Protecting Sensitive Data using Differential Privacy and Role-based Access Control

Mémoire

Hajaralsadat Torabian

Maîtrise en informatique
Maître ès sciences (M.Sc.)

Québec, Canada

Protecting Sensitive Data using Differential Privacy and Role-based Access Control

Mémoire

Hajaralsadat Torabian

Sous la direction de :

Josée Desharnais
Nadia Tawbi

Résumé

Dans le monde d'aujourd'hui où la plupart des aspects de la vie moderne sont traités par des systèmes informatiques, la vie privée est de plus en plus une grande préoccupation. En outre, les données ont été générées massivement et traitées en particulier dans les deux dernières années, ce qui motive les personnes et les organisations à externaliser leurs données massives à des environnements *infonuagiques* offerts par des fournisseurs de services. Ces environnements peuvent accomplir les tâches pour le stockage et l'analyse de données massives, car ils reposent principalement sur *Hadoop MapReduce* qui est conçu pour traiter efficacement des données massives en parallèle. Bien que l'externalisation de données massives dans le nuage facilite le traitement de données et réduit le coût de la maintenance et du stockage de données locales, elle soulève de nouveaux problèmes concernant la protection de la vie privée. Donc, comment on peut effectuer des calculs sur de données massives et sensibles tout en préservant la vie privée. Par conséquent, la construction de systèmes sécurisés pour la manipulation et le traitement de telles données privées et massives est cruciale. Nous avons besoin de mécanismes pour protéger les données privées, même lorsque le calcul en cours d'exécution est non sécurisé. Il y a eu plusieurs recherches ont porté sur la recherche de solutions aux problèmes de confidentialité et de sécurité lors de l'analyse de données dans les environnements infonuagique. Dans cette thèse, nous étudions quelques travaux existants pour protéger la vie privée de tout individu dans un ensemble de données, en particulier la notion de vie privée connue comme *confidentialité différentielle*. Confidentialité différentielle a été proposée afin de mieux protéger la vie privée du forage des données sensibles, assurant que le résultat global publié ne révèle rien sur la présence ou l'absence d'un individu donné. Enfin, nous proposons une idée de combiner confidentialité différentielle avec une autre méthode de préservation de la vie privée disponible.

Abstract

In nowadays world where most aspects of modern life are handled and managed by computer systems, *privacy* has increasingly become a big concern. In addition, data has been massively generated and processed especially over the last two years. The rate at which data is generated on one hand, and the need to efficiently store and analyze it on the other hand, lead people and organizations to outsource their massive amounts of data (namely *Big Data*) to *cloud* environments supported by cloud service providers (CSPs). Such environments can perfectly undertake the tasks for storing and analyzing big data since they mainly rely on *Hadoop MapReduce* framework, which is designed to efficiently handle big data in parallel. Although outsourcing big data into the cloud facilitates data processing and reduces the maintenance cost of local data storage, it raises new problem concerning privacy protection. The question is how one can perform computations on *sensitive* and big data while still preserving privacy. Therefore, building secure systems for handling and processing such private massive data is crucial. We need mechanisms to protect private data even when the running computation is untrusted. There have been several researches and work focused on finding solutions to the privacy and security issues for data analytics on cloud environments. In this dissertation, we study some existing work to protect the privacy of any individual in a data set, specifically a notion of privacy known as *differential privacy*. Differential privacy has been proposed to better protect the privacy of data mining over sensitive data, ensuring that the released aggregate result gives almost nothing about whether or not any given individual has been contributed to the data set. Finally, we propose an idea of combining differential privacy with another available privacy preserving method.

Table of Contents

List of Tables

List of Figures

Acknowledgment

First of all, I would like to express my sincere gratitude to my supervisor Prof. Josée Desharnais and my co-supervisor Prof. Nadia Tawbi for their continuous encouragement and support throughout my master period, and also for guiding me and giving me many precious advices during this project. I have learned a lot from them.

Besides, I want to thank Prof. François Laviolette for his kindness and helpful advices during my earliest days in Canada.

I am thankful to my best friend Shima who is the most honest and trustful person I have ever known. The birth of her sweet angel Hana brings to me a lot of energy and joy to write this thesis. Thank you for your continued friendship, support, love, hugs, laughs and many great things you gave me.

Also many thanks to Miad for all his love and support, and all my beloved friends here, particularly Solmaz, Asra, Atefeh, Mona, Amir, Rezvan, and Parnian, for sharing their moments and personal experiences of living in Canada with me.

Most important thanks to my mother, my sister Fatima, and my brother Hassan, who have been a constant source of love and consolation. Despite of being thousand miles away from me, they have truly supported me in all aspects of my life.

Huge love and appreciation go to Mona, the most supportive sister ever, who has always believed in me. In spite of passing many happy and sad moments together in our challenging life in Canada, she has always been there for me throughout my education and beyond.

Finally, a personal thank to anyone reading this thesis.

To my mother and in memory of my father, whose hopes exceed these pages.

Chapter 1

Introduction

1.1 Motivation

Currently, data is generated beyond the available users storage. Therefore, numerous users and IT organizations are more prone to outsource their huge amount of data (namely *big data*) to a remote and secure party. Hence, there is an increasing need to provide robust infrastructure and interfaces that can store, retrieve, and process big data in an efficient, scalable, flexible, and cost-effective fashion. This is feasible through *cloud computing*, which relieves data owners of the cost and burden of local data storage and its continuous maintenance. Data will be stored in the cloud so that cloud users can easily find their data on the go. Cloud Service Providers (CSPs) like Amazon, Google, and Microsoft provide components of cloud computing. They support massive processing power and storage, by which their users are able to deploy applications without infrastructure investment.

Cloud computing mainly relies on particular systems such as *Hadoop* [1], an open source software to process extensively vast amounts of data. It is actually based on Google File System (GFS) and the *MapReduce* algorithm introduced by Google. Hadoop first scatter massive datasets and store each piece of data across tens, hundreds, or even thousands of servers and process each part in parallel. The answers of each part are then combined in order to output the final answer.

Using Hadoop, big data outsourced to the cloud environments can benefit from parallel data processing and flexible data management services. Hadoop actually provides a reliable

shared storage and a capable analysis framework. The storage is provided by Hadoop Distributed File System while analysis is provided by MapReduce. This software is widely adopted in various investigations, such as clustering and classification algorithms, recommender systems, machine learning, and so on.

While cloud computing has emerged as an economical and efficient solution to store, manage, and process customers' data, it raises new concerns related to the security and privacy. Cloud users always worry about losing privacy of their stored data in the cloud databases scattered around the Internet. Hence, there are security and privacy issues in cloud computing that CSPs always face at every phase of design.

