing of (the arrival of) logical requests and the schedule of asynchronous network. Hence, the Processor-server communication is oblivious of the logical requests.

Another aspect that has been neglected (on purpose) so far is the timing of replies (to logical requests). Consider the scenario where a sequence of repetitive logical requests arrives in a burst, triggering a real read (for the assigned path), followed by many fake reads (for random paths). When the real read returns, the requested block is found; but, if the Processor replies to all logical requests in one shot and an adversary observes this event, it can infer that there are likely repetitions. To eliminate this leakage, the Processor only replies to a request when the corresponding read-path request has returned, even if it is a fake read. To achieve this, the Processor uses a **response map**, denoted as response.map, that maps each request $(\mathsf{type}, \mathsf{bid}, v)$ to a tuple $\mathsf{response.map}[(\mathsf{type}, \mathsf{bid}, v)] = (b, w)$ indicating whether this request is

ready to be replied to (i.e., $b = \mathsf{true}$ if the corresponding read-path request has returned) and what the answer w is. A request is replied to only when both $b = \mathsf{true}$ and $w \neq \perp$. (See Step 2 and 3 of ANSWER-REQUEST.)

Unfortunately, a more subtle leakage of information still exists in an asynchronous network, and is exploited by our attack against CURIOUS in Section 5.1.2. To see this, consider again the above scenario with one real-read followed by many fake reads. If in addition the real-read is indefinitely delayed due to the asynchrony of the network, the requested block is not retrieved and none of the requests can be answered (even if all fake-reads return without delay). This delay of replies again leaks information; we have explained how an adversary can use this information to violate obliviousness in Section 5.1.2. In order to prevent this attack, TaORAM runs an additional auxiliary module, the **Sequencer**, whose sole function is enforcing that logical requests are replied to in the same order as they arrive.

5.3.3 Client Memory Consumption

The client memory of an ORAM scheme contains both temporary data related to on-going processing of requests, and permanent data that keeps the state of the ORAM scheme. Since the latter needs to be stored even when there is no request present, it is also called the client storage. In TaORAM, the client storage consists of the position map, the stash, and the secret key key, of size respectively $O(N \log N)$, $\omega(\log N)$, and λ (the security parameter); thus,

$$\text{TaORAM Client Storage Size} = O(N \log N + \lambda) \ ,$$

which is the same as Path ORAM.

On the other hand, unlike Path ORAM and other sequential ORAM schemes, the size of temporary data in TaORAM (and other concurrent cloud storage system such as [37]) depends on the number I of concurrent "incomplete" (more details below) logical requests. The number

I in turn depends on various (dynamically changing) parameters, from the rate of arrival of logical requests, to the schedule of asynchronous network, to the processing power of the server and client. Hence, we analyze the size of temporary data w.r.t. I. For TaORAM, we say that (the processing of) a logical request is incomplete, if it has not yet been answered, or updates induced by the request (due to being a write request itself and/or flushing) has not been committed to the server. For each incomplete request, TaORAM keeps temporary data of size $O(\log N)$, leading to

$$\text{TaORAM Temporary Data Size} = O(I \log N) \ .$$

In a normal execution where the rate of processing and the rate of arrival of logical requests are "balanced", since TaORAM writes-back to the server after every k paths are retrieved and flushed, the number I of incomplete requests is roughly k; hence,

$$\text{Normal TaORAM Memory Consumption} = O(k \log N + N \log N + \lambda) \ .$$

Of course, a malicious adversary can drive the number I to be very large, by simply preventing write-back operations to complete. When this is a concern, we can let the system halt whenever I reaches a certain threshold (note that I is known to the adversary, and thus this operation does not break obliviousness of the scheme).

5.3.4 Partitioning

It may be often advantageous to store our tree in a distributed fashion across multiple partitions, e.g. to prevent I/O and bandwidth bottlenecks.

TaORAM is easily amenable to partitioning, without the need of storing an additional partition table as in previous systems [36–38]. If $m = 2^i$ partitions are desired, we can simply

"remove" the top i levels of the tree, storing them in TaORAM's local memory. (Note that this requires storing $O(m)$ additional data blocks locally, but this number is generally not too large.) Then, the rest of the tree can be thought as a forest of m sub-trees (the root of each sub-tree is one of the nodes at the i-th level of the original tree). One can then store each of these sub-trees on a different partition.

Note that the scheme remains unchanged – the only difference is in the data-fetch logic. The tree is now distributed across m partitions, and the TaORAM's local memory. When a path is to be fetched, one retrieves the contents of the first i levels on the path from the local memory, and the remaining levels from the appropriate partition. Every access being on a random path, the load on the partitions is uniformly distributed.

5.3.5 Security

The following theorem summarizes our security statement for TaORAM. The proof, given in Appendix A.2, follows from two facts: First, from our use of the sequencer module, ensuring that the i-th operation is *not* answered until all previous operations are answered. Second, from the fact that all requests retrieve random paths.

Theorem 5.3.2 (TaORAM security) *Assume that the underlying encryption scheme is IND-CPA secure. Then TaORAM is* aaob*-secure.*

5.3.6 Correctness

It is a priori not clear whether the system behaves as expected, or say (for example) we may return inconsistent or outdated values for different requests. Proving correctness of the scheme, therefore, becomes a non-trivial issue in the asynchronous setting (which is in fact even *harder* than proving security). In Appendix A.4, we prove that TaORAM exhibits atomic semantic, i.e., completed operations appear (to an external observer) as if they took effect *atomically*

at some point during their invocation and their response. (We provide formal definitions for correctness in Appendix A.3.)

The core of the proof lies in showing that the fresh-subtree invariant mentioned above always holds (i.e., the contents in the local storage at the proxy is the most up-to-date). Operations then take effect when a write operation writes its value into, or when a value is retrieved from the proxy's local storage.

Remark. We note that packet dropping or delays have a very isolated impact on TaORAM. Indeed, loss of some of the read-path/write-back operations will not result in stalling the system (just in slightly increased memory consumption). This is in sharp contrast to the background shuffling process of ObliviStore [37], which cannot be halted at any point as otherwise the system will stall.

5.4 Experiments

The experiments evaluate TaoStore in two different test environments: simulation based and real world deployment. We start by providing a detailed analysis of TaoStore's performance by deploying the untrusted server to a public cloud (AWS [154]). We then compare TaoStore with ObliviStore and Path ORAM in the hybrid cloud setting using a simulation based environment, which is similar to the setting in ObliviStore paper.

5.4.1 Implementation

We implemented a prototype of TaoStore in C#. We start by briefly highlighting some technical aspects of our implementation.

The trusted proxy (see Figure 5.8) runs an implementation of TaORAM as described in Section 5.3, which internally runs many *threads*, where each is a processing unit responsible

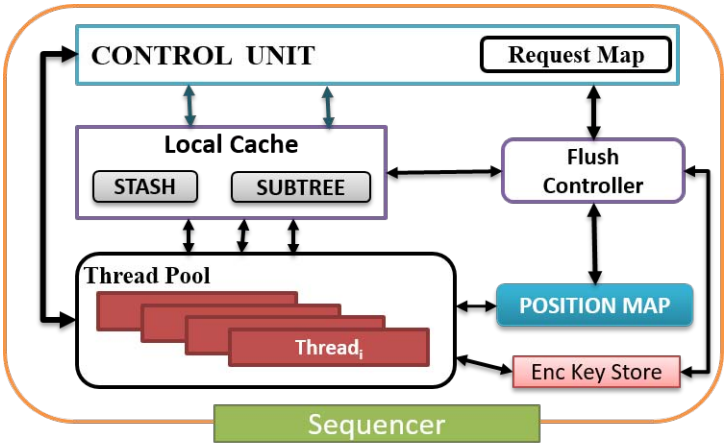


Figure 5.8: Trusted Proxy Implementation

for handling a client request and then returning a response to the client. The *request map* is implemented as a dynamic dictionary in the format of *(*bid*, queue)* pairs where block id, bid, is a key in the map and each value is a queue object that keeps track of the threads waiting for block bid. Additionally, the *control unit* communicates with the threads and the flush controller to maintain the state of the system. The *position map* is an array based data structure. The proxy also has a local cache with 2 components: a subtree and a stash. The subtree is implemented as a dynamic data structure that takes advantage of a dictionary and a tree structure as shown in Figure 5.9. For faster lookup, the dictionary component maintains the information for mapping the blocks to buckets. If a block is stored in the *subtree*, the dictionary points to the bucket in which the block is stored. The nodes themselves also use a dictionary structure to store blocks. Maintaining this two-level structure enables an $O(1)$ lookup for stored blocks. The other caching component, the *stash*, has a dictionary format of (bid, $block$). To provide data confidentiality, the data is encrypted at the bucket level using a semantically secure randomized encryption scheme, AES-128 [155] in CBC-mode, before it is outsourced to the cloud storage.

The components of the local cache are implemented in memory. When paths are fetched from the untrusted cloud storage, concurrent fetches are likely to have overlapping buckets, especially at the top levels of the tree. To avoid locking the complete subtree (which would

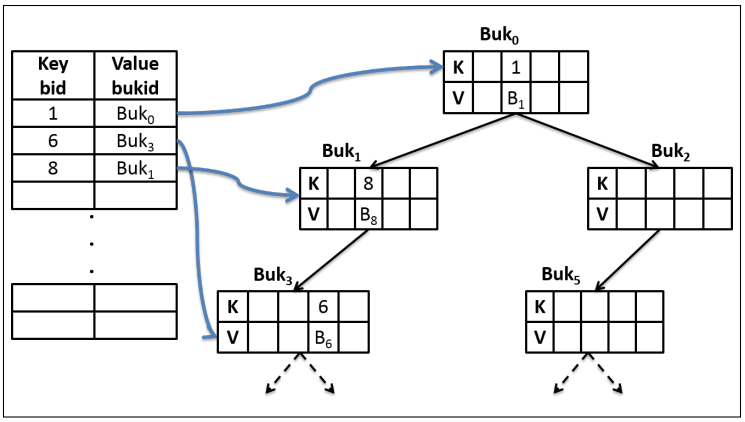


Figure 5.9: Subtree Structure

be very costly), we apply the *read-write lock* mechanism [156] at the bucket level to control concurrent accesses to the shared buckets in the local cache.[9] If a thread wants to perform an insert, an update or a delete operation on a bucket, it has to acquire a write lock for this bucket, to which it gains exclusive access. In contrast, for read operations, it is enough for the thread to acquire a read lock, which still allows several threads to access the same bucket for reading at the same time. The stash is another shared data structure that needs to be controlled. Since it is a block level data storage, we apply read-write locks at the block level. The control unit also uses block level read-write locks to maintain concurrent operations on the request map.

Our server implementation performs I/O operations directly on the disk. TaoStore is an I/O intensive infrastructure, and for higher performance it is important to minimize the I/O overhead. Our implementation performs I/O operations at the path level, i.e., reading or writing the buckets along the path at once, rather than at the bucket level, which would require separate I/O operations for each bucket. Performing I/O at the bucket level requires more I/O scheduling and context-switch overheads; therefore TaoStore avoids it. The server responses are returned with *callbacks* which have significant performance advantages over thread pooling and scheduling.

[9]We stress that our algorithm presentation above *does* lock the whole tree – this makes the proof slightly simpler, but the proof extends also to this higher level of granularity.

TaoStore can cache the top levels of the tree and serve directly from memory to eliminate a significant amount of I/O overhead in the untrusted cloud storage. In our implementation, caching is done using a dictionary data structure.

In real world deployment scenario, the trusted proxy and the server communicate and exchange data over asynchronous TCP sockets.

5.4.2 Experimental Setup

The first set of experiments are conducted to analyze how TaoStore performs as an oblivious cloud storage in the real world. The trusted proxy runs on a machine on a university network with i5-2320 3 GHZ CPU, Samsung 850 PRO SSD, and 16 GB memory. The cloud storage server is deployed to an i2.4xlarge Amazon EC2 instance. The average round-trip latency from the trusted proxy to the storage server is 12 ms. The average downstream and upstream bandwidths are approximately 11 MBytes/s[10].

The second set of experiments are conducted to compare TaoStore with ObliviStore. To be comparable with ObliviStore, we use a configuration which is similar to the ObliviStore paper. The network communication between the trusted proxy and the storage server is simulated with a 50 ms latency. Although there are multiple clients and they query the trusted proxy concurrently, the network latency between the clients and the trusted proxy is assumed to be 0 ms. The trusted proxy and the storage server run on the same machine -it is the machine that is used as a trusted proxy in the initial set of experiments.

In both set of experiments, each bucket is configured to have four blocks of size 4 KB each. The default dataset sizes are 1 GB, i.e. 244,140 blocks and 13 GB, i.e. 3,173,828 blocks for real world and simulation based experiments, respectively. Additionally, the write-back threshold is set to $k = 40$ paths.

[10]Measured using iPerf tool [157].

In our experiments, the clients issue concurrent read and write requests. Three parameters may affect the performance of the system: 1) the number of clients, 2) the scheduling of client requests, and 3) the network bandwidth. For 2), we consider an *adaptive scheduling of requests*, where each client sends the next request immediately after receiving the answer for the previous one. The requested blocks are selected from a uniformly distributed workload and each set of experiments uses the same workload[11].

The main metrics to evaluate the performance are *response time* and *throughput*. Response time spans the time period from initiating a client request until the time that this client receives a response. This metric shows how fast the system can handle client requests. Throughput is defined as the number of (concurrent) requests that the system answers per unit time. The goal is to achieve a low average response time while ensuring high throughput. To report reliable results, each set of experiments is run multiple times and the averages of the gathered results are presented with a 95% confidence interval. Some intervals are not clearly seen in Figure 5.10 due to their small sizes compared to the scale.

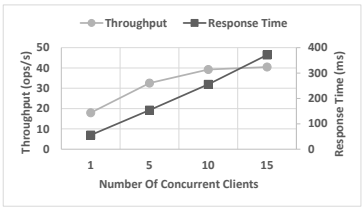
We also note that in order to calculate the experimental results in the steady state, the system is warmed up before taking any measurements. Warming up is achieved by the first 10% of the workload.

5.4.3 Experimental Results

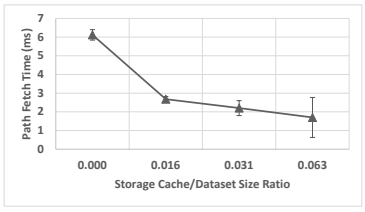
Cloud-based TaoStore Evaluation

In this section, we vary different system parameters and study their effects on the performance of TaoStore by deploying it to a real world environment using AWS.

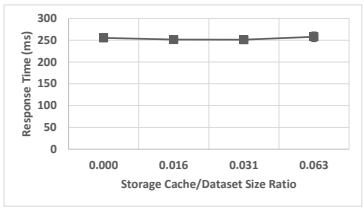
[11]Please note that the distribution of requested blocks does not affect the performance of TaoStore unlike ObliviStore.