On one hand, a broad spectrum of organizations needs to analyze such vast amount of stored data in the cloud for many valuable uses. On the other hand, any unauthorized or improper access to such informative data (mostly sensitive) can compromise its privacy and jeopardize future access to it. Moreover, some cloud customers may allow CSPs to publicly release their *statistical databases* stored in the cloud for research purposes.

A statistical database [2] is a regular database that is mainly used to return *only* statistical information to the queries (e.g., the average salary of employees). An *adversarial* user, for example, may attempt to submit malicious queries to such released databases in order to intentionally reveal confidential information about their individuals. Therefore, securing such databases against these kinds of adversarial uses is crucial.

Cloud users can trust cloud computing and enjoy all its advantages, without worrying about the privacy of their sensitive information, only when CSPs can properly address users' privacy and security issues. A key challenge of preserving security and privacy in cloud environment is how to design a secure and practical system, through which developers and cloud clients spend as little mental labor and system resources on security and privacy as possible [3]. In this regard, a wide variety of techniques have been proposed either offering new ways or improving the previous ones. Our work gives an overview of these privacy preserving methods used in cloud computing services, ranging from access control mechanisms, to query auditing, to data perturbation techniques.

Among input perturbation, anonymization-based algorithms have been widely adopted, including ***k-anonymity***, ***ℓ-diversity***, ***t-closeness***, and ***m-invariance***. However, as we will see in this thesis, anonymization methods suffer from many flaws and thus fail to provide meaningful privacy guarantees. For this reason, differential privacy proposed by Dwork [4], has been adopted as a new and different way to defeat almost all the flaws in the previous work, aiming to provide robust privacy guarantees.

1.2 Objective of Research

In summary, we have the following main objectives in mind:

1. Reviewing several key concepts related to privacy protection over big data in the cloud systems as well as a discussion of potential security and privacy challenges in this way;
2. Studying multiple existing mechanisms to handle and process such vast amounts of data in an efficient, practical, and private way;
3. Suggesting a combination of differential privacy and role-based access control inspired by Airavat [3], in which differential privacy has been used in combination with mandatory access control.

1.3 Thesis Structure

This thesis is structured as follows:

We start with defining privacy in Chapter 2. Then we review some well-known privacy breaches that have been occurred so far. Afterwards, we study defensive mechanisms including *access control models*. However, access control models are either too restrictive or not strong enough to completely guarantee privacy of sensitive data. They actually lack flexibility. *Differential privacy* is thus described as a stronger paradigm for preserving large sensitive data sets stored in the virtual environments like the cloud, followed by some related works on differential privacy.

Chapter 3 provides a detailed definition of cloud computing as well as its main characteristics, service models, deployment models, and its famous service providers. It also presents pros and cons of using cloud environments, where the main challenges are security and privacy concerns. We then study several methods that have been already proposed by researches in order to deal with such concerns. Finally, we give an introduction to big data and explain how Apache Hadoop and MapReduce handle such massive amounts of data.

In Chapter 4, we study major privacy-preserving mechanisms over big data in cloud systems. We specifically give a detailed overview of differential privacy and its formal definition because we believe that among all presented mechanisms, differential privacy is the "right" notion of privacy for cloud computing environments.

Chapter 5 provides the current and the most important implementations of differential privacy. This chapter indicates that these implementations, however, are vulnerable against side-channels, including timing channels, state channels, and privacy budget channels. Finally, two proposed frameworks are described that are able to completely close these channels.

In Chapter 6, motivated by mechanisms used in role-based access control and one of the differentially private implementations (namely *Airavat*), we think about a combination of differential privacy with role-based access control for MapReduce computations. However, we do not evaluate it through running our algorithm on any real dataset to examine to what extent it is practical.

Finally, we make conclusion and discuss possible avenues for further work in Chapter 7.

Chapter 2

Background and Related Work

2.1 Privacy

Privacy is a right of any person or individual or group to keep their information private and is their ability to decide when, how, and to what extent reveal it. For example within an organization, all laws, mechanisms, standards and processes that manage *personally identifiable information (PII)*, must comply with the privacy policy of the given organization [5]. Actually, privacy is very context-dependent so that its meaning differs from one person to another and even for the same person based on the situation [6].

Privacy versus utility

Highly valuable information can be extracted from various statistical databases. For instance, customers' purchasing behavior can be analyzed in order to create targeted ads. As another example, consider a researcher who has intention to mine medical data in order to investigate new drugs. In these cases, there is always a trade-off between privacy and data utility. Several approaches exist to preserve privacy in different ways, but most of them sacrifice data utility. There are several definitions of data utility based on what kind of problems we want to address. For some cases, a querier can acquire reasonable accuracy for his result while database privacy is compromised. Suppose a hospital wishes to share its

medical database containing its patients' information for research purposes. If a querier is allowed to submit whatever query he[1] wants to such database and obtain accurate answer to his query, then it definitely comes at a price of losing privacy of the patients. Conversely, a querier may gain no meaningful utility as a way to preserve privacy. For example, the hospital in the above example, can enforce a limitation on the number of queries that will be executed on its database, and return perturbed and noisy answers to the querier. Clearly, no one can expect to achieve simultaneously full privacy together with meaningful utility guarantees for a dataset, that is, increasing privacy normally leads to decreasing utility and vice versa.

Hence, we always need to consider and control the trade-off between utility and privacy in order to preserve privacy while achieving optimal data utility. Furthermore, it must be taken into account that publishing sensitive information, such as census, voter registration, medical records, and salary information, can significantly compromise privacy of its individuals. In this regard, there is a wide variety of privacy violations over the past few years. A privacy violation involves unauthorized and improper access, use, or leak of personal and sensitive information, either intentionally or unintentionally.

Some privacy fiascos:

One of the first published attacks on privacy dates back to 2002 by Sweeney [7], and is known as the "Linking Attack". After removing all "identifiable information" (i.e., names, social security numbers, phone numbers, etc.), *Group Insurance Company (GIC)*[2] released a copy of the medical records of approximately 135,000 state employees and their families. The dataset consists of ZIP codes, birth dates, genders, and diagnoses (leftmost circle in Figure 2.1) and was supposed to be safe. Afterwards, Sweeney purchased a copy of the voter registration list for Cambridge Massachusetts including the name, address, ZIP code, birth

[1] Throughout this thesis, masculine pronouns are used to refer to a third person.
[2] Group Insurance Company (GIC) is responsible for purchasing health insurance for all Massachusetts state employees [7].

date, and gender of each voter (rightmost circle in Figure 2.1). She then linked these two datasets using ZIP codes, birth dates and genders that were the same in both lists. She then was able to re-identify the medical records of some individuals, particularly William Weld, the governor of Massachusetts at that time.

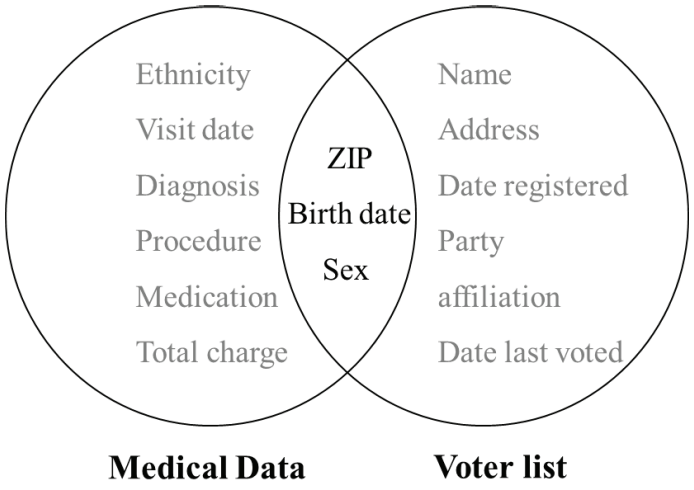

Figure 2.1: Linking to re-identify data [7]

AOL (known as America Online) is an American media technology company that provides an internet portal to let clients around the world access the best collection of journalists, artists and musicians on the web site for more than 30 years[3]. In 2006, AOL released anonymized search query logs publicly, including 20 million of web queries gathered over three months from 650,000 users, as well as clicked URLs among the search results and their ranking. In order to preserve privacy, AOL replaced all user IDs with random numbers and then released this dataset to help analysts provide better services for users and advertisers. After three days, without any outside information, two New York Times researchers re-identified user number 4417749 to be Thelma Arnold [8]. She was a 62-year-old widow who had frequently searches on topics like "*numb fingers*", "*60 single men*", and "*dog that urinates on everything*." In fact, this information disclosure was due to the information captured from the search histories (such as last names, the name of the town,

[3] http://corp.aol.com/

and so on) those were found in the search logs. In this case, there were only 14 citizens with the last name Arnold and the attackers contacted all 14 persons with phonebook listings and found Thelma Arnold.

AOL admitted it as a huge mistake, apologized for its release and eliminated the search data from its website; however, the removal was too late and the data was distributed widely and is still available on mirror sites [9].

Another example of linkage attacks was due to the public release of Netflix Prize dataset. The Netflix Prize was an open competition for improving Netflix's algorithm by 10% to predict users' ratings for movies based on their previous ratings, with a 1 million dollar prize[4]. To do so, Netflix anonymized movie ratings of 500,000 Netflix users by replacing user names with random numeric identifiers. Narayanan et al. [10] carried out a linking attack by cross-referencing this dataset with the public IMDb[5] dataset of information related to the movies, television programs, and video games, containing users' PII. That is, they re-identified a user only by a handful of his rated movies in both Netflix and IMDb without requiring that all movies rated by that user in IMDb be also rated in Netflix, or vice versa.

In another attack, Homer et al. [11] showed that, given a sample of an individual's DNA, they could accurately detect whether this individual has participated in a Genome-wide association study (GWAS). They developed a theoretical framework for determining the presence of specific individuals within a GWAS. The significance of being a member of a GWAS is that the members in the study have been diagnosed with a disease, that is, they have the disease, not just a gene for the disease. Therefore, revealing such sensitive and private information can completely compromise the privacy of its participants. For preserving privacy of the participants within a GWAS, their identity have to be masked.

[4] http://en.wikipedia.org/wiki/Netflix_Prize
[5] IMDb. The Internet Movie Database. http://www.imdb.com/

These privacy breaches occurred because data providers pay little attention to these kinds of attacks, which can exploit outside database or the semantic content of the released data set itself. To defend against these types of attacks, different models for privacy-preserving were proposed, including *Access Control mechanisms*. Also, there exist other approaches such as auditing untrusted codes and scrubbing or adding random noise to the attributes of database before releasing it as well as denaturing the output of the queries through adding random noise to the results. In the following section, we talk about access control and we will discuss other privacy-preserving approaches in chapter four.

What is Access Control?

Access Control is a collection of mechanisms that control how a resource can be accessed, consumed, or used by an individual within a system. Based on its policies, it enforces restrictions in order to prevent a system from unauthorized access (*confidentiality*) and improper or illegitimate modification (*integrity*), while making the resource available to the authorized users when needed (*availability*). Actually, it is all about keeping an intruder away from getting unauthorized access to the most critical resources and also limiting the operations that authorized users can carry out. Access Control policies can be enforced through four classes [12]:

1. Mandatory Access Control (MAC)
2. Discretionary Access Control (DAC)
3. Role-based Access Control (RBAC)
4. Rule-based Access Control

Throughout this thesis, RBAC is always used as the abbreviation for role-based access control. In the rest of this chapter, we elaborate on these access control models.

2.2 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is the most restrictive and probably most secure model of access control with the goal of preventing illegal information flow [13]. It was primarily defined and used by government and military environments, where information

classification and confidentiality are very important. the operating system controls all access to objects by assigning security labels to both objects and subjects and then applying these labels to constrain the interaction between them [14].

This model is mainly based on assigning security labels to all subjects (i.e., users, processes, or programs) and objects (system resources like files, folders, disk, memory, etc.). Subjects have a hierarchical security level, known as *clearance* (secret, top secret, confidential, and so on), which reflects the user's trustworthiness. Objects have a security label consisting of a classification (in the same way as clearance) and different categories (indicating compartments of data within a system that do not follow a hierarchical structure). The classification is defined based on the environment, in which it is applied. For example in a military environment, the classification can be defined as top secret, secret, confidential, and unclassified, while in another environment it may be quite different. The categories enforce need-to-know rules [14].

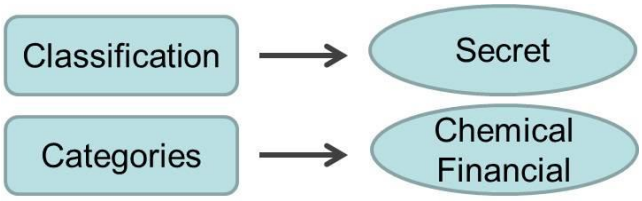

Figure 2.2: An example of an object's security label that consists of a classification and two categories.

Under MAC enforcement [14], when a subject requests to access an object, the operating system first check the subject's clearance with the security label of the object (both its classification and categories). Then, the access will be granted only if:

- The clearance level of the subject is equivalent or higher than the classification level of the object.
- And the subject has a sufficient access to all the object's categories.

As illustrated in Figure 2.2, MAC allows access to the object only to the users who have the secret (or higher) clearance and are authorized to access both chemical and financial categories [14].

2.3 Discretionary Access Control (DAC)

Discretionary Access Control (*DAC*) in another type of access control, in which access decisions are made at the discretion of data owners using Access Control Lists (ACLs) that are set by the owners in order to dynamically decide who can access what resources and to what extent [14].