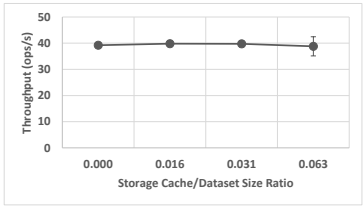
(a) Effect of Number of Concurrent Clients



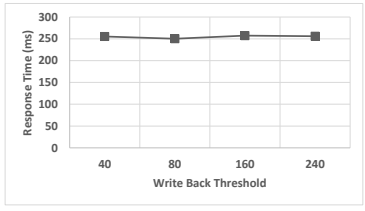
(b) Effect of caching at the untrusted server on average path fetch time from disk



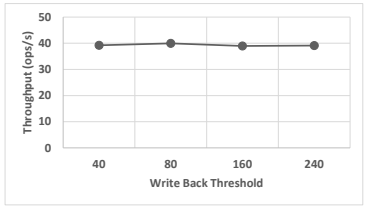
(c) Effect of caching at the untrusted server on response time



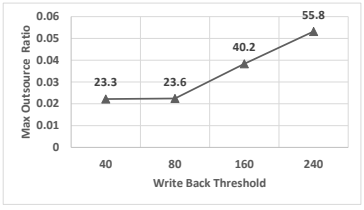
(d) Effect of caching at the untrusted server on throughput



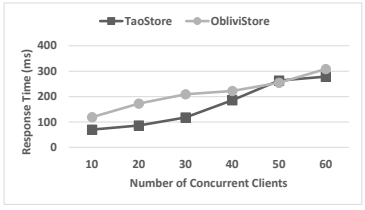
(e) Effect of write-back threshold on response time



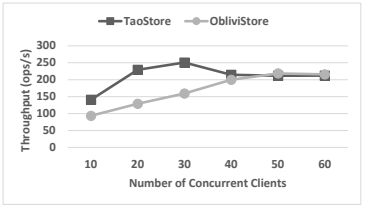
(f) Effect of write-back threshold on throughput



(g) Effect of write-back threshold on maximum outsource ratio (Data labels represent maximum utilized memory in MBytes)



(h) Effect of number of concurrent clients on response time



(i) Effect of number of concurrent clients on throughput

Figure 5.10: TaoStore Performance Analysis

Effect of Concurrency Figure 5.10(a) shows the effect of concurrency on TaoStore's average response time and throughput while varying the number of concurrent clients from 1 to 15. The left and right vertical axes represent throughput and response time, respectively.

With a single client, the response time is 55.68 ms, which leads to a throughput of 17.95 op/s. As the number of concurrent clients increases, the throughput also increases as long as the system can support more simultaneous operations. The system reaches its limit and stabilizes at a throughput of approximately 40 ops/s when the number of concurrent clients is 10. When the number of concurrent clients goes above 10, the clients generate more requests than the system can handle concurrently. In such a case, the clients experience increasingly worse performance in terms of response time although the performance of the system does not degrade in terms of throughput. Consider the case when the number of clients is 15. Although the system achieves approximately the same throughput at around 40 ops/s, the response time increases by 45% compared to the case with 10 concurrent clients. We observe that the network bandwidth is the main bottleneck in our experiments and it is the main reason for the observed behavior. Each path request results in transferring approximately 260-270 KBytes of data from the storage server to the proxy. Since the system handles 40 ops/s, the bandwidth utilization of the system is approximately 10.4-10.8 MBytes/s. Recall that the downstream network bandwidth is 11 MBytes/s, the system utilizes almost all the bandwidth and achieves its best throughput performance at around 40 ops/s.

To understand the system behavior with higher network bandwidth, we perform an additional set of experiments by running a proxy on another Amazon EC2 instance in the same datacenter where the storage server is located. The proxy runs on an m3.xlarge EC2 machine and we measure the bandwidth between the server and the proxy to be 125.25 MBytes/s. In this setting, the system achieves a throughput of 97.63 ops/s with an average response time of 102 ms when the number of clients is 10. The system performance increases dramatically with the increase in network resources, 149% increase in the throughput and 60% decrease in the

response time.

As a result of our experiments we observe that higher bandwidth can facilitate outstanding improvements in the system performance. Therefore, the bandwidth is one of the important issues for oblivious cloud storage systems in a realistic deployment setting as well as supporting concurrency and asynchronicity.

Please note that the default setting for the number of concurrent clients is 10 in the rest of our experiments unless otherwise stated.

Caching at the Cloud Storage Caching the top levels of the tree at the untrusted cloud storage eliminates a significant amount of the I/O overhead. Figure 5.10(b), 5.10(c) and 5.10(d) present the effects of applying caching in terms of response time, throughput, and path fetch time versus caching ratio. The caching ratio represents the amount of data cached in the cloud memory compared to complete dataset size. When there is no caching, the requested buckets in the path are fetched in 6.12 ms from the disk. When the caching is applied, the cached buckets are retrieved from the memory and the remaining buckets are fetched directly from the disk. Caching 1.6% of the dataset, approximately 16 MBytes, decreases path retrieval time from 6.12 ms to 2.68 ms. As the caching ratio increases, the time to fetch path decreases. When this ratio is 6.3%, the path is fetched in 1.7 ms. However, 3-4 ms performance improvement in data retrieval is not reflected in the overall system performance in terms of response time and throughput because of the network bandwidth limitations. As Figure 5.10(c) and 5.10(d) show, the system provides similar throughput and response time over varying caching ratios.

Impact of the Write-back Threshold Recall that the write-back threshold k determines the number of paths that are retrieved from the untrusted cloud storage before a write-back operation is initiated. A large k requires storing more data at the trusted proxy. However, this results in triggering less write-back operations and performing them in bigger batches. The

effects of this parameter in terms of the average response time and throughput are demonstrated in Figure 5.10(e) and 5.10(f). As it can be seen in the results, there is no significant change in the performance with respect to k. This explicitly shows the design advantages of the non-blocking write-back mechanism, since the system performance is independent of the frequency of write-backs.

Memory and Bandwidth Overhead TaoStore's memory overhead mostly depends on the write-back threshold k. In our experiments, we observe that the number of stored blocks in the stash usually does not exceed $2k$. When k equals 40, the stash usually does not contain more than 80 blocks, which requires approximately 320 KB in memory. Therefore, the stash memory overhead is a small constant, while the subtree uses more memory to store retrieved blocks from the untrusted storage. The overall memory usage for the trusted proxy is usually not more than 24 MB when $k = 40$ as shown in Figure 5.10(g), which has an approximate outsource ratio of 0.02. The *outsource ratio* is the ratio of maximum memory usage at the trusted proxy over dataset size. To answer one client query, the trusted proxy needs to fetch approximately 16 buckets, i.e., 256 KB. Increasing the flush trigger count results in using more memory at the trusted proxy; however, there is not much performance gain from increasing the write-back threshold. When $k = 240$, the trusted proxy uses a maximum of 55.8 MB memory, but achieves a throughput of 39.09 ops/s. The results show that TaoStore can deliver a good performance with a very low outsource ratio.

Comparison with Other Works

We now compare TaoStore with Path ORAM and ObliviStore to show how TaoStore can achieve high throughput and lower response times. The implementation of ObliviStore was provided by its authors[12] and we implemented our own version of Path ORAM. All experiments

[12]We would like to thank the authors of ObliviStore for providing us the implementation graciously.

in this section are simulation based and have the same configuration.

Path ORAM provides relatively low response times of 63.63 ms with a corresponding throughput of 7.9 ops/s. Since Path ORAM does not support concurrency, it is not fair to compare it directly with TaoStore. However, the results highlight the importance of providing concurrency for cloud storage systems (also highlighted in [38]).

Although ObliviStore is not secure over asynchronous networks and fails to provide complete access privacy when concurrent requests access the same item even over synchronous networks, the comparisons with ObliviStore aim to provide insights about TaoStore's performance while providing stronger security. Note that the simulation based experiments assume a 50 ms fixed round-trip network latency. Such an assumption prevents network bandwidth limitation issues. Once data is fetched from the disk drive, operations are executed in memory with delays on the order of 1 ms. The performance is affected mainly by the ORAM client side processing and data retrieval from the disk. Please note that since a uniformly distributed workload is used in the experiments, the probability for accessing the same ORAM blocks, which causes a slowdown for ObliviStore as highlighted in [38], is negligible.

Response times and throughput are compared for both systems in Figures 5.10(h) and 5.10(i), respectively. TaoStore and ObliviStore achieve their highest performances at 30 and 50 clients, respectively. When the number of clients is 30, TaoStore reaches a throughput of 250.79 ops/s with a response time of 117.91 ms. When the number of concurrent clients is 30, ObliviStore delivers a throughput of 159.35 ops/s with a response time of 209.07 ms. Hence, TaoStore achieves 57% high throughput with 44% lower response time. ObliviStore has performance issues against demanding applications due to its complex background shuffling and eviction operations (also pointed out in [38]). It deploys an internal scheduler to manage evictions and client requests but in contrast to TaoStore, the eviction process is not directly decoupled from the client request processing. The scheduler schedules a client request if the system has enough resources available. When the client request is scheduled, it acquires

some amount of system resources and these resources are released once the eviction operations are completed. On the other hand, TaoStore can process client requests concurrently and asynchronously, and the write-back operations are decoupled from the client request processing. This allows TaoStore to continue processing client requests while one or more write-back operations are ongoing. With 30 concurrent clients, available resources are utilized aggressively to provide better performance in terms of throughput and response time. This explicitly demonstrates the design advantages of TaoStore compared to ObliviStore. If the number of concurrent clients goes above 30, Taostore's throughput shows a slight decline and the response time increases, due to the increased contention on processing units and I/O. TaoStore's performance plateaus after 40 clients with a throughput of 211-215 ops/s. ObliviStore's achieves its highest throughput of 218.56 ops/s with a response time of 254.45 ms at 50 clients.

In these experiments, a 13 GB dataset is used as in the experimental setup for ObliviStore [37]. In order to operate over a 13 GB dataset, TaoStore requires 15.9 GB physical disk storage in the untrusted cloud storage, while ObliviStore requires 42.9 GB. The difference in storage overhead is due to a significant number of extra dummy blocks ObliviStore requires [37], i.e., if a level in a partition is capable of storing up to x number of real blocks, the same level stores x or more dummy blocks. However, in tree ORAMs, dummy blocks are used to pad buckets if they contain a lower number of real blocks than their capacity. As also seen in the results, TaoStore is a lot less costly compared to ObliviStore in terms of required physical disk storage.

Our evaluations show that TaoStore handles flush and write-back operations better than ObliviStore, which leads to a high client request processing performance.

5.5 Conclusion

TaoStore is a highly efficient and practical cloud data store, which secures data confidentiality and hides access patterns from adversaries. To the best of our knowledge, TaoStore is the first *tree-based* asynchronous oblivious cloud storage system. Additionally, we propose a new ORAM security model which considers completely asynchronous network communication and concurrent processing of requests. It is proven that TaoStore is secure and correct under this security model. Our experiments demonstrate the practicality and efficiency of TaoStore.

Chapter 6

Fault-tolerant Oblivious Data Storage

Recent oblivious cloud storage systems have shown great improvements in terms of efficiency and throughput. However, to the best of our knowledge, none of the earlier oblivious cloud storage systems addresses failures of different system components. Designing systems that run on commodity machine should consider failures of different system components (e.g. machine crashes and network partitioning) are the norm [158]. By missing fault tolerance as an important design principle, existing oblivious cloud storage solutions are not reliable and robust for real-world applications. Oblivious cloud storage systems have many requirements to guarantee privacy against different attacks and applying typical database replication techniques need to be carefully studied in order not to violate privacy or allow attacks. As will be discussed later in Section 6.4, a naive application of existing replication protocols could cause severe security and privacy issues which result in the violation of obliviousness, and reveling access patterns.

In this dissertation, we introduce the first formal study of fault-tolerance for oblivious data storage systems. We develop *generic* fault-tolerance models for a wide class of oblivious cloud systems that consists of a *trusted proxy* and *untrusted cloud storage* for outsourcing the data. For concreteness, we use a recently proposed oblivious, multi-client cloud storage system, called TaoStore [159]. The failure model considers network partitioning and server crash fail-

ures. To overcome such failures, we propose *quorum* based replication strategies for three distinct deployment models: 1) a simple storage replication, 2) centralized replication with the help of a coordinator, and 3) fully distributed replication. Quorum based replication strategies have been used to increase the availability of distributed data. Considering the high computational and disk I/O cost of oblivious cloud storage systems, quorum based replication strategies are good fit for oblivious cloud storage systems. However, the selection of the specific quorum model has a direct impact on the sizes of read and write quorums, and the number of tolerated failures. We evaluate each deployment model separately and develop model specific quorum requirements to ensure correctness while hiding access patterns. Our models are proven secure under the asynchronous ORAM security definition, *aaob-security*, which is introduced in the state-of-the-art multi-client oblivious cloud storage system, TaoStore [159].

6.1 Preliminary

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 and managed by the data owner. Designing systems where privacy is a first class requirement narrows the design decisions such that many common design decisions that enhance the performance might lead to a violation to privacy of access. In this section, we first give a brief overview of quorums and discuss different quorum variations. Then, we explain the generic design of oblivious storage systems that require a trusted proxy. The obliviousness requirements are explained to show how they limit replication design choices.

6.1.1 Quorums

Database replication is commonly used to tolerate server failures and to enhance read throughput and latency. Once data is replicated, consistency becomes an important challenge. We assume linearizability [160] 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 [161]. 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, *how many replica failures f can the system tolerate before it stops completely?*, and performance, *how many replicas should be accessed per read or update operations?* for a given consistency requirement. In this section, we present the trade offs of different replication strategies.

Linearizability and operation consistency on the object level are achieved using quorums and version numbers [162]. 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 of 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 6.1 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 requires majority read quorums and majority write 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. Cheung et al. [163] 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 [164] 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.

Table 6.1: Summary of different replication strategies, their requirements, and their guarantees.

Replication Strategy	$\lvert q_r \rvert$	$\lvert q_w \rvert$	f
Master/Slave (read optimized)	1	N	0
Master/Slave (write optimized)	N	1	0
Majority quorums	N/2	N/2	N/2 - 1
Grid quorums [163]	$\sqrt{N}$	$2.\sqrt{N}$	$\sqrt{N}-1$
Tree quorums [164]	$log(N)$	$log(N)$	$log(N)-1$

6.1.2 Overview of Oblivious Systems with a Trusted Proxy

In this section, we give a brief overview of oblivious storage systems that depend on a trusted proxy [36, 38, 159]. As shown in Figure 6.1, 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 encrypted and outsourced to the storage and the meta-data to locate objects in the storage is maintained

in the trusted proxy. Clients submit object lookups, *get(k)*, and object updates, *put(k,v)*, to the trusted proxy. 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 between 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 set), 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.

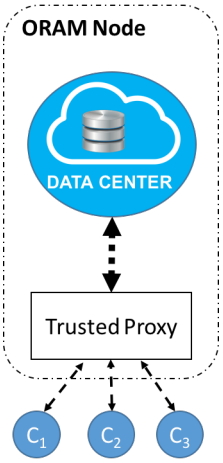


Figure 6.1: An ORAM node and clients

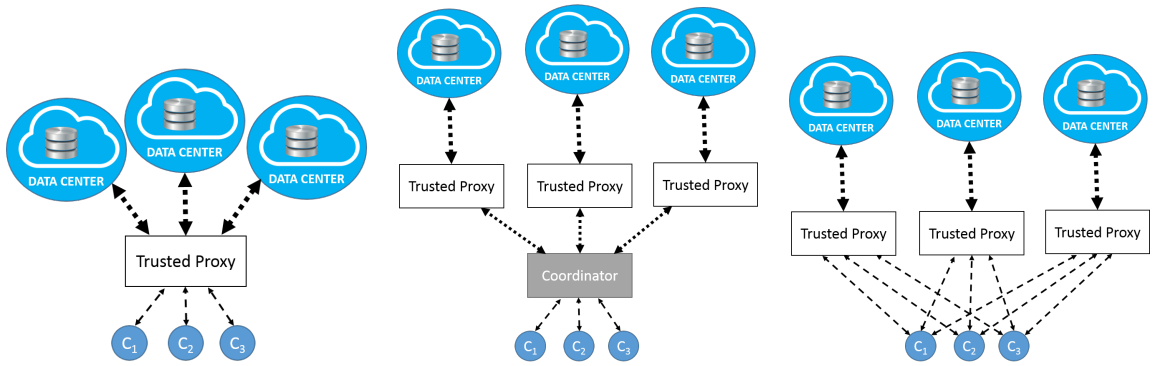
In an asynchronous oblivious storage system like TaoStore [159], the trusted proxy can perform multiple oblivious retrievals, cache the set of retrieved objects in memory, and perform a batched oblivious eviction for multiple retrievals. By doing so, the cost of eviction is amortized. It is important to mention that an eviction should not write any data objects that have not been previously retrieved.