Compared to DAC, in MAC systems, users cannot override security policies defined by the system administrator and users cannot decide who can access their resources [14]. On the one hand, giving access based on the choice of the owner makes DAC more flexible than strict policies in MAC, like those in military [13]. It also makes DAC more user-friendly and less expensive than MAC [14]. On the other hand, no control on released data makes DAC vulnerable to information leakage without the user's awareness. This vulnerability is popular for attackers who intend to exploit the system (e.g., Trojan horses) [12]. In this case, MAC provides more security since users are not allowed to install software (including malware), add new users, modify object permissions, and so on; however, these restrictions limit user capability [14].

In most of the operating systems such as Windows, Linux, Macintosh systems, and Unix, permissions are based on DAC through enforcing ACLs. One system that uses MAC is *Security-Enhanced Linux (SELinux)*, a modification to traditional Linux (discretionary) access control to support MAC, developed by the United States National Security Agency (NSA) [14].

2.4 Role-based Access Control (RBAC)

Neither discretionary access control nor mandatory access control are exactly what the most commercial enterprises need [13]. One alternative is *Role-base access control (RBAC)* that has a flexible mechanism that makes it powerful to simulate and enforce several variations of MAC [15]–[17] or DAC [18].

The concept of role-based access control pioneered in the 1970s [18] and started attracting a lot of attention in the 1990s [14], mainly for commercial applications [12].

Permissions under RBAC are based on roles among a system [15], and a user is usually defined as a human being who has some authority and responsibility within that system. Roles are assigned to a job function or job title according to the Organization Chart and can be applied to both users and groups. Users are then mapped to the roles based on their responsibilities rather than directly to permissions [19].

This model of access control is the best choice for a corporation that has a high staff turnover. That is, rather than frequently modifying the ACLs on the user's objects (which is the case in DAC), the system administrator only defines a role, maps permissions to this role according to its security policies, and then assigns the new users to this particular role. For example, if a user is assigned to the cashier role, any person who replaces this role after he quits his job, can be simply assigned to this role [14]. However, it is not possible that users go beyond permissions assigned for their role. For example, the cashier mentioned above gains exactly the same permissions as all other cashiers, nothing more and nothing less[6].

There are three levels of RBAC as follows:

Core RBAC: This is the fundamental component of every RBAC implementation. It includes users, roles, permissions, operations, and sessions, in which each session connects one user to a subset of roles according to the security policy. Different groups can be created with various access rights and many users can belong to multiple groups. If a user is a member of different roles, when he logs on to the system (establishes a session), he then automatically gains all of the access rights associated with these various groups at the same time. RBAC not only uses the user ID and credential to make access decisions, but also other information such as role's location, time of day, weekday, and so on [14].

---

[6] Security+ Essentials, an eBook by Neil Smyth.

Hierarchical RBAC: Role hierarchies let system administrators establish an inheritance (hierarchical) relation among the roles, whereby roles can inherit from other roles. This model is good enough for organizations, which have personnel hierarchical structures in their specific environment. In this case, a user who is mapped to a role inherits all access rights and permissions of other roles below his position within a pre-defined hierarchical relation. Then, he would gain more rights other than those already assigned to his role. However, a person in the senior role may not have the required skills for performing the tasks of a lower-grade role in the system.

This level has two types of role hierarchies [14]:

- *General Hierarchical RBAC:* It supports arbitrary and multiple role inheritances, that is, a user can inherit whatever role's permissions below.
- *Limited Hierarchical RBAC:* It only supports one level of hierarchy to impose limitations on the role hierarchy. for example, role *A* only inherits role *B*`s permissions if *B* is his immediate descendant, nothing more from other roles.

Constrained RBAC: It adds *Separation of duties (SoD)* to the hierarchal RBAC. SoD partitions authorities for a single task among more than one role in order to prevent fraud and error. In doing so, a single user is prohibited from having much permissions, and hence involvement of multiple users is required to commit a malfeasance. This model provides different separations of duties [20]:

- *Static SoD* disallows combination of roles assigned to a user (i.e., the user is not allowed to be a member of more than one role.). Static is easier to test.
- *Dynamic SoD* enforces constrains on the combination of privileges that can be activated in any session. That is to say, a user is able to be a member of more than one role but not at the same time. It means when a user logs in as a role, during this session, his other roles' permissions are not available to him. Dynamic is more flexible.

2.5 Rule-based access control

Rule-based access control indicates interaction between a subject and an object under a set of predefined rules that must be met before any access to the object. These rules can be either simple or complex. Examples of this model include situations such as accessing a network connection only at given hours of the day or weekdays together with other circumstances like user's security clearance. Some systems such as MAC, firewalls, and routers enforce rule-based mechanism for their complex access rules [14].

Contrary to DAC that creates access control lists based on identity of users, rule-based access control usually sets rules for all users without checking their identity (e.g., consider a policy that controls e-mail attachments to be maximum 4MB). This avoids complexity and confusion [14]. Compared to role-based access control, which focuses on the subjects (persons) and setting permissions to the roles, rule-based access control focuses on the objects and on assigning permissions to an object, specifying what operations can be done by which subject or group of subjects.

Intuitively, operating systems can be modeled to use one, two, or all three models (i.e., MAC, DAC, and RBAC) together; however, access control framework is not good enough for completely protecting privacy against all kinds of attacks. One possible way is extending role-based access control through enforcing more restrictions on the users' access to data and the operations they can perform on that data. To do so, we can define more restrictive access control for some sensitive attributes that is known as *Privacy-Aware Role-Based Access Control*. For instance, a user who has access to a private database can only access some less private attributes such as customers' home addresses but not their Social Security numbers [14].

Besides all these access control mechanisms, we need stronger mechanisms to provide more meaningful privacy protection like *Differential Privacy*, which we describe in the following section.

2.6 Differential Privacy overview

Recently, there has been a tremendous need to publish and mine personal data by analysts for a wide variety of reasons like research purposes, offering better services, recommendation systems like Amazon or Netflix, and advertising purposes. One of the main challenges we face is protecting data against any unauthorized access or disclosure as well as minimizing the risk of data leakage during analyzing data.

Sometimes, one's privacy can be harmed not only by explicitly querying about his information, but also by submitting aggregate queries like COUNT, SUM, AVG, , MIN, MAX. The participant hence loses privacy even without participating in the dataset. Suppose a database publishes the average heights of its individuals; in this case, disclosing one's exact height is a privacy breach. An adversary who has access to the actual average value and the auxiliary information "*X* is three inches higher than the average heights" can trivially learns *X*' height, no matter *X* participates in the dataset or not [4].

Therefore, we need a proper and robust definition of privacy that is totally independent from the presence or absence of any individual's information within a dataset and also regardless of any external and auxiliary information that an attacker may obtain from other resources. That is to say, anything an attacker can learn from a dataset when an individual's data is included is approximately the same without its data.