When the storage is replicated, a read quorum determines the number of storage replicas that should be accessed in every oblivious retrieval. Notice that an oblivious retrieval is performed for every client call regardless of weather it is a read(get) or a write(put). A write quorum determines the number of storage replicas that should be updated in every oblivious eviction.

6.2 Failure Model

Many ORAM constructions depend on a trusted proxy to serve client requests [37,38,159]. Data is outsourced to the cloud and client requests are sent to the trusted proxy. A trusted proxy

typically resides in a private cloud and is responsible to serve client requests in an oblivious way. In this section, we present the failure model and propose solutions that tolerate different failures. We provide a privacy analysis for the proposed models in Section 6.4.



(a) Storage Replication (b) Centralized Replication (c) Distributed Replication

Figure 6.2: Fault-tolerant Deployment Models

Cloud storage outage: the first failure threat that can hit an ORAM construction is the outage of the storage system. This could happen in the form of a storage server crash failure or a network partitioning. To tolerate this failure, we propose the first model, Figure 6.2(a), where data is replicated into multiple storage servers and requests are handled by one trusted proxy. These servers could be in the same datacenter to tolerate one or more machine crash failures, or they could be hosted in multiple datacenters to tolerate datacenter scale outages, planned or unplanned due to a nature disaster or power failures. Also, data replicas could be hosted in datacenters of different providers (e.g. Amazon and Azure) to tolerate cascading failures within one provider datacenters. System administrators of different applications should decide to replicate the storage within one datacenter or multiple datacenters of one cloud provider or multiple cloud providers depends on the application requirements.

Trusted proxy failure: the trusted proxy is a crucial component in ORAM constructions. It maintains the meta-data used to locate objects in the storage. Also, it runs the ORAM algorithm to preserve privacy. The failure of the trust proxy takes an ORAM node out of

service. Therefore, we present our second model, Figure 6.2(b), where the trusted proxy and the cloud storage are replicated. Unlike the first model which assumes one ORAM node backed up with multiple storage replicas, the second model assume totally independent ORAM nodes. To manage client requests, a coordinator is introduced to manage client requests, update data object in different ORAM nodes, and retrieve data objects with the highest version number among all ORAM nodes. This model handles failures of an ORAM node represent by a failure of its trust proxy or a failure of its storage.

Coordinator failure: the second model introduces a coordinator to handle client requests from different ORAM nodes to achieve linearizability of object accesses. However, a failure of a coordinator can bring the whole system down. Therefore, the third proposed model, Figure 6.2(c), assumes a fully distributed replication where clients are responsible for directly communicating with different ORAM nodes. This model tolerates the failure of ORAM nodes without introducing any single point of failure for the whole system.

In the first model, the storage data and its structure are identical in all the replicas and one trusted proxy maintains the meta-data of how to access objects from the storage system and caches previously retrieved objects until they are evicted back. However, in the second and the third models, the structure of the data can be completely different between different storage nodes and the meta-data of access is independently managed by different trusted proxies.

Unhandled failures: in this section, we presented different failures that can breakdown an ORAM systems and we proposed different models to handle these failures. We consider server crash failures and network partitioning. However, Byzantine failures [165, 166] where servers can act maliciously or data packet can be corrupted in the network are out of the scope of in this work.

6.3 Threat Model and Security Definition

We consider the threat and security model for asynchronous ORAM that is introduced in [159]. 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 she can delay the messages arbitrarily long. Additionally, an adversary can schedule access operations adaptively and learn the timing of the responses. The 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.

aaob security 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 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 OClient. The attacker $\mathcal{A}$ can, at any point in time, invoke 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 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`.

The aaob-*advantage* of the adversary $\mathcal{A}$ against ORAM is defined as

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

An ORAM scheme 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).

6.4 Models

As discussed earlier, we consider three different deployment scenarios that ensure different levels of fault-tolerance. In this section we discuss the details of the proposed models. Our contribution of bringing fault-tolerance in oblivious data storage is orthogonal to oblivious algorithms and can also be integrated with any proxy based oblivious data storage frameworks that are aaob-secure. We use TaoStore as an instance of such frameworks to explain our protocols.

6.4.1 Storage Replication

The initial deployment scenario is a simple extension of existing oblivious cloud storage systems with a trusted proxy like ObliviStore [37] and TaoStore [159]. In this extended model, the storage is replicated on multiple instances to ensure fault-tolerance on the remote storage, but not the proxy (Figure 6.2(a)). The trusted proxy communicates with multiple storage replicas and clients, and executes the oblivious algorithms. The failure model considers cloud storage outage and the aim is to provide high availability and fault-tolerance even in the presence of such failures.

Protocol

In this model, clients send their read/write requests for an object to the trusted proxy. In oblivious data storage systems, any read or write on a object initially generates an oblivious retrieval to the cloud storage to retrieve the requested object hidden among some other objects. In tree-based ORAMs, this involves retrieving a set of objects along a path from a storage which is organized as a binary tree. In hierarchical ORAMs, this involves accessing every level of an storage which is organized as $\log N$ level hierarchical hash tables. Some number of oblivious retrievals are followed by a single oblivious eviction that shuffles the retrieved

objects and evicts them to the remote storages. For oblivious eviction, TaoStore [159] performs flushing and write-back in batches, while ObliviStore [37] performs background shuffling and eviction. Although they differ in technical details, at a conceptual level, they are quite similar. We here explain how our fault-tolerance model works on top of any ORAM scheme with a trusted proxy.

Recall that we deploy a *quorum* based replication strategy. In this context, a *read* quorum is used to perform oblivious retrieval, and a *write* quorum is used to perform oblivious eviction. Although the approach is straightforward at a high level, the protocol requires clarifications regarding the technical details to ensure correctness. Let $\mathcal{S}$ denotes a set of storage replicas, $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_N\}$. Each storage replica consists of a fixed number of objects, and a storage $\mathcal{S}_i$ is defined as the union of M objects, i.e., $\mathcal{S}_i = \bigcup_{id=1}^{M} b_{id}$ where b_{id} is the id^{th} object in the storage. Each storage replica is initialized identically during the deployment. A set Q consists of all non-empty subsets of $\mathcal{S}$. Read and write quorums, denoted by q_r and q_w, respectively, are elements of Q.

Irrespective of whether the request is read or write, the proxy, first, has to perform oblivious retrieval (or issue a fake access in some special cases to hide repetitive access) to process the request for an object. This is important to ensure oblivious access, since an adversary should not be able to distinguish any write request from reads. In proxy based solutions, objects contained in the proxy have the most up-to-date value. The aim is to fetch all up-to-date objects that are not stored in the proxy with oblivious retrieval. The proxy initially, forms q_r and then performs oblivious retrievals on all $\mathcal{S}_i$ that are in q_r, i.e., $\mathcal{S}_i \in q_r$. As a response to oblivious retrieval, each storage replica in q_r returns the same set of objects to the proxy. Each storage replica might have a different state at any time, i.e., the same objects at different replicas might have different content. When the proxy receives the multiple versions of an object, it has to decide on the latest value of the object. To overcome such an issue, our protocol requires a simple *versioning control* mechanism for objects to maintain correctness under asynchrony and

concurrency. Maintaining version numbers for each object is necessary for two crucial cases in our model to ensure correctness. First, when oblivious retrieval fetches objects from the set of storage replicas in q_r, the proxy is able to select the most up-to-date versions of objects by checking their version numbers. After deciding on the most up-to-date version of objects, the proxy executes the base oblivious algorithm using the most up-to-date objects and returns a response to the client requests. Second, stale data never overwrites the up-to-date data in the remote storage during the oblivious eviction. After some number of oblivious retrievals, the proxy performs an oblivious eviction on q_w which shuffles the objects in the proxy and evicts them back to the remote storages in q_w. During the oblivious eviction, each object is assigned a new incremented version number. Upon the acknowledgments from all replicas in q_w, the local copies of objects can be removed from the proxy by following the base ORAM algorithm. For the next oblivious retrieval for the same object, the proxy will be able to identify the most up-to-date version of the object using to the versioning mechanism (which is incremented at each eviction). Note that some cloud storage systems already have their own versioning control mechanism to maintain correctness. For example, TaoStore tags each bucket (container with a fixed number of objects) with a simple version number during the write-back operation.

The storage replication protocol with a version control mechanism needs to follow rules for the selection of q_r and q_w to ensure the correctness of operations and fault-tolerance. We now discuss the details of quorum requirements for the storage replication model.

Quorum Requirements

The main goal is to ensure that the proxy is able to retrieve the most up-to-date objects from the remote storage unless the objects have copies in the proxy. Recall that the objects in the proxy are up-to-date, while the storage server contains the most up-to-date content for the remaining objects. We can guarantee such a property by imposing the basic quorum requirement:

- Any read-path and write-back quorums must intersect in at least one storage replica.

This is the standard quorum requirements and it ensures that $\forall q_r, q_w \; q_r \cap q_w \neq \varnothing$. The read quorum is always able to bring the most up-to-date content into the proxy. Any quorum based protocol ensuring this property can be used to provide availability. However, depending on the quorum model the provided fault-tolerance and availability changes. For example, majority based protocols [162,167] provide good availability for both read and write, but reasonably low fault-tolerance. On the other hand, a simple "*read-one, write-all*" approach is very efficient and provide high read availability at the cost of least write availability and no fault-tolerance. All replicas must be operational for write operation to proceed. Although this might be considered infeasible for many applications, if any ORAM scheme is capable of performing data retrieval from the cloud storage and processing client request without being able to perform eviction procedure due to failures, such an ORAM scheme can benefit from such a quorum selection at a cost of the increase in the local cache size in the proxy. The decision of the quorum selection is left to system administrators.

Theorem 6.4.1 *The storage replication model that satisfies the necessary quorum constraint is correct and secure if the underlying ORAM scheme is linearizable and aaob-secure.*

Proof: **Correctness.** The correctness of the storage replication model relies on ensuring linearizability. Basically, once a write completes, all later read operations should return that value of write or a one with a higher version number. Once a read returns a particular value, all later reads should return that value or a value with a higher version number. We already assume that the underlying ORAM scheme is linearizable. As long as the oblivious retrieval from the remote storage brings most up-to-date data that has not been already contained locally, the ORAM scheme satisfies linearizability.

Assume that the oblivious retrieval happens after an eviction to the cloud storage and the proxy retrieves some set of objects from q_r. The eviction was confirmed from replicas in q_w.

Assume that one of the objects that was fetched from q_r is stale (was written in an earlier write quorum $q_w^{'}$) and not contained in the trusted proxy. This means $q_w^{'}$ happened before q_w and q_r reads from $q_w^{'}$. Since $q_r \cap q_w \neq \varnothing$, this is impossible. There is at least one replica in q_r that has most up-to-date version of the object. By contradiction, the oblivious retrieval always reads from the latest confirmed eviction; therefore, the storage replication model is linearizable. ■

Proof: **Security.** The storage replication model is secure if it satisfies aaob-security. We already assume that the underlying ORAM scheme is secure. This means Adv that the $\mathcal{A}$ gains is negligible in the base case. The storage replication model causes trusted proxy to perform oblivious retrieval from q_r and perform eviction on q_w. The selection of quorums and the execution of algorithms on top of them does not depend on the actual choice of the challenge bit b, when $\mathcal{A}$ cannot see the content of the messages sent over the network. In particular, $\mathcal{A}$ can explicitly see the mapping between operations and the formed quorums (also their sizes) in addition to its view in the base case. Irrespective of whether the operation is read or write, the proxy forms a randomly uniformly selected read quorum q_r in each oblivious access, fetches the content from remote storages in q_r, and later form a randomly uniformly selected write quorum q_w to write the local content back to the remote storage replicas in q_w. The two-timing consistent execution of requests are oblivious in the adversarial model and the $\mathcal{A}$ does not gain any aaob-advantage from the storage replication model. Therefore, the model is secure. ■

Recall that this model only ensures storage level fault-tolerance and the proxy is still a single point of failure. If it fails, the storage framework will halt. To ensure fault-tolerance against proxy failures, we introduces two new models in the following sections.

6.4.2 Centralized Replication

The trusted proxy maintains the necessary metadata to execute oblivious algorithms and its failure causes the framework to halt. This is not desired behavior for any cloud storage

framework. Although it is a widely adopted assumption in oblivious cloud storage systems, it is not realistic to assume a no failure environment. To overcome this issue and prevent the proxy becoming a single-point of failure, we introduce a centralized approach where the storage and proxy are replicated and a new module, called *coordinator*, manages client requests as depicted in Figure 6.2(b). This approach can be considered as a black-box ORAM node replication, where each proxy is associated with one storage server and does not communicate with other proxies and storage servers. A proxy and a storage server forms an ORAM node. The coordinator is responsible for mediating communications between clients and proxies. Additionally, it ensures the correctness of replicated protocol by maintaining the global state of each object locally. The failure model considers cloud storage outages and trusted proxy failures.

Protocol

The obvious difference compared to the earlier model is the granularity of the replication. Rather than replicating only the server storage, this model replicates ORAM nodes. Here, the clients send their requests to the coordinator. Upon receiving a request from a client, the coordinator sends this request to a quorum of ORAM nodes, where the trusted proxies of each ORAM node receives the requests. Each ORAM node executes its oblivious access algorithm which involves oblivious retrieval and eviction, and returns a response to the coordinator. After receiving all responses, the coordinator returns a response to the corresponding client. In this model, *read* quorums (q_r) are formed to handle client read requests, while client write requests on objects are handled with write quorums (q_w).

The coordinator is the center of all communications and need to maintain a global view of objects to ensure correctness in the system. Each object has a version number and this information is maintained with the help of the coordinator. The coordinator keeps a separate version number for each object. In addition to this, each proxy has to maintain the current version num-

ber of each object for its framework. The coordinator and proxies in replicated ORAM nodes maintain a dictionary based data structure, called version.map, to store the version numbers for each object.

Let $\mathcal{O}$ denote a set of N ORAM node replicas, i.e., $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_N\}$. A set Q contains all non-empty subsets of $\mathcal{O}$. q_r and q_w are elements of Q in this model.

The clients initiate their requests in the form of $\mathsf{op}(\mathsf{type}, \mathsf{b_{id}}, \mathsf{data_{id}})$. The parameter type defines whether the operation is read or write, while $\mathsf{b_{id}}$ is the object identifier that the client wants to access. $\mathsf{data_{id}}$ is necessary for write operations and contains the new value of $\mathsf{b_{id}}$. The content is filled with dummy data for read operations to ensure obliviousness. We use $\perp$ to denote dummy data. After receiving a request, the coordinator checks the type of the request and forms a new request $\mathsf{req}(\mathsf{type}, \mathsf{b_{id}}, \mathsf{data_{id}}, \mathsf{v_{id}})$ to fetch data from ORAM node replicas. This new requests also contains the version information for object, which will be used in write operations. After receiving the request, the coordinator performs the following operations depending on type:

- $\mathsf{type} = read$: sends $\mathsf{req}(\mathsf{read}, \mathsf{b_{id}}, \perp, \perp)$ to retrieve b_{id}.
- **type** $= write$: initially check the latest version number of $v_{id} = \mathsf{version.map}[b_{id}]$ locally to get the latest version number for $\mathsf{b_{id}}$. Then, send the request $\mathsf{req}(\mathsf{write}, \mathsf{b_{id}}, \mathsf{data_{id}}, \mathsf{v_{id}} + 1)$ to q_w and updates version.map with a new version number, i.e., $\mathsf{version.map}[b_{id}] = v_{id}+1$.