In this regard, Dwork et al. [4], [21]–[23] proposed *"Differential Privacy",* a notion of privacy that provides a strong guarantee on generating results of a computation completely independent of whether or not any individual's data is in the dataset while still obtaining good utility. To protect privacy through this approach, the output of a computation is perturbed and then the perturbed result is returned to the querier, who may intend to infer the participant's information.

The idea of differential privacy is adding adequate noise to mask output so that an attacker is not able to learn useful information about the input by looking at the released outputs of the aggregate queries. In fact, we say a computation is differentially private if the probability of generating an output is not dependent much on whether an input item is

included in or excluded from the dataset. Differential privacy assures the data owner that it protects the individual's private data against disclosure by hiding the presence or absence of any participant in the dataset. Therefore, no further privacy breaches occurs by joining to or removing from a dataset [6]. We formally define differential privacy with more details in chapter four.

2.7 Related work in Differential Privacy

There is a bunch of literature on differential privacy and this work does not intend to review all of them but directly relevant work.

Differential privacy was originally introduced by Dwork et al. [4] in 2006. To achieve it, they defined *Laplace mechanism* [24] that adds appropriate amount of random Laplacian noise to the output. Dwork, in her talk at Berkman [25] entitled *"I'm in the Database, But Nobody Knows"*, offered this notion of privacy to let an analyst access to a sensitive dataset and run queries on it without revealing any participant's private information. It also copes with all auxiliary information in the past, present, and future databases.

Laplace mechanism is only applicable for computations that produce numeric outputs, so how to privately answer non-numeric queries in a differentially private way?

To address this problem, McSherry and Talwar [26] developed *exponential mechanism*, which is designed to support differentially private computation over arbitrary domains and ranges. The exponential mechanism takes a dataset D, output range T, privacy parameter ϵ, and a utility function $u: (D \times T) \rightarrow \mathbb{R}$ as inputs. The utility function maps every output t belongs to the range T to a real valued score, where higher scores imply better utility. They demonstrated that, in some cases like game theory, differential privacy could be considered as a key solution to obtain guarantees and it is improbable to have unwanted results using exponential mechanism.

Hun et al. [27] used exponential mechanism of McSherry and Talwar for proposing an algorithm to satisfy differential privacy. They named it *DiffMR* and aimed to use it in top-k query processing. DiffMR algorithm applied exponential mechanism to select top-k records

from large data sets while preventing leakage in the middle process. Then, they added Laplace noise of Dwork to the final top-k result for protecting individual's privacy, and finally a post-processing technique was applied to gain better accuracy. Their experimental results showed a good efficiency that this algorithm had in Map-Reduce framework while satisfied differential privacy and utility.

Jagannathan et al. [28] proposed a learning model inspired by *Semi-Supervised Learning* [29] that uses both private (a small amount) and public data in order to improve the accuracy of results while guaranteeing differential privacy. In this approach, they first created a differentially private classifier using private data and then the accuracy improvement of the initial classifier was achieved by using public data as a post-processing phase. Compared to previous mechanisms, which considered whole dataset private, this model dramatically boosted utility without sinking privacy.

McSherry [30] presented *Privacy Integrated Queries (PINQ)*, a reliable platform for privacy-preserving computations on sensitive data using a SQL-like language, while obtaining differential privacy guarantees for the results.

Roy et al. [3] introduced *Airavat*, a framework that combines mandatory access control with differential privacy in order to provide end-to-end security and privacy guarantees for distributed computations on private data. Airavat confines untrusted code, run it on unmodified data, and return aggregate results. In this system, Data owners control and define the security policy for their sensitive information and Airavat prevents leaks beyond this pre-defined policy.

2.8 Summary

In this chapter, the concept of privacy was first defined and then some known privacy breaches were presented. Next, we covered access control definition as well as its different models, from which the organizations choose according to their business and security requirements. Then, we gave an overview of differential privacy as a strong paradigm for preserving private information. In the last section of this chapter, some work in differential

privacy were presented that are relevant to our research subject. The next chapter will talk about Cloud systems characteristics, big data, and introduce some key concepts to handle big data and manage cloud systems such as MapReduce framework and Hadoop.

Chapter 3

Cloud Computing, Big Data, and Hadoop

3.1 What is Cloud Computing

With the development of technology, the Internet has entered into people's everyday life. At the same time, *Cloud Computing* has increasingly received attention from both IT industry and ordinary users due to valuable advantages it offers. Cloud computing has evolved over the past few years and it seems everything is moving into the cloud. Now, internet users are using cloud services, rather than running their applications on a desktop computer or server. But, what exactly is the “cloud”?

Cloud computing has emerged as an integration of *grid computing*[7], distributed computing, and parallel computing [31], which enables massive computation power and storage capacity. It provides users with an environment to deploy their applications and computations without investing in new infrastructure.

[7] Grid computing is a form of distributed computing, which gathers and merges large numbers of connected computers from multiple areas with different computing powers to achieve a common goal or solve a particular task like weather forecasting and earthquake simulation [75].

Subashini and Kavitha [32] define cloud computing as "*Cloud computing is a way to increase the capacity or add capabilities dynamically without investing in new infrastructure, training new personnel, or licensing new software. It extends Information Technology's (IT) existing capabilities"*. The idea is to lower the computing burden in client side by transferring it to the shared infrastructure.

The NIST[8] defines cloud computing as follows [33]: *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

The name "Cloud" originated from the fact that users have no idea where their data is actually located (a virtual location) and what kind of processes are going to be done over it. In fact, data is scattered and processed in multiple locations across the network in the world. On one hand, a cloud user does not need to care or being aware of what technical details are applied in the cloud environment while he benefits from the powerful computations of the cloud. On the other hand, he loses full control over his sensitive data and does not know who exactly performs what computations on his data [14].

To get an understanding on cloud computing, let us consider an example: suppose a company needs to do complex computations on a large dataset like customers' shopping records or healthcare data. For doing these kinds of computations, this company would need to have many strong servers and a team of experts to constantly maintain and update all related software and hardware.

Even after providing all these requirements, data will be getting bigger and bigger over time, and thus even an excellent server would be limited. Also, maintaining such resources and servers costs a lot and the experts have to continuously enhance their ability and

[8] National Institute of Standards and Technology

knowledge. It is the main reason why many users have turned to the cloud environments in order to store their large datasets as well as analyzing them.

The term *"moving to cloud"* is used when an organization decides to leave behind the constraints of traditional way (i.e., having its own data center and providing expensive devices and storages) and outsources its sensitive information to a cloud environment[9]. Cloud computing is a better way to run user's business on shared data centers without any need to train new staff, upgrade and license new software, and buy new infrastructures. To use a cloud application, a user only needs to log in, customize his setting, and then start working.