When a proxy receives $\mathsf{req}(\mathsf{type}, \mathsf{b_{id}}, \mathsf{data_{id}}, \mathsf{v_{id}})$, it executes the oblivious access algorithm with the following minor change depending on type:

- $\mathsf{type} = read$: send response to the coordinator with a version number of $\mathsf{b_{id}}$, $\mathsf{response}(\mathsf{b_{id}}, \mathsf{data_{id}}, \mathsf{version.map}[\mathsf{b_{id}}])$.
- $\mathsf{type} = write$: **a)** the version number of $\mathsf{b_{id}}$ in the proxy is less than $\mathsf{v_{id}}$, then the version number of $\mathsf{b_{id}}$ should be updated locally after processing the request such that

$\mathsf{version.map}[\mathsf{b_{id}}] = \mathsf{v_{id}}$, or **b)** the version number of $\mathsf{b_{id}}$ in the proxy is greater than $\mathsf{v_{id}}$, then the proxy issues a fake access since the value is already updated with a higher version number. The response will be $\mathsf{response}(\mathsf{b_{id}}, \perp, \perp)$.

After receiving responses from all replicas in the quorum, the coordinator performs the following operations depending on type:

- $\mathsf{type} = read$: Select the response with the maximum version number and return $data_{id}$ of this response to the client.
- $\mathsf{type} = write$: Acknowledge the client, the operation is done.

Quorum Requirements

Similar to the earlier model, the quorum sets have a direct impact on the level of availability and fault-tolerance. The goal of our quorum approach is significantly different, though. The protocol aims to executes client requests **obliviously** while preventing the coordinator from responding to clients with a stale data. This can be achieved with the standard quorum constraint:

- any read and write quorums must intersect in at least one ORAM node replica,

guarantees the requirement for retrieving most up-to-date data. However, to ensure obliviousness, the adversary must not be able to discover whether the operation is read or write. Consider "read-one, write-all" quorum system with 5 ORAM node replicas as an example. Whenever the coordinator receives a request for a read operation, it will select a quorum of size $|q_r| = 1$ and sends the request to the selected replica. On the other hand, if the operation is write, $|q_w| = 5$. By observing the sizes of quorums for each access, an adversary can distinguish reads from writes, which is an violation of obliviousness and the targeted security model. Therefore, the selection of quorum must also ensure the following constraint:

- the sizes of any read or write quorums must be the same, $\forall q_r, q_w \; |q_r| = |q_w|$.

Theorem 6.4.2 *The centralized replication model that satisfies the necessary quorum constraints is correct and secure in the presence of at most* $|q_r| - 1$ *failures if the underlying ORAM scheme is linearizable and aaob-secure.*

Proof: **Correctness.** The distributed replication model introduces a new round of communication form quorums to decide on the highest version number of an object. Indeed, this procedure replaces the existence of the coordinator in the centralized replication model. After deciding on the version number, both distributed and centralized models follow similar protocol. Therefore, if we prove that the version number decision algorithm is correct, then the distributed replication model also works correctly.

To learn the highest version number for $\mathsf{b_{id}}$, a client sends a request to q_r or q_w depending on the type of an operation. Since either $q_r \cap q_w \neq \varnothing$ or $q_w^{'} \cap q_w \neq \varnothing$ holds, there must be at least one replica that has the highest version number of $\mathsf{b_{id}}$. After learning the highest version number, the client increments the version number by 1 and attaches its identification number for the write operation. If there is no other concurrent write request on $\mathsf{b_{id}}$, the object will be accessed obliviously and updated with a new version number on the replicas in q_w. If there is a concurrent write request on $\mathsf{b_{id}}$, one of the version numbers is always higher than the other since clients have unique identifier. The request with higher version number is executed in q_w. Therefore, the replicated storage never becomes inconsistent with the proposed version number model. Since, the version number decision works correctly, the distributed replication model works correctly as well.

■

Proof: **Security.** As discussed in the proof of correctness, the distributed replication model introduces a new round of distributed version decision round. In this phase, a client

requests version numbers from either q_r or q_w. When $\mathcal{A}$ cannot see the content of the messages, it cannot differentiate whether the request is for a read or a write, since $|q_r| = |q_w|$. Therefore, Adv that $\mathcal{A}$ gains is negligible during this process. Intuitively, the remaining of the distributed replication model works similarly to the centralized replication model. Given the centralized replication model is secure, the distributed replication model is secure as well. ■

This model provides a fault-tolerance and availability even in the presence of trusted proxy failure. However, the coordinator is a centralized global entity that can form a single point of failure. Though, it provides a good distribution of load (performance) and high availability.

6.4.3 Distributed Proxy and Storage Replication

To overcome the issue of having a single point of failure in the system deployment, this model is completely distributed where there is no central entity (trusted proxy or coordinator) that causes a system to halt in the case of failure. Similar to the centralized replication model, the distributed replication model replicates ORAM nodes as depicted in Figure 6.2(c). Clients can communicate with any ORAM node replica, in which a proxy is only associated with one storage. The failure model again considers cloud storage outages and trusted proxy failures.

Protocol

The main contribution of the coordinator in the centralized replication model is to keep track of version numbers for each object and assign incremented version numbers for each write operation. This is necessary to retrieve the most up-to-date object values by read operations and prevent conflicting write operations. Maintaining and updating version numbers is also crucial for the correctness of the distributed replication model. Similar to the centralized replication model, the read quorums are formed for client read requests and write quorums are formed for client write requests. Before requesting an operation, a client needs to learn

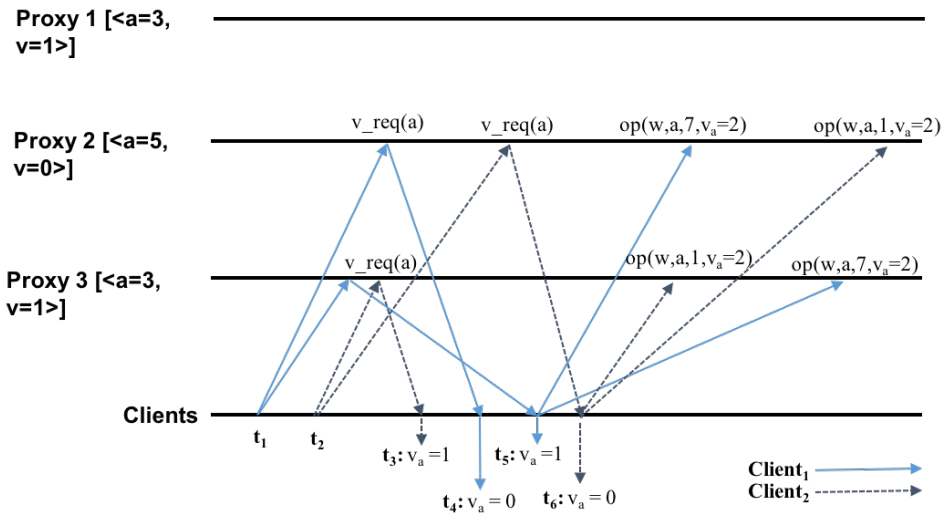


Figure 6.3: Inconsistency of Concurrent Operations with Majority Quorum

the highest version number of an object to ensure the correctness of an operation. Similar to the centralized model, proxies in each ORAM node maintain version numbers for each object in version.map. Assume a client wants to perform a write operation on b_{id}. The client initially selects a quorum of replicas, and then requests a version number of b_{id} from each replica in the quorum. Upon receiving version numbers from all replicas, the client decides on the maximum version number, and then requests a new operation by incrementing the maximum version number of b_{id} by 1. Note that whether the operation is read or write, the client needs to ask for version numbers from a quorum to hide the type of the operation before sending a request to a quorum to perform an operation on the object.

Concurrency Problem in Oblivious Setting We need a distributed synchronization algorithm that decide on a global state of an object. This is a popular and well-studied concept in the distributed systems, e.g., [168]. There have been well established solutions that have been adopted widely in real world applications. However, in the context of oblivious storage systems, the direct application of existing approaches either leads to a conflicting state or a violation of obliviousness. As an illustration, consider a majority quorum replication example

with 3 replicas provided in Figure 6.3. At time t_0, the proxies have $[< a = 3, v = 1 >]$, $[< a = 5, v = 0 >]$ and $[< a = 3, v = 1 >]$ for object a. To clarify, the first replica has $\mathsf{a} = 3$ with a version number 1. The third replica also has the same value and the version number for a. On the other hand, the second replica has a stale a with a value of 5. The first client requests a version number of a from quorum of $\{2, 3\}$ at t_1. The second client also requests a version number of a from the same quorum at time t_2. The second client receives a response from the third replica at t_3. The version number of a is 1. The response with a version number 0 from the second proxy is delivered to the second client at t_6. On the other hand, the first client receives responses with version number 0 and 1 at t_4 and t_5, respectively. After receiving all responses from a quorum, the clients decide on the new version number and issue write operations to the same quorum of replicas. The first clients see that the maximum version number is 1, then it increments it by 1 and issue a write operation which updates a to 7. Similarly and simultaneously, the second client increments the maximum version number she received by 1 and sends a request to the quorum to update a to 1. The second proxy receives the request in the order of $\mathsf{op}(\mathsf{write}, \mathsf{a}, 7, 2)$, $\mathsf{op}(\mathsf{write}, \mathsf{a}, 1, 2)$ whereas the third proxy receives them in the reverse order. This will put the storage in a inconsistent state, and, of course, should be avoided.

A widely used approach to deal with such inconsistencies is use of *locking*. A simple application of locking to control concurrent access solves the problem of inconsistency, however, it causes a violation of obliviousness and access privacy. Consider the example provided in Figure 6.3 again. Assume proxies execute the following simple algorithm: Upon receiving a request for an object, acquire a lock for this object and return a response to the client with the current version number. With the completion of write operation release the lock. This prevents any concurrent access on the same object and ensures consistency guarantees. When the second and third proxy receive version number requests for object a, they put a lock on object a. Any concurrent request on a, which is simultaneously sent by the second client will wait until the locks are released. If the concurrent request is on a different object other than a, then the

proxy will acquire a lock for this object and continue processing client request concurrently with the initial request for a. This allows an adversary to understand whether two concurrent accesses are on the same object or not, which obviously violates access privacy.

Oblivious and Concurrent Access To overcome the shortcomings of existing techniques, we introduce an oblivious consensus protocol benefiting from Lamport logical clocks [168] with unique client IDs and pseudorandom permutation (PRP) [169].

Lamport clocks are used to determine the partial order of events. If any event A happens before an event B, the logical clock of A is less than the logical clock of B, i.e., $\mathsf{C}(\mathsf{A}) < \mathsf{C}(\mathsf{B})$. This is similar to the version numbers that are used in the earlier sections. However, in this case unique client IDs are attached to the version numbers to differentiate two events if they are concurrent events with the same version number. The version number of x is an ordered pair $v_x =< v_{max}, c_k >$ where v_{max} is the maximum version number for x and c_k is the identification number of client k that generates the version number. The second entry is used to break ties. When a write request is issued, a proxy needs to confirm whether the provided version number is higher than the existing version number. Only if it is higher, the request is executed. For any two version numbers $v_I =< v_i, c_i >$ and $v_J =< v_j, c_j >$, if $v_i > v_j$, then $v_I > v_J$. In case $v_i = v_j$, the order is decided by comparing c_i and c_j. If $c_i > c_j$, then $v_I > v_J$. Recall that client identifier numbers are unique numbers associated with clients.

A client initiates the request for a version number in the form of $\mathsf{v\_req}(\mathsf{b_{id}})$, from a quorum of replicas independent of its $write$ or $read$ intention. This is necessary for obliviousness, since any adversary should not be able to distinguish read operations from writes. Upon receiving the request, the proxy responds to the clients by returning the most recent version number of $\mathsf{b_{id}}$, i.e., $v_{id} =< v_{id}, c_k >$ from the where c_k is the identification number of the client that updated $\mathsf{b_{id}}$ last. The proxy returns $\mathsf{v\_res}(\mathsf{v_{id}})$ to the client. After receiving responses from the quorum, the client selects the response with the highest version number. For a write operation,

the client generates a request $\mathsf{op}(\mathsf{write}, \mathsf{b_{id}}, \mathsf{data_{id}}, < \mathsf{v_{id}} + 1, \mathsf{c_k} >)$, and then sends it to the quorum. In case of a read, the request is $\mathsf{op}(\mathsf{read}, \mathsf{b_{id}}, \perp, <\perp, \perp>)$. At any time, if multiple clients want to perform concurrent and simultaneous write operations on the same object, the global state of the storage will be consistent thanks to unique identifier. A subtle issue with using such an approach is that a client with a higher ID is always prioritized for all concurrent write operations, since the basic storage model updates the value if it has a higher timestamp. To overcome this subtle issue, we use pseudorandom permutation which provides uniform randomness for providing priorities to clients.

Pseudorandom permutations (PRP) is a function that cannot be distinguishable from a random permutation. For any given key $\mathsf{K} \in \{0,1\}^s$, a pseudorandom function is defined as $\pi_K : \{1,0\}^s \times \{1,0\}^n \rightarrow \{1,0\}^n$. The key is used to generate the random permutation. Recall that the aim is to make all clients equivalent in case of concurrent operations. Using the same key every time will generate the same permutation every time, which would obviously be contrary to the purpose. To solve this problem, the keys are derived from a single seed by appending an object identifier and version number together.

During the initialization, each proxy replica receives the same seed, denoted by seed, to generate permutation keys later on. During the execution, the permutation key is generated using the SHA-256 cryptographic hash function, which generates an almost unique hash. It is deterministic, which means as long as the same input is provided the outputted hashes will also be equal to each other. The proxy needs to decide on the higher version number for each object. Therefore, the object identifier is concatenated to the seed, which ensures that $\mathsf{SHA256}$ generates hashes specific to objects. Assuming a is the object that is going to be updated, the generated hash on any proxy will be $\mathsf{SHA256}(\mathsf{seed}||\mathsf{a})$. Unfortunately, this does not solve the issue completely, since one of the clients will always be prioritized for specific object. To prevent this, the version number is also appended to the input of $\mathsf{SHA256}$. Basically, $\mathsf{SHA256}(\mathsf{seed}||\mathsf{b_{id}}||\mathsf{v_{id}})$ gives us the key that is going to be used in PRP, i.e., $\pi_{\mathsf{SHA256(seed}||\mathsf{b_{id}}||\mathsf{v_{id}})}$.

Upon receiving $\mathsf{op}(\mathsf{write}, \mathsf{b_{id}}, \mathsf{data_{id}}, < \mathsf{v_{id}} + 1, \mathsf{c_k} >)$, a proxy initially computes a permuted version number, $v_{per} =< v_{id} + 1, \pi_{SHA256(seed||b_{id}||(v_{id}+1))}(c_k) >$, and then compares with $\mathsf{version.map}[\mathsf{b_{id}}]$. If $v_{per} > \mathsf{version.map}[\mathsf{b_{id}}]$, the proxy executes the request and updates the version number in version map for b_{id} with its new value, i.e., $\mathsf{version.map}[\mathsf{b_{id}}] = v_{per}$. Otherwise, the proxy issues a fake access to maintain obliviousness and return a failure response to the client. In case of a read request, the proxy processes the request obliviously and return $\mathsf{response}(\mathsf{b_{id}}, \mathsf{data_{id}}, \mathsf{version.map}[\mathsf{b_{id}}])$ to the client.

Quorum Requirements

The quorum requirements of the distributed replication model is same with the centralized replication model which is discussed in Section 6.4.2. In brief, the necessary quorum requirements are:

- any read and write quorums must intersect in at least one TaoStore node replica,
- the sizes of any read or write quorums must be the same, $\forall q_r, q_w \; |q_r| = |q_w|$.

Theorem 6.4.3 *The distributed replication model that satisfies the necessary quorum constraints is correct and secure in the presence of at most* $|q_r| - 1$ *failures if the underlying ORAM scheme is linearizable and aaob-secure.*

Proof: **Correctness.** The distributed replication model introduces a new round of communication form quorums to decide on the highest version number of an object. Indeed, this procedure replaces the existence of the coordinator in the centralized replication model. After deciding on the version number, both distributed and centralized models follow similar protocol. Therefore, if we prove that the version number decision algorithm is correct, then the distributed replication model also works correctly.