Cloud computing aims at establishing a user friendly environment, which is more flexible and scalable than current services and applications. Cloud computing platforms let users download easily whatever services they need from the internet (usually for free) without installing basic hardware and software infrastructures. Users only need a high-speed internet and then they can mainly focus on their business [5].

Cloud computing consists of some key features, three service models, and four deployment models that we elaborate in the following sections.

3.1.1 Cloud characteristics

The essential features of cloud computing are as follows:

- **Pay-per-use:** Cloud users pay only for what services and resources they use (pay-as-you-go model), i.e., based on this pay-as-you-go strategy, if a user wants to rent a service or a server for one hour, then he only pay for one hour [34] [33] [35].
- **Reduce cost:** Cloud reduces costs and energy consumption because users don't need to buy and install expensive hardware and software systems as the cloud

[9] http://en.wikipedia.org/wiki/Cloud_computing

providers buy all of them. Users are also free from spending so much time and energy on hardware and software maintenance and manual update, they rather outsource them to the cloud, where upgrades are automatically done [31] [36].

- **On-demand capabilities and services:** Cloud enables its users to access cloud capabilities such as email, applications, servers, and services over the network and let them change their preferences and services whenever they want. Payment terms and conditions differ from each cloud provider where billing units can be monthly subscription, pay per usage, and so on [31] [33] [36].
- **Quickly scale up or scale down:** Using a cloud platform, users can quickly and easily upscale or downscale their resources in any quantity at any time. Suppose at first a user rents two servers and after 20 minutes he decides to rent 100 servers more, then a cloud system (like amazon) gives him this flexibility and do it for him in a few clicks [31] [33] [36].
- **Anytime, anywhere availability:** Since data is located in the cloud, ubiquitous access to data from smartphones, tablets, laptops, and so on is quite easy from anywhere and at anytime via internet with no need to carry data everywhere [31] [33] [36].
- **Resource pooling:** Using the cloud services, millions of users can be concurrently supported through sharing resources (e.g., storage, memory, network bandwidth, etc.) without any need to provide each user with a proprietary hardware. Since the computations are virtualized and not physically tied to a data center, several business offices and sales teams who are usually outside the office can be served from any location, at any time. Costs will be significantly reduced as well [33] [36].
- **Flexibility:** Through the cloud, consumers can be more flexible to unlimitedly share whatever documents and resources they want over the network [37] [34].
- **Security:** With the shift to the cloud, users can benefit from security team and expertise of the cloud providers; however, cloud itself raises several concerns about lack of security and control over sensitive data that we mention later [34] [38].
- **Data replication:** Since reliability is an important issue with distributed systems like cloud, data is replicated on multiple (normally three) servers across several data centers to achieve fault tolerance and reliability in real time computing and makes

cloud more accessible. By replicating users' data, if data is accidentally lost or denatured during the processing, then the cloud redirects to another server where it holds a copy of data and continues processing without any interruption. While it may seem difficult to hold multiple copies of a large dataset, the cloud systems properly do it across multiple nodes in order to make a cluster resistant to the inevitable failures of any single node [37] [35] [39].

- **Multi-Tenancy:** It allows multiple clients to access one single instance of a shared application so that every user is able to customize it for his specific need [31]–[33], [36].

Using cloud computing, users can easily use any resources and services they want even without any knowledge on the details of cloud implementation and the number of its hardware, CPU's, and so on. Users only need to properly understand what a service offers and how to work with it. Also, both the user side and the cloud provider's side are able to systematically back up and monitor data in order to provide transparency.

3.1.2 Cloud service models

As illustrated in Figure 3.1, there are three models with respect to the cloud services. They are presented in the following [33] [36] [40]:

1. *Infrastructure-as-a-service (IaaS)* is an evolution of traditional IT that provides the underlying infrastructure including networking, virtual machine, server, storage, and virtualization technology as a service. The cloud users then deploy and manage their own application, data, software, and probably operating system themselves just the way they do in their own datacenter. This model is responsible for managing and maintaining offered services, therefore users can benefit from significant cost saving on managing the underlying infrastructure components. Amazon Elastic Compute Cloud (EC2) and Secure Storage Service (S3) are examples of IaaS offerings.
2. *Platform-as-a-service (PaaS)* builds and run on top of the IaaS model and is a collection of middleware, web servers, databases, and development tools. In the

cloud diagrams, PaaS is mostly deployed between the SaaS layer above it and the IaaS layer below. Using PaaS model, cloud users don't need to manage and control the underlying hardware and software layers such as servers, network, storages, and operating systems as well as all the cost and complexity related to managing and maintaining them. Sometimes PaaS providers have their own programming languages (e.g., force.com from Salesforce.com) or only support a particular language (e.g., developers using Google App Engine can only write in Python or Java).

3. *Software-as-a-service (SaaS)* builds on top of the two previous services. This model provides, controls, and manages all the underlying infrastructure and platforms that run an application (e.g., email, CRM, virtual desktop, games, etc.) and then delivers it as a service to serve multiple users over the web. Using this model, the customers are free of the complexity of programming and developing the applications and they may only need to do some simple and flexible modifications to the given application configuration setting. Through SaaS, total cost of hardware and software operations is significantly reduced because users only pay as they use without buying anything else.

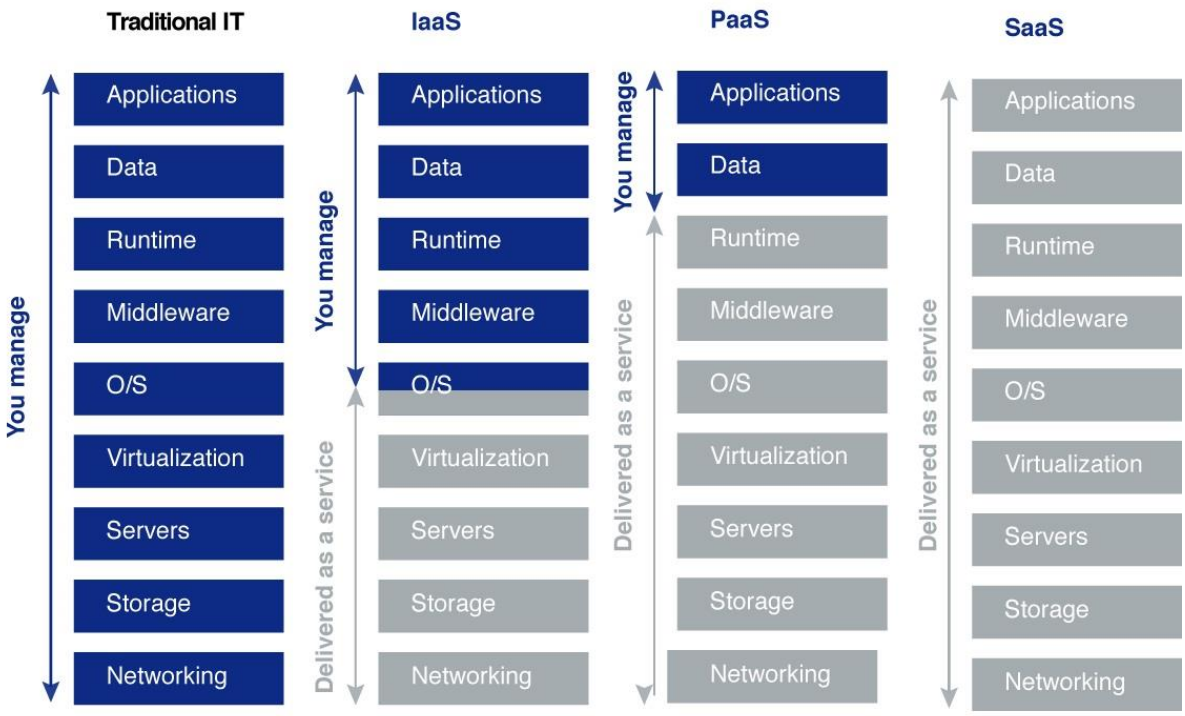

Figure 3.1: Taxonomy of Cloud Service Models[10]

3.1.3 Cloud Deployment models

Cloud can be classified as four deployment models (see Figure 3.2) [33] [36]:

1. *Private Cloud*: Similar to traditional IT, the cloud provides infrastructure and services for a single organization or a group of clients, typically using the client's software and hardware even within the client's data center and firewall. By this model, the organization itself, or along with the cloud party, can control and manage all the processing tasks.
2. *Community Cloud:* the cloud infrastructure is provisioned and shared between multiple companies comprising community of clients that generally have common purposes.

[10] The figure is taken from https://peterhanselman.wordpress.com/2011/05/22/cloudnomics/

3. *Public Cloud*: This model provides cloud services to the public via Internet (the service is physically delivered from outside the clients' data centers and firewall). This model is a good choice for commercial purposes when people or companies have common requirements. Compared to private cloud, this model is more economical but less trustful.
4. *Hybrid Cloud*: This model is an integration of two or more private, public, or community cloud that takes advantage of all the mentioned models. For instance, a cloud user can employ public cloud for non-private data while he can use private cloud in case of sensitive data.

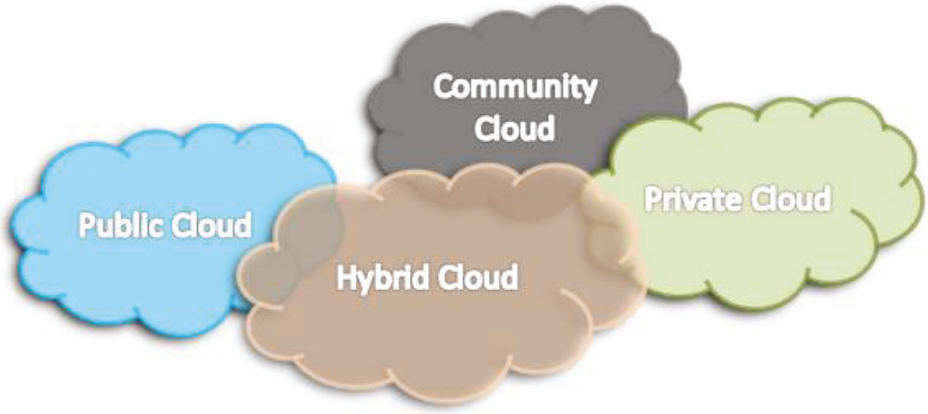

Figure 3.2: Deployment models of cloud computing

3.1.4 Cloud services providers

Well-known Cloud Service Providers (CSPs) are: Amazon, Salesforce, IBM, Google, Microsoft, Citrix, CSC, BlueLock, NephoScale, Verizon/Terremark, Joyent, RackSpace, CenturyLink/ Savvis, VMware, SAP, Netsuite, 3Tera

3.2 Advantages and challenges in cloud systems

When it comes to the complex and large-scale computing, cloud computing emerges as the best way for any enterprise, which does not have or even want to have huge resources and facilities as well as high cost of maintaining and managing them. Therefore, many

enterprises and people turn to cloud and take advantage of its unique characteristics in order to handle large-scale computations over their own big data [27].

Although cloud has many benefits for their customers such as saving users' money, ability to store more data compared to private servers, automatic updates, low cost, high scalability, availability, flexibility, and other advantages and services we mentioned above; there exist several challenges in using cloud systems, such as the need to constantly connect to a high speed internet while using cloud applications. However, security and privacy issues ─ especially when it comes to sensitive information that are processed in multiple data centers ─ are the most important concerns in this system that people always face and worry about. In fact, when a data owner decides to outsource his sensitive data to the cloud and let an untrusted code run over it for commercial, advertisement, or medical purposes, then how he can ensure that his sensitive information will not be in danger of leakage!

3.2.1 Security and privacy issues

Despite of the rise of cloud computing services and its widespread utility, preserving security and privacy is the main challenge in this area [5]. By outsourcing data to the cloud, users have no idea where their own data is exactly located and they no longer have physical access and full control over it. Therefore, security and privacy issues need to be properly addressed to establish a robust guarantee [31]. We now highlight key security and privacy issues.

Security issues in cloud systems must be addressed to avoid potential problems that might arise. All the security issues given below are gathered from: [5], [31], [36], [41], [38].

According to cloud providers, security issues are:

- How to provide confidentiality, integrity, and availability for any cloud user's sensitive data? And how to guarantee availability and data retrieval in the cloud while users have no access to the physical data and they only possess the virtualized data?

- According to the common issues in IT such as fraud, exploitation, and phishing, how to protect confidential data against unauthorized and illegal access, hijacking, and attacks by adversarial cloud users?
- How to avoid interpretability problems caused by using different identity tokens and protocols by distinct users according to the multi-tenancy feature in cloud computing?
- Suppose a user decides to change his cloud provider (e.g., move from Google cloud to Microsoft cloud). How to provide compatibility for security requirements between multiple cloud vendors services where different cloud providers have different policies?
- On one hand, some countries have strict regulation concerning its citizen's data storage. For example, bank policies normally disallow customer's financial data to be stored outside the country. On the other hand, cloud data is stored or probably transferred to another data center in another country. Therefore, how to support such severe regulations in the cloud?

According to cloud users, security issues are:

- Due to the general lack of transparency and long term data storage in the cloud systems, how is a cloud server entirely monitored and traced, aiming to prevent wicked insider from access and misuse users' private data?
- Due to the huge amount of data stored in the cloud, how can users audit all the activities and processes done by CSPs on their data in order to have more secure and trusty cloud?
- How to eliminate or lower the risk of data seizure by foreign government arising from violating the law by CSPs?
- How to ensure that firm and proper laws are established in the cloud in order to protect cloud users against their cloud providers in case of any malicious and illegal data utilization by them?
- Who exactly provides and manages the encryption/decryption keys, which should be logically done by the cloud users?