To learn the highest version number for b_{id}, a client sends a request to q_r or q_w depending on the type of an operation. Since either $q_r \cap q_w \neq \varnothing$ or $q_w^{'} \cap q_w \neq \varnothing$ holds, there must be at least one replica that has the highest version number of b_{id}. After learning the highest version number, the client increments the version number by 1 and attaches its identification number for the write operation. If there is no other concurrent write request on b_{id}, the object will be accessed obliviously and updated with a new version number on the replicas in q_w. If there is a concurrent write request on b_{id}, one of the version numbers is always higher than the other since clients have unique identifier. The request with higher version number is executed in q_w. Therefore, the replicated storage never becomes inconsistent with the proposed version number model. Since, the version number decision works correctly, the distributed replication model works correctly as well.

■

Proof: **Security.** As discussed in the proof of correctness, the distributed replication model introduces a new round of distributed version decision round. In this phase, a client requests version numbers from either q_r or q_w. When $\mathcal{A}$ cannot see the content of the messages, it cannot differentiate whether the request is for a read or a write, since $|q_r| = |q_w|$. Therefore, Adv that $\mathcal{A}$ gains is negligible during this process. Intuitively, the remaining of the distributed replication model works similarly to the centralized replication model. Given the centralized replication model is secure, the distributed replication model is secure as well. ■

6.5 Conclusion

In this chapter, we present, to the best of our knowledge, the first study of fault-tolerance for oblivious data storage systems with a trusted proxy and untrusted cloud storage. Considering privacy as a first class system requirement, we introduce three generic quorum based replication models for different deployment scenarios to tolerate the failures of different sys-

tems components. We evaluate the trade-offs of selecting specific quorums, their applicability in the oblivious setting, the achieved fault-tolerance level, and design each replication model separately considering the privacy constraints specific to oblivious storage systems. Our contribution of bringing fault-tolerance in oblivious data storage is orthogonal to oblivious algorithms and can also be integrated with any proxy based oblivious data storage frameworks that are aaob-secure and linearizable. We prove that our models are correct and able to hide access patterns.

Part IV

Concluding Remarks

Chapter 7

Conclusion and Future Work

In this dissertation, we explore and demonstrate the possibility of providing high performance and functional data services outsourced to the cloud or any untrusted third party without compromising security and privacy. To achieve this goal, we analyze privacy and security requirements for each application separately and design an appropriate framework. We explicitly deal with the problems of *range query processing over encrypted data*, *privacy preserving data mining* in the context of environmental sustainability studies, and *access privacy* in the cloud.

To enable private range query execution, we introduce PINED-RQ, a highly efficient and differentially private range query execution framework that constructs a novel differentially private index over an outsourced database. Unlike other differentially private systems, PINED-RQ is extended to support update operations. To the best of our knowledge, PINED-RQ is the first work that builds, uses and maintains a differentially private index for performing *selection range queries*. We have demonstrated the security of PINED-RQ and shown empirically its practicality and efficiency through extensive experiments performed on synthetic and real datasets.

To better evaluate the environmental impacts of the industrial processes privately, we formally define *privacy preserving certification* paradigm. We additionally propose solutions for

two certification problems, mean and quantile, that give important insights to the practitioners about environmental impacts. To perform privacy preserving certifications without compromising any sensitive information, we propose a framework, which considers a realistic network communication model for the certification model, which enables a certifier to certify parties based on a well agreed upon set of criteria. Our simulation/prototype demonstrates that the proposed approach is not only secure but also efficient and practical. Additionally, we present a comprehensive study to explore the privacy concerns over publicizing the industrial activities in the form of LCA computations. This dissertation initiates a study to explore privacy and security challenges that prevent organizations from making public disclosures about their activities. Our empirical studies show that the application of privacy-preserving techniques is required to preserve the privacy of private data. Otherwise, it is possible to expose the private data by reverse-computing from the publication. To support the needs of the sustainability research community, this dissertation proposes differentially private LCA computations and explains how to achieve it for LCA computations by either perturbing the input data or the output data. Our evaluations on a real LCA example from a distillers grain study demonstrates that the use of differential privacy to publish more detailed information ensures strong privacy while revealing useful information for analysts.

This dissertation also highlights the importance of access privacy in the cloud setting. To this end, we design and develop TaoStore, a highly efficient and practical cloud data store, which secures data confidentiality and hides access patterns from adversaries. To the best of our knowledge, TaoStore is the first *tree-based* asynchronous oblivious cloud storage system. Additionally, we propose a new ORAM security model, called aaob-security, which considers completely asynchronous network communication and concurrent processing of requests. It is proven that TaoStore is secure and correct under this security model. Our experiments demonstrate the practicality and efficiency of TaoStore. To highlight the important security features of oblivious storage systems, we develop an educational game, *Guess the Access*. The goal is to

provide the database community with an opportunity to appreciate some of the intricate issues involved in the development and understanding of access security, specifically in a distributed cloud-based data management setting. This game should bridge the gap between the security and the database community, and help a database audience recognize the complexity of attacks that can be mounted on oblivious storage as well as the resulting significant overheads that truly secure oblivious stores require.

7.1 Future Research Directions

There are still many problems that need to be addressed in data security and privacy to increase the adoption of cloud for outsourcing databases. The integration of privacy and security techniques into current cloud services should have a low impact in terms of performance.

Towards enabling query processing over encrypted data, the existing full-fledged secure database systems like CryptDB [53] and MONOMI [54] still suffer from not being able to support same levels of data confidentiality in response to all queries. In the long run, the level of confidentiality provided is limited by the weakest encryption mechanism in these systems. There is still a great need for specialized encryption schemes that process specific tasks without compromising security and privacy. Fully homomorphic encryption is capable of executing arbitrary functions over the encrypted data. However, in practice, fully homomorphic encryption is very impractical which decreases its chances for being deployed on real cloud applications in the near future. On the other hand, recently proposed *functional encryption* scheme, which has taken a reasonable interest in the cryptography community, shows some promises in developing specialized functions. The advances in this area would directly be incorporated into the current systems and execute some special functions without sacrificing the confidentiality of the data.

For secure data management and query processing in the cloud, another approach would

be taking advantage of tamper-proof secure hardware. Secure hardware provides an isolated computation environment where an adversary cannot gain any information inside the black box even if she has the box physically. Currently, these devices have limited resources in terms of computation power and storage. However, it is not unlikely to expect better equipped secure hardware in the near future. The usage of secure hardware would allow both researchers and practitioners to develop more practical and functional data processing services in the cloud.

The recent works in building a secure index over the encrypted data to process range queries seems promising but it definitely needs drastic performance improvements. The industrial applications in the cloud are very demanding and they cannot tolerate executing a single query in the order of tens of seconds. The overhead of including security and privacy techniques should be low. Theoretical advances in cryptographic tools would make secure index construction and scanning faster.

Differential privacy is a very strong privacy notion and it has been received too much attention from the academia for the last decade. Despite its great success in the academia, the adoption of differential privacy in the industry is still very low. We believe that the main reason is the lack of understanding of differential privacy semantics. A way to overcome this issue would be to develop a comprehensive framework that allows practitioners to better understand the differential privacy semantics and its limitations. This framework will guide practitioners throughout the whole process from raw input to differentially private output (e.g., transforming of the structure of data or sensitivity adjustment).

Many cloud applications today rely on processing big data to deliver smarter results for richer user experience. This brings more revenue and more personalized service. To achieve this, machine learning is a standard field which explores algorithms that can learn from and make predictions on data. The success of machine learning algorithms relies on the success of modeling that processes the data and learns some characteristics out of it. The modeling process might reveal some sensitive information though. Therefore, the algorithms should

protect the privacy of records. A way would be to develop a machine learning framework that applies convenient machine learning techniques while preserving the privacy of data. If the confidentiality of data is also necessary, the new machine learning algorithms should work over encrypted data. Given the large body of applications of machine learning algorithms almost everywhere today, machine learning over encrypted data or machine learning with differential privacy would be emerging fields that might produce great products in the near feature.

Oblivious cloud storage systems are still in their early productions in the academia and the most of the advances in the fields are theoretical so far. TaoStore is one of the first end-to-end cloud storage implementations. Since it is open source, the next step would be deployed oblivious cloud storage systems on a small scale real world applications to better understand the practical concerns. The current practical limitations are not obvious. Moreover, high-performance multi-client oblivious cloud storage constructions use a proxy to improve the performance. However, the usage of proxy model limits deploying these systems on a large scale. Moving oblivious cloud storage systems one step further by enabling scalability would increase the chances of oblivious cloud storage systems to be deployed in the cloud setting. In addition, the oblivious data storage systems should be able to support similar features of traditional database storage systems like fault-tolerance and data partitioning.

Appendix A

TaoStore

A.1 Security of Asynchronous ORAM Schemes

This section develops a framework to analyze the security asynchronous ORAM schemes. We exercise this model to prove TaORAM secure.

Reactive systems. We consider a model of randomized interactive stateful reactive machines (sometimes simply called "algorithms"), which we only specify informally here, and which mimic the architecture running TaoStore. These machines have multiple interfaces, each with a given name.

The machines can activate at any time a *thread* by a certain input condition being met a certain interface (for example, a set of messages satisfying a certain condition have been input) and the corresponding messages are removed and input to the thread. During its execution, the thread can output messages at an interface, can set local variable and global variables (and can lock and unlock global variables), and can halt waiting for input messages to satisfy some condition to be re-started. Such threads can be run concurrently, and we do not make any assumptions about how thread executions are interleaved.

Such machines can then be combined with each other by connecting interfaces with the

The **storage server** SS is initialized with an array D of M items from T (which is kept as the state), exposes a *network* and an *adversarial* interface. It associates with every $\mathsf{bid} \in [M]$ a corresponding timestamp τ_{bid} – initially set to 0 – and operates as follows:

- At initialization, it outputs D at the adversarial interface.
- On input $\mathsf{op} = (\mathsf{bid}, u, \tau)$ at the *network interface*, the request is associated with a unique identifier id and $\mathsf{op} = \mathsf{op}_{\mathsf{id}}$ is added to the *input buffer*. The message $(\texttt{input}, \mathsf{id}, \mathsf{bid}, u, \tau)$ is output at the adversarial interface.
- On input $(\texttt{process}, \mathsf{id})$ at the adversarial interface, then $\mathsf{op}_{\mathsf{id}} = (\mathsf{bid}, u, \tau)$ is removed from the input buffer. We then set $v_{\mathsf{id}} = D[\mathsf{bid}]$ and if $u \neq \perp$, also sets $D[\mathsf{bid}] = u$ if $\tau_{\mathsf{bid}} < \tau$ (and update τ_{bid} to τ). The value v_{id} is added to the *output buffer* and returned at the adversarial interface.
- On input $(\texttt{output}, \mathsf{id})$ at the adversarial interface, the value v_{id} is removed from the output buffer, and output at the network interface.

Figure A.1: The storage server functionality SS.

same name. (We can think of a combination of such machines as a network of machines, but also as a bigger machines.) Consistent with literature on cryptography and asynchronous systems, we do not assume a global clock: When a thread halts waiting for a message, it does not learn how long it has been waiting.

Asynchronous ORAM. An **asynchronous ORAM scheme** is a pair $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$ consisting of the two following algorithms:

1. The **encoding algorithm** Encode on input a data set D (i.e., an array of N items from a set S), outputs a processed data set $\hat{D}$ and a secret key K. Here, $\hat{D}$ is an array of $M = M(N)$ elements from a set T.

2. The **ORAM client** $\mathsf{OClient}$ is initiated with the secret key K, as well as M and N. It maintain two interfaces: The **user interface** receives read/write requests (bid_i, u_i), where $\mathsf{bid}_i \in [N]$ is a logical address for the data set and $u_i \in S \cup \{\perp\}$ a data item. These

requests are eventually answered by a value $v_i \in S$. The **network interface**, OClient issues server read/write requests of form $(\mathsf{bid}_j, u_j, \tau)$, where $\mathsf{bid}_j \in [M]$, $u_i \in T \cup \{\perp\}$, and $\tau \in \mathbf{N}$, and which are eventually answered with a value $v_i \in T$.

The (finite) sets S and T denote the data types of the items held by the ORAM data structure and the storage server, respectively. Formally, all algorithms take as input a security parameter λ in unary form, and the sets S and T may depend on this security parameter. We omit mentioning λ explicitly for ease of notation. We also stress that in contrast to our algorithm descriptions in the body of the paper, for notational compactness here we think of OClient as answering a single type of read-write operation – i.e., (bid, u) simply retrieves the value of block bid if $u = \perp$, and additionally overwrites it with u if $u \neq \perp$.

Our scheme TaORAM can naturally be expressed in this framework. Here, the set S would correspond to individual data items addressed by bid, whereas T would correspond to bit-strings representing encrypted blocks.

Adaptive security. Our security definition, which we refer to as *adaptive asynchronous obliviousness*, or aaob-security, is *indistinguishability* based. In contrast to existing security notions – which are typically non-adaptive – our definition allows for adaptive scheduling of operations and messages. In particular, we model the non-deterministic nature of scheduling messages in the communication between the server and the client by leaving the scheduling task to the adversary $\mathcal{A}$. To achieve this, the security game involves a *storage server* SS, which is initially given an array of M elements from some set T, and exposes a *network* interface and an *adversarial* interface. It operates as described in Figure A.1. In particular, beyond its natural functionality at the network interface, the adversarial interface leaks the contents of read/write accesses and allows control of their scheduling.

For an asynchronous ORAM scheme $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$ and an adversary $\mathcal{A}$, we define the experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ as in Figure A.2. We can then define the aaob-*advantage*

Experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$:

- Initially, a challenge bit $b \xleftarrow{\$} \{0,1\}$ is chosen uniformly at random.
- The adversary $\mathcal{A}$, given no input, outputs two data sets D_0, D_1, each with N items.
- Then, $(\widehat{D}, K) \leftarrow \mathsf{Encode}(D)$ is computed, and we give $\widehat{D}$ and K as initial inputs to the server SS and to the client OClient, respectively.
- After that, the adversary $\mathcal{A}$ communicates with the adversarial interface of SS. Also, the network interfaces of OClient and SS are connected with each other. Finally, at any point in time, $\mathcal{A}$ can output a pair of operations $(\mathsf{op}_{i,0}, \mathsf{op}_{i,1})$, and the operation $\mathsf{op}_{i,b}$ is forwarded to the user interface of OClient.
- When each operation terminates and a reply is given at OClient's user interface, the adversary $\mathcal{A}$ is going to be notified (however, it does *not* learn the result of the operation). Note that leaking which value is returned by the operation can lead to easy distinguishability.
- Finally, $\mathcal{A}$ outputs a guess b'. If $b = b'$, the experiment returns `true`, and `false` otherwise.

Figure A.2: Experiment for aaob-security definition.

of the adversary $\mathcal{A}$ against ORAM as

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

We stress that the adversary schedules concurrent operation pairs – previous operations do not need to have returned (and thus $\mathcal{A}$ has been notified) before other operations are scheduled by $\mathcal{A}$.

Definition A.1.1 (ORAM Security) *We say that am ORAM Protocol* $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$ *is* aaob**-secure** *(or simply secure) if* $\mathsf{Adv}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ *is negligible for every polynomial-time adversary* $\mathcal{A}$*.*

We note that aaob-security in particular implies[1] security according to Definition 1 in [37], which has adversaries issue a fixed sequence of operations with fixed timings.

A.2 Security of TaORAM

We now prove the following theorem, assuming that the underlying encryption scheme satisfies the traditional notion of (secret-key) IND-CPA security [93, 170].

Theorem A.2.1 (Security) *Assume that the underlying encryption scheme is IND-CPA secure, then TaORAM is secure.*

Proof: [Proof (Sketch)] The proof is more involved than for traditional, non-concurrent, ORAM schemes. We omit a complete formal proof for lack of space. However, we outline the main steps necessary for the formal argument to go through, which in particular explains the central role played by the sequencer.

Specifically, we note the following central properties of TaORAM:

- Every operation op to OClient results in the Processor immediately starting a thread retrieving the contents of exactly one fresh random tree-path $\mathsf{pid}_{\mathsf{op}}$ from the server. This is regardless of the type of operation issued, or whether fake.read is set or not. The adversary can then schedule OClient's requests as it wishes.

- The processor never replies to an operation *before* the whole contents of $\mathsf{pid}_{\mathsf{op}}$ have been received from the storage server, and never replies *after* the last path $\mathsf{pid}_{\mathsf{op}'}$ associated with an operation op' preceding op in sequencer.queue is completely retrieved.

[1] Formally speaking, their definition allows the choice of the scheduling of operations to be fixed according to some absolute clock. Following the cryptographic literature here we omit access to an absolute clock, and parties have only accesses to logical sequences of events. We note that [37] does not include a formal model.

- The sequencer replies to an operation request op immediately after $\mathsf{pid}_{\mathsf{op}}$ and all paths $\mathsf{pid}_{\mathsf{op}'}$ associated with operations op′ preceding op in sequencer.queue have been completely retrieved.

- Write backs occur after a fixed number of paths have been retrieved, independently of the actual operations having been issued, and consists of fresh encryptions.

The above four items imply that the *communication* patterns are oblivious: The view of the adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ does *not* depend on the actual choice of the challenge bit b, when the adversary cannot see the contents of the messages sent over the network. In particular, $\mathcal{A}$ can see explicitly the mapping between op and the path $\mathsf{pid}_{\mathsf{op}}$, and $\mathcal{A}$'s decision on when the contents of the path are given back to OClient completely determines the timings of the responses.

Given this, we note that the case $b = 0$ and $b = 1$ cannot be distinguished even given the contents of the messages and the storage server. To show this, the proof first replaces every encrypted block (either in a message or on the server) with a fresh encryption of a dummy block (e.g., the all-zero block). This does not affect the adversary's aaob advantage much by IND-CPA security of the underlying encryption scheme, and the fact that the adversary never sees the actual responses to its operations. Given now that the encrypted contents can be simulated and are independent of the actual operations issued, we can now apply the above argument showing that the actual access patterns are indistinguishable. ■

A.3 Histories, Linearizability, and Correctness

We note that security of an asynchronous ORAM scheme as defined above does not imply its correctness – one can just have the client do nothing (i.e., not sending any message to a server) and immediately reply requests with random contents, and have a secure scheme. For

this reason, we handle correctness separately and show that our TaORAM satisfies very strong correctness guarantees, and in particular provides so-called *atomic* semantics of the underlying storage from a user-perspective. This means that every operation appears to have taken place atomically at some point between the request and the answer is provided. To formalize this notion, we follow the tradition of the literature on distributed systems and consistency semantics. We start with some definitions.

To reason about correctness, let us think of a variation of Experiment $\mathsf{Exp}^{\mathsf{aaob}}_{\mathsf{ORAM}}(\mathcal{A})$ defined above where the reply to each adversarial request is actually given back to the adversary, and moreover, we do not have a challenge bit any more. More formally, we define $\mathsf{Exp}^{\mathsf{corr}}_{\mathsf{ORAM}}(\mathcal{A})$ as the following experiment, with no output:

Experiment $\mathsf{Exp}^{\mathsf{corr}}_{\mathsf{ORAM}}(\mathcal{A})$:

- The adversary $\mathcal{A}$, given no input, outputs a data set D with N items.
- Then, $(\widehat{D}, K) \leftarrow \mathsf{Encode}(D)$ is computed, and we give $\widehat{D}$ and K as initial inputs to the server SS and to the client $\mathsf{OClient}$, respectively.
- After that, the adversary $\mathcal{A}$ communicates with the adversarial interface of SS. Also, the network interfaces of $\mathsf{OClient}$ and SS are connected with each other. Finally, at any point in time, $\mathcal{A}$ can output an operation op_i, which is forwarded to the user interface of $\mathsf{OClient}$.
- When each operation terminates and a reply is given at $\mathsf{OClient}$'s user interface, the adversary $\mathcal{A}$ is going to be notified and learns the outcome of the operation.

Recall that the client $\mathsf{OClient}$ processes requests of the form (bid_i, v_i), where v_i is either a data item (for an overwrite operation), or $v_i = \perp$ (for a read operation), and this operation is replied with a data item u_i. In an execution of the above experiment, we associate with every request a unique *operation identifier* $i \in \mathbf{N}$ in increasing order, with the goal of paring it with the

corresponding reply.

A **history** Hist consists of the initial data set D, as well as a sequence of items of the form $\mathsf{req}_i = (\mathsf{bid}_i, v_i)$ and $\mathsf{rep}_i = u_i$, such that every occurrence of some item $\mathsf{rep}_i = u_i$ is preceded by a (unique) element $\mathsf{req}_i = (\mathsf{bid}_i, v_i)$ with the same identifier i. We say that a history is *partial* if there exists $\mathsf{req}_i = (\mathsf{bid}_i, v_i)$ without a corresponding $\mathsf{rep}_i = u_i$, and otherwise it is *complete*. An execution of $\mathsf{Exp}^{\mathsf{corr}}_{\mathsf{ORAM}}(\mathcal{A})$ naturally generates a history at the user interface of OClient, where the sequence of requests and responses corresponds to the point in time in which they were given as an input to OClient by $\mathcal{A}$, and returned as an output to $\mathcal{A}$, respectively.

In a complete history Hist, we refer to the pair $(\mathsf{req}_i, \mathsf{rep}_i)$ as op_i (the i-th operation) and we say that op_i *precedes* op_j if and only if rep_i occurs before req_j. Also, we often write $\mathsf{op}_i = (\mathsf{bid}_i, u_i, v_i)$. We say that a complete history Hist is **linearizable** if there exists a total order $\leq_{\mathsf{lin}}$ over the operation identifiers such that: (1) If op_i precedes op_j, then $\mathsf{op}_i \leq_{\mathsf{lin}} \mathsf{op}_j$. (2) If $\mathsf{op}_i = (\mathsf{bid}_i, v_i, u_i)$, then either the largest $\mathsf{op}_j = (\mathsf{bid}_j, v_j, u_i)$ such that $\mathsf{op}_j \leq_{\mathsf{lin}} \mathsf{op}_i$ and $v_j \neq \perp$, if it exists, is such that $v_j = u_i$, or no such op_j exists and $D[\mathsf{bid}_i] = u_i$.

With the above definitions in place, we are ready to state the following definition.

Definition A.3.1 (Correctness) *An asynchronous ORAM scheme* $\mathsf{ORAM} = (\mathsf{Encode}, \mathsf{OClient})$ *is* **correct***, if for all adversaries $\mathcal{A}$ (even computationally unbounded ones) that deliver all messages, the history generated by* $\mathsf{Exp}^{\mathsf{corr}}_{\mathsf{ORAM}}(\mathcal{A})$ *is complete and linearizable, except with negligible probability.*

A.4 Correctness Proof for TaORAM

We apply the above definition to TaORAM.

Theorem A.4.1 (Correctness) *TaORAM is correct.*

Proof: For this analysis, we assume that memory never overflows, and thus the system will never crash or abort. (We discussed above that lack of memory overflows can be assumed without loss of generality.)

We show below that if $\mathcal{A}$ delivers all messages, then every history is complete at the end of the execution of $\mathsf{Exp}^{\mathsf{corr}}_{\mathsf{ORAM}}(\mathcal{A})$. The core of the proof is to show that the resulting complete history Hist is linearizable. This requires first defining the corresponding order $\leq_{\mathsf{lin}}$.

For every operation $\mathsf{op}_i = (\mathsf{bid}_i, v_i, u_i)$, there is a point in time t_i in which it takes effect in the global event sequence (we assume that every event is associated with a unique time). This is always within ANSWER-REQUEST in the execution of Item 3. In particular, an operation $\mathsf{op}_i = (\mathsf{bid}_i, v_i, u_i)$ takes effect when it is popped from the queue $\mathsf{request.map}[\mathsf{bid_i}]$. (Note that this may be within a thread running ANSWER-REQUEST for another operation op_j for which $\mathsf{bid}_j = \mathsf{bid}_i$.) We order two operations $\mathsf{op}_i = (\mathsf{bid}_i, v_i, u_i)$ and $\mathsf{op}_j = (\mathsf{bid}_j, v_j, u_j)$ so that $\mathsf{op}_i \leq_{\mathsf{lin}} \mathsf{op}_j$ if op_i takes effect before op_j. Clearly, if op_i precedes op_j, then $\mathsf{op}_i \leq_{\mathsf{lin}} \mathsf{op}_j$, since every operation takes effect between the request and the response.

During the execution of TaORAM, we can track the contents of the local storage, and we are going to prove the following invariant:

> **Invariant.** At every point in time, there exists at most one value B_{bid} for the block bid in the local storage (sub-tree or stash). Moreover, this value is the latest value assigned to bid according to the "take-effect" order defined above (or the initial value, if no such value exists).

Note that before returning a value u for an operation on bid, we must have set the local value B_{bid} before returning $B_{\mathsf{bid}}[\mathsf{bid}]$, and thus the above implies that $\leq_{\mathsf{lin}}$ is a proper ordering to show that the history is linearizable.

To prove the invariant, we proceed by induction over steps that can modify the contents of the local storage. The invariant is true when the system has been initialized, and the client's

local memory is empty. The following operations can modify the contents of the local storage (here, a pair $(\mathsf{bid}, B_{\mathsf{bid}})$ in the local storage simply denotes a pointer to block bid and the actual contents of the block).

1. A pair $(\mathsf{bid}, B_{\mathsf{bid}})$ is added to the local storage as part of some node w through processing of some path pid in Step 1 of ANSWER-REQUEST.

2. A pair $(\mathsf{bid}, B_{\mathsf{bid}})$ is deleted at Step 5 of WRITE-BACK because it is on a path pid written back to the server.

3. A pair $(\mathsf{bid}, B_{\mathsf{bid}})$ is moved to a new location (either in the tree or into the stash) when shuffling within FLUSH

4. A pair $(\mathsf{bid}, B_{\mathsf{bid}})$ is present in the local storage, and we assign B_{bid} to some new value v, in the third item of Step 3 of ANSWER-REQUEST.

Clearly, 3–4 do not violate the invariant. As for 2, if B_{bid} has been modified after it has been written to the server, then it will not be deleted due to the node timestamp being now higher than $v \cdot k$. If it is deleted, then no modification has occurred since the write-back has started, and thus the server holds the latest version.

The core of the proof is showing that 1 cannot violate the invariant, which we do next. In fact, we prove now that if at some time t^* the invariant has been true so far, and we now insert $(\mathsf{bid}, B_{\mathsf{bid}})$ as part of the contents of a node N, then this is the latest value of bid and no other value for bid appears in the local storage at this point in time t^*.

First off, if this is the initial value written by the Encode procedure into the server, and it gets written into node N, and $(\mathsf{bid}, B_{\mathsf{bid}})$ was never locally in node N, then the value of bid was never modified locally, because we need to retrieve it from the server at least once for the first change to take effect. Therefore, we can assume that $(\mathsf{bid}, B_{\mathsf{bid}})$ was already once earlier in the local storage at node N, either because it was written back from there (if this is not the

initial value), or because we need to retrieve it at least once if this is the initial value and some modification has taken place. Now, consider the time $t \leq t^*$ at which $(\mathsf{bid}, B_{\mathsf{bid}})$ was in N for the last time. Note that if $t = t^*$, then the value would not be overwritten (as the node N is occupied in the local storage) and by the induction assumption this node holds the latest value. Therefore, assume that $t < t^*$, and we have two cases.

The first (and more difficult) case is that, at time t, $(\mathsf{bid}, B_{\mathsf{bid}})$ left N, and was possibly modified one or more times. In this case, we show that the local storage is *not* updated because the node N is already occupied with some pre-existing contents. The important observation here is that if there are one or more completed write-backs between t and t^*, the node N is never deleted after the write back completed. If it left N, then N was modified, and a write-back terminating after t would not delete N *unless* it just wrote this new contents of N back (or an even newer version). But this means that at that point we have already overwritten the contents of N *on the server* with something different than what received within pid (i.e., where in particular $(\mathsf{bid}, B_{\mathsf{bid}})$ would not be in N any more). Hence, the contents $(\mathsf{bid}, B_{\mathsf{bid}})$ of N received with pid must have been sent by the server before the new contents have been written (this is ensured by our server time stamping), and thus when this write-back completes, we have $\mathsf{pid} \in \mathsf{PathReqMultiSet}$, and hence N is left untouched and unmodified.

The second case is that, at time t, $(\mathsf{bid}, B_{\mathsf{bid}})$ was deleted after a successful write back completed. As this was the last time $(\mathsf{bid}, B_{\mathsf{bid}})$ ever appeared in N before t^* it cannot be that any operation to effect on bid between t and t^*, and thus the value re-covered with pid is the latest one.

We still need to show that every operation eventually terminates, and thus every history is eventually completed. We first show that the Processor Module replies to every request. Note that if all messages are delivered by $\mathcal{A}$, the wait instructions in READ-PATH always terminates, and the thread is waken up. Therefore, every retrieved path is eventually received by the client.

Now there are two cases, for the thread executed for an operation accessing bid_i – either it results in a fake read or not, i.e., the flag fake.read returned by READ-PATH is either 1 or 0.

- *Case 1:* $\mathsf{fake.read} = 0$: Here, we know that the path P contains bid, and when executing ANSWER-REQUEST, either the entry in response.map for this operation has form $(\texttt{false}, x)$ for $x \neq \perp$, then the operation is answered right away in Step 2. Alternatively, if $x = \perp$, because the block is in the path P, this query must be replied later in Step 3.

- *Case 2:* $\mathsf{fake.read} = 1$. Then, this means that while executing READ-PATH in the main thread T, another thread T' has invoked READ-PATH for the same bid_i without returning $\mathsf{fake.read} = 1$, and thread T' has not yet gone through Step 3 in ANSWER-REQUEST. Now, there are two cases. Either T' will update the value for the current request in response.map in Step 3 of ANSWER-REQUEST before T goes though Step 2 in its own ANSWER-REQUEST, in which case T will return the value. Alternatively, if T goes through Step 2 first, the value will be output when T' goes through Step 3.

Finally note that the sequencer module may delay answering, but the above argument implies that the processor eventually answers all previous requests, and thus the sequencer will also eventually answer them all. ■

Bibliography

[1] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, *Controlling data in the cloud: Outsourcing computation without outsourcing control*, in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, (New York, NY, USA), pp. 85–90, ACM, 2009.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, *Order preserving encryption for numeric data*, in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, (New York, NY, USA), pp. 563–574, ACM, 2004.

[3] D. X. Song, D. Wagner, and A. Perrig, *Practical techniques for searches on encrypted data*, in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, (Washington, DC, USA), pp. 44–, IEEE Computer Society, 2000.

[4] Y.-C. Chang and M. Mitzenmacher, *Privacy preserving keyword searches on remote encrypted data*, in *Applied Cryptography and Network Security: Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005. Proceedings* (J. Ioannidis, A. Keromytis, and M. Yung, eds.), (Berlin, Heidelberg), pp. 442–455, Springer Berlin Heidelberg, 2005.

[5] Z. Yang, S. Zhong, and R. N. Wright, *Privacy-preserving queries on encrypted data*, in *Computer Security – ESORICS 2006: 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006. Proceedings* (D. Gollmann, J. Meier, and A. Sabelfeld, eds.), (Berlin, Heidelberg), pp. 479–495, Springer Berlin Heidelberg, 2006.

[6] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, *Executing sql over encrypted data in the database-service-provider model*, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, (New York, NY, USA), pp. 216–227, ACM, 2002.

[7] B. Hore, S. Mehrotra, and G. Tsudik, *A privacy-preserving index for range queries*, in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pp. 720–731, VLDB Endowment, 2004.

[8] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, and R. Canetti, *Modular order-preserving encryption, revisited*, in *Proceedings of the 2015 ACM SIGMOD*

International Conference on Management of Data, SIGMOD '15, (New York, NY, USA), pp. 763–777, ACM, 2015.

[9] T. Ge and S. Zdonik, *Answering aggregation queries in a secure system model*, VLDB '07, pp. 519–530, VLDB Endowment, 2007.

[10] C. Gentry, *Fully homomorphic encryption using ideal lattices*, STOC '09, pp. 169–178, ACM, 2009.

[11] J. Katz, A. Sahai, and B. Waters, *Predicate encryption supporting disjunctions, polynomial equations, and inner products*, EUROCRYPT '08, pp. 146–162, Springer-Verlag, 2008.

[12] B. Schneier, *Homomorphic encryption breakthrough*, 2009. `http://www.schneier.com/blog/archives/2009/07/homomorphic_enc.html`, 2009.

[13] L. H. Cox, *Suppression methodology and statistical disclosure control*, *Journal of the American Statistical Association* **75** (1980), no. 370 377–385, [http://www.tandfonline.com/doi/pdf/10.1080/01621459.1980.10477481].

[14] T. Dalenius, *Finding a needle in a haystack-or identifying anonymous census record*, *Journal of official statistics* **2** (1986), no. 3 329–336.

[15] L. Sweeney, *Achieving k-anonymity privacy protection using generalization and suppression*, *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* **10** (Oct., 2002) 571–588.

[16] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, *Incognito: Efficient full-domain k-anonymity*, in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, (New York, NY, USA), pp. 49–60, ACM, 2005.

[17] P. Samarati, *Protecting respondents? identities in microdata release*, *IEEE Transactions on Knowledge and Data Engineering* **13** (2001) 1010–1027.

[18] L. Sweeney, *k-anonymity: A model for protecting privacy*, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10** (2002), no. 5 557–570.

[19] R. J. Bayardo and R. Agrawal, *Data privacy through optimal k-anonymization*, in *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, (Washington, DC, USA), pp. 217–228, IEEE Computer Society, 2005.

[20] X. Xiao and Y. Tao, *Anatomy: Simple and effective privacy preservation.*, in *VLDB* (U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim, eds.), pp. 139–150, ACM, 2006.

[21] Q. Zhang, N. Koudas, D. Srivastava, and T. Yu, *Aggregate query answering on anonymized tables*, in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 116–125, April, 2007.

[22] J. Domingo-Ferrer and V. Torra, *A Critique of k-Anonymity and Some of Its Enhancements*, in *Third International Conference on Availability, Reliability and Security (ARES 08)*, pp. 990–993, IEEE, 2008.

[23] R. Brand, *Microdata protection through noise addition.*, in *Inference Control in Statistical Databases* (J. Domingo-Ferrer, ed.), vol. 2316 of *Lecture Notes in Computer Science*, pp. 97–116, Springer, 2002.

[24] D. Agrawal and C. C. Aggarwal, *On the design and quantification of privacy preserving data mining algorithms*, in *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, (New York, NY, USA), pp. 247–255, ACM, 2001.

[25] P. Samarati and L. Sweeney, *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*, in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1998.

[26] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, *L-diversity: Privacy beyond k-anonymity*, *ACM Trans. Knowl. Discov. Data* **1** (Mar., 2007).

[27] N. Li, T. Li, and S. Venkatasubramanian, *t-closeness: Privacy beyond k-anonymity and l-diversity*, in *IEEE 23rd International Conference on Data Engineering, ICDE 2007.*, pp. 106–115, April, 2007.

[28] C. Dwork, *Differential privacy*, in *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II* (M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, eds.), (Berlin, Heidelberg), pp. 1–12, Springer Berlin Heidelberg, 2006.

[29] Y. Lindell and B. Pinkas, *Secure multiparty computation for privacy-preserving data mining*, *IACR Cryptology ePrint Archive* **2008** (2008) 197.

[30] M. S. Islam, M. Kuzu, and M. Kantarcioglu, *Access pattern disclosure on searchable encryption: Ramification, attack and mitigation*, in *NDSS 2012*, (San Diego, California, USA), The Internet Society, Feb. 5–8,, 2012.

[31] O. Goldreich, *Towards a theory of software protection*, in *CRYPTO '86*, pp. 426–439, Springer-Verlag, 1987.

[32] O. Goldreich and R. Ostrovsky, *Software protection and simulation on oblivious rams*, *J. ACM* **43** (May, 1996) 431–473.

[33] D. Boneh, D. Mazieres, and R. A. Popa, *Remote oblivious storage: Making oblivious ram practical*, tech. rep., MIT, 2011. MIT Tech-report: MIT-CSAIL-TR-2011-018.

[34] J. R. Lorch, B. Parno, J. Mickens, M. Raykova, and J. Schiffman, *Shroud: Ensuring private access to large-scale data in the data center*, in *the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, (San Jose, CA), pp. 199–213, USENIX, 2013.

[35] P. Williams, R. Sion, and A. Tomescu, *PrivateFS: a parallel oblivious file system*, in *ACM CCS 12* (T. Yu, G. Danezis, and V. D. Gligor, eds.), (Raleigh, NC, USA), pp. 977–988, ACM Press, Oct. 16–18,, 2012.

[36] E. Stefanov, E. Shi, and D. X. Song, *Towards practical oblivious RAM*, in *NDSS 2012*, (San Diego, California, USA), The Internet Society, Feb. 5–8,, 2012.

[37] E. Stefanov and E. Shi, *ObliviStore: High performance oblivious cloud storage*, in *2013 IEEE Symposium on Security and Privacy*, (Berkeley, California, USA), pp. 253–267, IEEE Computer Society Press, May 19–22,, 2013.

[38] 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 *ACM CCS 15* (I. Ray, N. Li, and C. Kruegel:, eds.), (Denver, CO, USA), pp. 837–849, ACM Press, Oct. 12–16,, 2015.

[39] G. Rebitzer, T. Ekvall, R. Frischknecht, D. Hunkeler, G. Norris, T. Rydberg, W. P. Schmidt, S. Suh, B. P. Weidema, and D. W. Pennington, *Life cycle assessment: Part 1: Framework, goal and scope definition, inventory analysis, and applications*, *Environ. Int.* **30** (July, 2004) 701–720.

[40] ISO 14044, *Environmental management — Life cycle assessment — Requirements and guidelines*. ISO, Geneva, Switzerland, 2006.

[41] A. Boldyreva, N. Chenette, and A. O'Neill, *Order-preserving encryption revisited: Improved security analysis and alternative solutions*, *IACR Cryptology ePrint Archive* **2012** (2012) 625.

[42] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, *Fast range query processing with strong privacy protection for cloud computing*, *PVLDB* **7** (2014), no. 14 1953–1964.

[43] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, *Practical private range search revisited*, in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), pp. 185–198, ACM, 2016.

[44] O. Goldreich, *Foundations of cryptography: a primer*, *Found. Trends in Theoretical Computer Science* **1** (April, 2005) 1–116.

[45] C. Dwork and A. Roth, *The algorithmic foundations of differential privacy*, *Found. Trends Theor. Comput. Sci.* **9** (Aug., 2014) 211–407.

[46] F. D. McSherry, *Privacy integrated queries: An extensible platform for privacy-preserving data analysis*, in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, (New York, NY, USA), pp. 19–30, ACM, 2009.

[47] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating noise to sensitivity in private data analysis*, in *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, (Berlin, Heidelberg), pp. 265–284, Springer-Verlag, 2006.

[48] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, *Boosting the accuracy of differentially private histograms through consistency*, *Proc. VLDB Endow.* **3** (Sept., 2010) 1021–1032.

[49] N. Li, W. Yang, and W. Qardaji, *Differentially private grids for geospatial data*, in *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, (Washington, DC, USA), pp. 757–768, IEEE Computer Society, 2013.

[50] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett, *Differentially private histogram publication*, *VLDB J.* **22** (2013), no. 6 797–822.

[51] W. Qardaji, W. Yang, and N. Li, *Understanding hierarchical methods for differentially private histograms*, *Proc. VLDB Endow.* **6** (Sept., 2013) 1954–1965.

[52] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu, *Differentially private spatial decompositions*, in *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, (Washington, DC, USA), pp. 20–31, IEEE Computer Society, 2012.

[53] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, *Cryptdb: Protecting confidentiality with encrypted query processing*, in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, (New York, NY, USA), pp. 85–100, ACM, 2011.

[54] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, *Processing analytical queries over encrypted data*, in *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pp. 289–300, VLDB Endowment, 2013.

[55] X. Zhang, X. Meng, and R. Chen, *Differentially private set-valued data release against incremental updates*, in *Database Systems for Advanced Applications: 18th International Conference, DASFAA 2013, Wuhan, China, April 22-25, 2013. Proceedings, Part I* (W. Meng, L. Feng, S. Bressan, W. Winiwarter, and W. Song, eds.), pp. 392–406, Springer Berlin Heidelberg, 2013.

[56] E. Cho, S. A. Myers, and J. Leskovec, *Friendship and mobility: User movement in location-based social networks*, in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, (New York, NY, USA), pp. 1082–1090, ACM, 2011.

[57] "US Postal Employees." Data Universe, Asbury Park Press. `http://php.app.com/agent/postalemployees/search`.

[58] X. Xiao, G. Wang, and J. Gehrke, *Differential privacy via wavelet transforms*, vol. 23, (Piscataway, NJ, USA), pp. 1200–1214, IEEE Educational Activities Department, Aug., 2011.

[59] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, *Secure multidimensional range queries over outsourced data*, *The VLDB Journal* **21** (2012), no. 3 333–358.

[60] C. Li, M. Hay, G. Miklau, and Y. Wang, *A data- and workload-aware query answering algorithm for range queries under differential privacy*, *PVLDB* **7** (2014), no. 5 341–352.

[61] E.-J. Goh, "Secure indexes." Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[62] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, *Differential privacy under continual observation*, in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, (New York, NY, USA), pp. 715–724, ACM, 2010.

[63] T.-H. H. Chan, E. Shi, and D. Song, *Private and continual release of statistics*, *ACM Trans. Inf. Syst. Secur.* **14** (Nov., 2011) 26:1–26:24.

[64] D. Morrow and D. Rondinelli, *Adopting corporate environmental management systems:*, *European Management Journal* **20** (Apr, 2002) 159–171.

[65] M. Delmas and V. D. Blass, *Measuring corporate environmental performance: the trade-offs of sustainability ratings*, *Bus. Strat. Env.* **19** (Apr, 2010) 245–260.

[66] J. Kaenzig, D. Friot, M. Saad, M. Margni, and O. Jolliet, *Using life cycle approaches to enhance the value of corporate environmental disclosures*, *Bus. Strat. Env.* **20** (Dec, 2010) 38–54.

[67] E. Heiskanen, *The institutional logic of life cycle thinking*, *Journal of Cleaner Production* **10** (Oct, 2002) 427–437.

[68] K. Lee and H. Stensel, *ISO standards on environmental labels and declarations and its implications on the market*, *Proceedings First International Symposium on Environmentally Conscious Design and Inverse Manufacturing* (1999).

[69] A. Del Borghi, *LCa and communication: Environmental product declaration*, *The International Journal of Life Cycle Assessment* **18** (Oct, 2012) 293–295.

[70] A. M. Fet and C. Skaar, *Eco-labeling, product category rules and certification procedures based on ISO 14025 requirements (6 pp)*, *The International Journal of Life Cycle Assessment* **11** (Jan, 2006) 49–54.

[71] B. R. Beloff, J. M. Schwarz, and E. Beaver, *Use sustainability metrics to guide decision-making*, *Chemical Engineering Progress* **98** (July, 2002) 58–63.

[72] G. Wernet, S. Papadokonstantakis, S. Hellweg, and K. Hungerbhler, *Bridging data gaps in environmental assessments: Modeling impacts of fine and basic chemical production*, *Green Chem.* **11** (2009), no. 11 18–26.

[73] UNEP/SETAC, *Global guidance principles for life cycle assessment databases*, tech. rep., United Nations Environment Programme, 2011.

[74] K. Nakano and M. Hirao, *Collaborative activity with business partners for improvement of product environmental performance using LCA*, *Journal of Cleaner Production* **19** (Jul, 2011) 1189–1197.

[75] F. Kerschbaum, J. Strüker, and T. G. Koslowski, *Confidential information-sharing for automated sustainability benchmarks*, in *Proceedings of the International Conference on Information Systems, ICIS 2011, Shanghai, China, December 4-7, 2011*, 2011.

[76] *IBM 4764 product and PCIXCC feature overview*, Mar., 2012. `https://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml`.

[77] S. Bajaj and R. Sion, *Trusteddb: a trusted hardware based database with privacy and data confidentiality*, in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pp. 205–216, 2011.

[78] F. Baldimtsi and O. Ohrimenko, *Sorting and searching behind the curtain*, in *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pp. 127–146, 2015.

[79] J. Katz, *Universally composable multi-party computation using tamper-proof hardware*, in *Advances in Cryptology-EUROCRYPT 2007*, pp. 115–128. Springer, 2007.

[80] A. C. Yao, *Protocols for secure computations*, in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, (Washington, DC, USA), pp. 160–164, IEEE Computer Society, 1982.

[81] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, *et. al.*, *Fairplay-secure two-party computation system.*, in *USENIX Security Symposium*, vol. 4, San Diego, CA, USA, 2004.

[82] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, *Machine learning classification over encrypted data*, in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[83] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, *On private scalar product computation for privacy-preserving data mining*, in *Information Security and Cryptology–ICISC 2004*, pp. 104–120. Springer, 2005.

[84] F. Kerschbaum, *Practical privacy-preserving benchmarking*, in *Proceedings of The IFIP TC-11 23rd International Information Security Conference, IFIP 20th World Computer Congress, IFIP SEC 2008, September 7-10, 2008, Milano, Italy*, pp. 17–31, 2008.

[85] F. Kerschbaum, *Secure and sustainable benchmarking in clouds - A multi-party cloud application with an untrusted service provider*, *Business & Information Systems Engineering* **3** (2011), no. 3 135–143.

[86] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk, *Rmind: a tool for cryptographically secure statistical analysis*, tech. rep., Cryptology ePrint Archive, Report 2014/512, 2014.

[87] M. Backes, A. Kate, M. Maffei, and K. Pecina, *Obliviad: Provably secure and practical online behavioral advertising*, in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, (Washington, DC, USA), pp. 257–271, IEEE Computer Society, 2012.

[88] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, *Policy-sealed data: A new abstraction for building trusted cloud services*, in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, (Bellevue, WA), pp. 175–188, USENIX, 2012.

[89] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, *Orthogonal security with cipherbase*, in *Proc. of the 6th CIDR, Asilomar, CA*, 2013.

[90] *Tpm main specification*, March, 2011. `http://www.trustedcomputinggroup.org/tpm-main-specification/`.

[91] *Building a high-performance, programmable secure coprocessor*, *Comput. Netw.* **31** (Apr., 1999) 831–860.

[92] P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, (Berlin, Heidelberg), pp. 223–238, Springer-Verlag, 1999.

[93] S. Goldwasser and S. Micali, *Probabilistic encryption*, *Journal of Computer and System Sciences* **28** (1984), no. 2 270–299.

[94] F. Kerschbaum and O. Terzidis, *Filtering for private collaborative benchmarking*, in *Emerging Trends in Information and Communication Security, International Conference, ETRICS 2006, Freiburg, Germany, June 6-9, 2006, Proceedings*, pp. 409–422, 2006.

[95] I. Damgård, M. Geisler, and M. Krøigaard, *Efficient and secure comparison for on-line auctions*, in *Information security and privacy*, pp. 416–430, Springer, 2007.

[96] I. Damgård, M. Geisler, and M. Krøigaard, *A correction to 'efficient and secure comparison for on-line auctions'*, *IJACT* **1** (2009), no. 4 323–324.

[97] F. Kerschbaum, D. Biswas, and S. de Hoogh, *Performance comparison of secure comparison protocols*, in *Database and Expert Systems Applications, DEXA, International Workshops, Linz, Austria, August 31-September 4, 2009, Proceedings*, pp. 133–136, 2009.

[98] T. Veugen, *Comparing encrypted data*, *Multimedia Signal Processing Group, Delft University of Technology, The Netherlands and TNO Information and Communication Technology, Delft, Tech. Rep* (2011).

[99] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, *Efficient garbling from a fixed-key blockcipher*, in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 478–492, IEEE, 2013.

[100] R. Heijungs and S. Suh, *The computational structure of life cycle assessment*, vol. 11. Springer Science & Business Media, 2002.

[101] B. Kuczenski, *Partial ordering of life cycle inventory databases*, *The International Journal of Life Cycle Assessment* **20** (Oct, 2015) 1673–1683.

[102] World Steel Association, *Life cycle inventory study for steel products*, tech. rep., World Steel Association, 2011.

[103] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, *Privacy-preserving public auditing for secure cloud storage*, *IEEE Trans. Comput.* **62** (Feb, 2013) 362–375.

[104] C. Baum, I. Damgrd, and C. Orlandi, *Publicly auditable secure multi-party computation*, *Security and Cryptography for Networks* (2014) 175–196.

[105] United Nations, "The millennium development goals report 2015." http://www.un.org/millenniumgoals/, 2015.

[106] T. O. Wiedmann, H. Schandl, M. Lenzen, D. Moran, S. Suh, J. West, and K. Kanemoto, *The material footprint of nations*, *Proceedings of the National Academy of Sciences* (2013) [http://www.pnas.org/content/early/2013/08/28/1220362110.full.pdf+html]. Early publication; doi:10.1073/pnas.1220362110.

[107] M. A. Curran, *Environmental Life-Cycle Assessment*. McGraw-Hill Professional Publishing, July, 1996.

[108] G. Finnveden, M. Z. Hauschild, T. Ekvall, J. Guinée, R. Heijungs, S. Hellweg, A. Koehler, D. Pennington, and S. Suh, *Recent developments in life cycle assessment*, *Journal of Environmental Management* **91** (2009), no. 1 1–21.

[109] R. G. Hunt and W. E. Franklin, *LCA — how it came about*, *International Journal of Life Cycle Assessment* **1** (1996), no. 1 4–7.

[110] R. Frischknecht, *Transparency in LCA-a heretical request?*, *Int J LCA* **9** (jul, 2004) 211–213.

[111] A.-M. Tillman, *Significance of decision-making for LCA methodology*, *Environ. Impact Assess. Rev.* **20** (2000), no. 1 113 – 123.

[112] B. P. Weidema, C. Bauer, R. Hischier, C. Mutel, T. Nemecek, J. Reinhard, C. Vadenbo, and G. Wernet, *Overview and methodology. data quality guideline for the ecoinvent database version 3*, tech. rep., The ecoinvent Centre, St. Gallen, 2013.

[113] M. Baitz, C. M. Colodel, T. Kupfer, J. Florin, O. Schuller, F. Hassel, M. Kokborg, A. Köhle, D. Thylmann, A. Stoffregen, S. Schöll, J. Görke, and M. Rudolf, *Gabi database & modelling principles 2013*, tech. rep., PE International AG, 2013.

[114] E. H. Moore, *On the reciprocal of the general algebraic matrix*, *Bulletin of the American Mathematical Society* **26** 394–395.

[115] G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solutions*, *Numerische mathematik* **14** (1970), no. 5 403–420.

[116] S. Chountasis, V. N. Katsikis, and D. Pappas, *Applications of the moore-penrose inverse in digital image restoration.*, *Mathematical Problems in Engineering* **2009** (2009) Article ID 170724, 12 p.–Article ID 170724, 12 p.

[117] R. Van de Walle, H. H. Barrett, K. J. Myers, M. Aitbach, B. Desplanques, A. F. Gmitro, J. Cornelis, and I. Lemahieu, *Reconstruction of mr images from data acquired on a general nonregular grid by pseudoinverse calculation*, *Medical Imaging, IEEE Transactions on* **19** (2000), no. 12 1160–1167.

[118] J. Steiner, R. Menezes, T. Ricci, and A. Oliveira, *Pca tomography: how to extract information from data cubes*, *Monthly Notices of the Royal Astronomical Society* **395** (2009), no. 1 64–75.

[119] J. R. Magnus, H. Neudecker, *et. al.*, *Matrix differential calculus with applications in statistics and econometrics*, ch. Kronecker products, the vec operator and the Moore-Penrose inverse. John Wiley & Sons, 1995.

[120] C. Dwork, K. Kenthapadi, F. McSherry, and I. Mironov, *Our data, ourselves: Privacy via distributed noise generation*, in *Advances in Cryptology (EUROCRYPT 2006)*, vol. 4004, (Saint Petersburg, Russia), pp. 486–503, Springer Verlag, May, 2006.

[121] *U.S. Life Cycle Inventory Database*, 2012. National Renewable Energy Laboratory, 2012. Accessed March 11, 2016: https://www.lcacommons.gov/nrel/search.

[122] J. Lee and C. Clifton, *How much is enough? choosing ϵ for differential privacy*, in *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, pp. 325–340, 2011.

[123] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth, *Differential Privacy: An Economic Method for Choosing Epsilon*, *arXiv:1402.3329 [cs]* (Feb., 2014). arXiv: 1402.3329.

[124] D. A. Belsey, E. Kuh, and R. E. Welsch, *"The condition Number". Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley, 1980.

[125] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, *Leakage-abuse attacks against searchable encryption*, in *ACM CCS 15* (I. Ray, N. Li, and C. Kruegel:, eds.), (Denver, CO, USA), pp. 668–679, ACM Press, Oct. 12–16,, 2015.

[126] 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 *ACM CCS 13* (A.-R. Sadeghi, V. D. Gligor, and M. Yung, eds.), (Berlin, Germany), pp. 299–310, ACM Press, Nov. 4–8,, 2013.

[127] K.-M. Chung and R. Pass, "A simple oram." Cryptology ePrint Archive, Report 2013/243, 2013. `http://eprint.iacr.org/`.

[128] K.-M. Chung, Z. Liu, and R. Pass, *Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead*, in *ASIACRYPT 2014, Part II* (P. Sarkar and T. Iwata, eds.), vol. 8874 of *LNCS*, (Kaoshiung, Taiwan, R.O.C.), pp. 62–81, Springer, Heidelberg, Germany, Dec. 7–11,, 2014.

[129] J. Dautrich, E. Stefanov, and E. Shi, *Burst oram: Minimizing oram response times for bursty access patterns*, in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 749–764, USENIX Association, Aug., 2014.

[130] 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)*, (Washington, D.C.), pp. 415–430, USENIX Association, Aug., 2015.

[131] S. Devadas, M. Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, *Onion ORAM: A constant bandwidth blowup oblivious RAM*, in *TCC 2016-A, Part II* (E. Kushilevitz and T. Malkin, eds.), vol. 9563 of *LNCS*, (Tel Aviv, Israel), pp. 145–174, Springer, Heidelberg, Germany, Jan. 10–13,, 2016.

[132] T. Moataz, T. Mayberry, and E. Blass, *Constant communication ORAM with small blocksize*, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015* (I. Ray, N. Li, and C. Kruegel, eds.), pp. 862–873, ACM, 2015.

[133] B. Pinkas and T. Reinman, *Oblivious RAM revisited*, in *CRYPTO 2010* (T. Rabin, ed.), vol. 6223 of *LNCS*, (Santa Barbara, CA, USA), pp. 502–519, Springer, Heidelberg, Germany, Aug. 15–19,, 2010.

[134] E. Kushilevitz, S. Lu, and R. Ostrovsky, *On the (in)security of hash-based oblivious RAM and a new balancing scheme*, in *23rd SODA* (Y. Rabani, ed.), (Kyoto, Japan), pp. 143–156, ACM-SIAM, Jan. 17–19,, 2012.

[135] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, *Oblivious storage with low I/O overhead*, *CoRR* **abs/1110.1851** (2011).

[136] M. T. Goodrich and M. Mitzenmacher, *Privacy-preserving access of outsourced data via oblivious RAM simulation*, in *ICALP 2011, Part II* (L. Aceto, M. Henzinger, and J. Sgall, eds.), vol. 6756 of *LNCS*, (Zurich, Switzerland), pp. 576–587, Springer, Heidelberg, Germany, July 4–8,, 2011.

[137] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, *Privacy-preserving group data access via stateless oblivious RAM simulation*, in *23rd SODA* (Y. Rabani, ed.), (Kyoto, Japan), pp. 157–167, ACM-SIAM, Jan. 17–19,, 2012.

[138] M. T. Goodrich, *Randomized shellsort: A simple oblivious sorting algorithm*, in *21st SODA* (M. Charika, ed.), (Austin, Texas, USA), pp. 1262–1277, ACM-SIAM, Jan. 17–19,, 2010.

[139] P. Williams, R. Sion, and M. Sotáková, *Practical oblivious outsourced storage*, *ACM Trans. Inf. Syst. Secur.* **14** (2011), no. 2 20.

[140] R. Ostrovsky and V. Shoup, *Private information storage (extended abstract)*, in *29th ACM STOC*, (El Paso, Texas, USA), pp. 294–303, ACM Press, May 4–6,, 1997.

[141] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, *Oblivious RAM simulation with efficient worst-case access overhead*, in *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pp. 95–100, 2011.

[142] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, *Oblivious RAM with $o((\log n)^3)$ worst-case cost*, in *ASIACRYPT 2011* (D. H. Lee and X. Wang, eds.), vol. 7073 of *LNCS*, (Seoul, South Korea), pp. 197–214, Springer, Heidelberg, Germany, Dec. 4–8,, 2011.

[143] C. Gentry, K. A. Goldman, S. Halevi, C. S. Jutla, M. Raykova, and D. Wichs, *Optimizing ORAM and using it efficiently for secure computation*, in *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*, pp. 1–18, 2013.

[144] X. Wang, T.-H. H. Chan, and E. Shi, *Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound*, in *ACM CCS 15* (I. Ray, N. Li, and C. Kruegel:, eds.), (Denver, CO, USA), pp. 850–861, ACM Press, Oct. 12–16,, 2015.

[145] C. W. Fletcher, M. van Dijk, and S. Devadas, *Towards an interpreter for efficient encrypted computation*, in *Proceedings of the 2012 ACM Workshop on Cloud computing security, CCSW 2012, Raleigh, NC, USA, October 19, 2012.*, pp. 83–94, 2012.

[146] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, *PHANTOM: practical oblivious computation in a secure processor*, in *ACM CCS 13* (A.-R. Sadeghi, V. D. Gligor, and M. Yung, eds.), (Berlin, Germany), pp. 311–324, ACM Press, Nov. 4–8,, 2013.

[147] L. Ren, X. Yu, C. W. Fletcher, M. van Dijk, and S. Devadas, *Design space exploration and optimization of path oblivious RAM in secure processors*, in *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, pp. 571–582, 2013.

[148] S. Wang, X. Ding, R. H. Deng, and F. Bao, *Private information retrieval using trusted hardware*, in *ESORICS 2006* (D. Gollmann, J. Meier, and A. Sabelfeld, eds.), vol. 4189 of *LNCS*, (Hamburg, Germany), pp. 49–64, Springer, Heidelberg, Germany, Sept. 18–20,, 2006.

[149] P. Williams, R. Sion, and B. Carbunar, *Building castles out of mud: practical access pattern privacy and correctness on untrusted storage*, in *ACM CCS 08* (P. Ning, P. F. Syverson, and S. Jha, eds.), (Alexandria, Virginia, USA), pp. 139–148, ACM Press, Oct. 27–31,, 2008.

[150] P. Williams and R. Sion, *Single round access privacy on outsourced storage*, in *ACM CCS 12* (T. Yu, G. Danezis, and V. D. Gligor, eds.), (Raleigh, NC, USA), pp. 293–304, ACM Press, Oct. 16–18,, 2012.

[151] E. Boyle, K.-M. Chung, and R. Pass, *Oblivious parallel RAM and applications*, in *TCC 2016-A, Part II* (E. Kushilevitz and T. Malkin, eds.), vol. 9563 of *LNCS*, (Tel Aviv, Israel), pp. 175–204, Springer, Heidelberg, Germany, Jan. 10–13,, 2016.

[152] B. Chen, H. Lin, and S. Tessaro, *Oblivious parallel RAM: Improved efficiency and generic constructions*, in *TCC 2016-A, Part II* (E. Kushilevitz and T. Malkin, eds.), vol. 9563 of *LNCS*, (Tel Aviv, Israel), pp. 205–234, Springer, Heidelberg, Germany, Jan. 10–13,, 2016.

[153] M. Maffei, G. Malavolta, M. Reinert, and D. Schröder, *Privacy and access control for outsourced personal records*, in *2015 IEEE Symposium on Security and Privacy*, (San Jose, California, USA), pp. 341–358, IEEE Computer Society Press, May 17–21,, 2015.

[154] "Amazon Web Services." `https://aws.amazon.com/`.

[155] National Institute of Standards and Technology, "Advanced encryption standard (AES)." Federal Information Processing Standards Publications - 197, November, 2001.

[156] P. J. Courtois, F. Heymans, and D. L. Parnas, *Concurrent control with readers and writers*, *Commun. ACM* **14** (Oct., 1971) 667–668.

[157] "iPerf - the TCP, UDP and SCTP network bandwidth measurement tool." `https://iperf.fr/`.

[158] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, *Dynamo: amazon's highly available key-value store*, *ACM SIGOPS operating systems review* **41** (2007), no. 6 205–220.

[159] C. Sahin, V. Zakhary, A. E. Abbadi, H. Lin, and S. Tessaro, *TaoStore: Overcoming Asynchronicity in Oblivious Data Storage*, in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 198–217, 2016.

[160] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **12** (1990), no. 3 463–492.

[161] M. K. Aguilera and D. B. Terry, *The many faces of consistency.*, *IEEE Data Eng. Bull.* **39** (2016), no. 1 3–13.

[162] D. K. Gifford, *Weighted voting for replicated data*, in *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, (New York, NY, USA), pp. 150–162, ACM, 1979.

[163] S. Y. Cheung, M. H. Ammar, and M. Ahamad, *The grid protocol: A high performance scheme for maintaining replicated data*, *IEEE Transactions on Knowledge and Data Engineering* **4** (1992), no. 6 582–592.

[164] D. Agrawal and A. El Abbadi, *An efficient and fault-tolerant solution for distributed mutual exclusion*, *ACM Trans. Comput. Syst.* **9** (Feb., 1991) 1–20.

[165] L. Lamport, R. Shostak, and M. Pease, *The Byzantine generals problem*, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **4** (1982), no. 3 382–401.

[166] J.-P. Martin, L. Alvisi, and M. Dahlin, *Minimal byzantine storage*, in *Proceedings of the 16th International Conference on Distributed Computing*, DISC '02, (London, UK, UK), pp. 311–325, Springer-Verlag, 2002.

[167] R. H. Thomas, *A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases*, *ACM Trans. Database Syst.* **4** (June, 1979) 180–209.

[168] L. Lamport, *Time, clocks, and the ordering of events in a distributed system*, *Commun. ACM* **21** (July, 1978) 558–565.

[169] M. Luby and C. Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, *SIAM Journal on Computing* **17** (1988), no. 2 373–386, [http://dx.doi.org/10.1137/0217022].

[170] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, *A concrete security treatment of symmetric encryption*, in *38th FOCS*, (Miami Beach, Florida), pp. 394–403, IEEE Computer Society Press, Oct. 19–22,, 1997.