In the context of cloud computing, privacy issues vary based on multiple cloud providers' policies, and these issues are as follows [5] [31] [36] [38]:

- How to assure users that they still have full control over their information during its storage and processing in the cloud where data is virtualized as well as guaranteeing their data against theft, malicious and unauthorized access by establishing strong privacy protection mechanisms?
- How to guarantee that users' data replication is properly done at multiple safe locations and keep it secure from any leakage, loss, forgery and unauthorized modification as well as deleting all these copies when they are no longer required?
- To what extent is it possible to correctly verify, check, and identify a subcontractor (third party) who runs critical functions on users data?
- For more clarity and protection, users must be announced and kept up to date for any change and improvement to the cloud services and application, and also they need to know how it will be processed by other parties.
- One significant privacy concern is how to protect individuals' data against unauthorized secondary use such as junk advertisement?

There are some other issues related to the *trust* as well. Trust is a personal and measurable belief in correctness of something that constructs relationship among people and entities based on multi-dimensional factors such as experiences, perception, feeling, communication, shared values, and so on. In fact, trust always depends on mutuality and is a non-symmetric, context-based, and partially transitive concept. In this regard, trust issues can be: (1) according to the unique architecture of the cloud computing, how to define and assess trust, (2) How to create distinct security levels for cloud services based on trust degree, (3) How to monitor, regulate and manage changes to the trust degree within time and context as well as providing dynamical trust relationship [5].

To address the security and privacy issues listed above, several researchers have proposed multiple methods. Wang [31] proposed four efficient mechanisms; however, they can only handle one or two issues in each domain (i.e., privacy and security domains). Two methods to deal with policy integration and access control problems and two methods as a solution

to the lack of user control over their cloud data as well as unauthorized secondary usage. In this regard, Tiwari and Mishra [41] presented steps to better understand, check, and clarify cloud security issues before adopting cloud computing as well as some solutions for security issues. Popovic and Hocenski [38] investigated security problems and requirements that CSPs encounter during designing and implementing cloud. They presented standards for better managing security and elaborated twenty security management models and their requirements to help CSPs improve their cloud systems. Yet, we need more effective methods and mechanisms to handle further issues in order to have more private and secure cloud.

Like every new technology, cloud computing has a few defects related to privacy and security issues that we mentioned above, however, it will get improved through the time. Also, CSPs have to follow a set of rigorous laws to better control how data is processed in the cloud and to properly manage the security and privacy of these environments. Indeed, proper laws must be established to keep and manage vast amount of sensitive data safe against any fraud and disclosure [36]. It is obvious that such massive amounts of data is growing at an exponential rate and needs to be safely and properly stored and processed. Hence, we need to understand the concept of big data and possible ways to handle it, which we explain in the following sections.

3.3 About Big Data

3.3.1 What is Big Data?

People all around the world are constantly generating and releasing data about themselves on the internet. Around 90 percent of all worldwide data has been generated and utilized over the last two years. All this data is coming from multiple sources like smart phones, social networks, digital photos and videos, medical records, twitter messages, purchase transaction records, surfing the net, and so on. A large volume of data is produced from mobile devices alone every second. This is what is called *big data*, which normally resides in cloud environments.

Big data is a term for any huge and complex collection of datasets that on-hand data management tools (such as the RDBMS[11]) and traditional data processing applications have difficulty managing and processing within a tolerable elapsed time[12]. No matter how big and powerful a computer is, it would definitely have limitations when it comes to big data. Note that what is deemed "big data" is relative, i.e., its definition can vary from one person to another according to his power and capabilities for storing and processing his own data. That is, "*For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration* [42]." Big data, however, does not only imply the volume of data; it is also about how complex its processing will be. Without data analytics, big data is only a bunch of data.

Big data is often described using five Vs as follows:

1. **Volume:** The name 'big data' itself implies a phrase related to a large size, and thus volume is an obvious attribute of big data. In general, many factors result in an excessive increase in data volume, such as text data constantly produced by social media, customers' transactions, sensor data, and so on. The size of the data indicates whether it can be considered as big data or not [34] [43].
2. **Variety:** It refers to the existence of different types of data from multiple sources, used in all today's data analysis and decision-making process. In the past, organizations mostly dealt with structured data that properly fits into data tables or relational databases. They could be simply processed by traditional tools like the RDBMS. In contrast, today's data is of all types (structured, semi-structured and unstructured), including all messages and comments on social networks, audio, video, click streams, photos, sensor data, log files etc. Emerging big data technology allows us to manage these various types of data [34] [43].

[11] Relational Database Management System
[12] https://en.wikipedia.org/wiki/Big_data

3. **Velocity:** It indicates how quickly data is generated and how quickly it can be accessed and processed to accomplish a task. It is about the time it takes to analyze the data or even inform the sender that the data has been delivered from the moment the data hits the wire. Responding fast enough to deal with velocity is an important issue for most organizations (consider the rate at which credit card transactions are checked for fraudulent activities). Using big data technology, we are able to process the data while it is being generated, without the need to initially store it into databases [34] [43].
4. **Variability:** It is about the inconsistency that the data can show over time. It brings new challenges to the process of effectively managing and handling the data, particularly when it comes to unstructured data [44].
5. **Veracity:** This term is used to indicate the quality or trustworthiness of the captured data. The veracity of source data is an important feature in an accurate data analysis, which needs to trust the integrity of the data for making important decisions. A big data platform smooths the way for users to better manage the less reliable and somehow uncontrollable data. As an example of such data, consider Twitter posts with hash tags, typos, colloquial speech, and abbreviations [34] [43].

3.3.2 How to handle and analyze Big Data?

Traditional storage and analytic tools are capable of storing and analyzing only a small portion of all the data in today's world. In the traditional approach, an enterprise gets a very powerful server that works great until its data does not get larger. Even high-performance and perfect servers are either too slow or too limited to store and process the data as its size grows. The reason is that these servers are not scalable despite their power. Therefore, we need to get help of distributed computing, through which the data is stored and processed in different locations across the network, interacting with each other by passing messages. Distributed computing divides a problem into multiple tasks, each of which is then assigned to a worker that processes it and then outputs the result. As the last step, the results of all workers will be combined to produce the final output. Although it would be beneficial to

employ distributed computing for big data, there are some issues that need to be addressed, such as:

- How multiple tasks can be assigned to different workers in an efficient way?
- How to handle a task in case of any kinds of failures like disk, hardware, and network failures, where probability of failures will increase with respect to the data volume and number of tasks?
- How to make sure that workers properly exchange the results?
- How to synchronize distributed tasks assigned to different workers?

In this regard, one solution is *Hadoop*, a popular and well-known cluster-based framework that is designed to store and process a very large amount of data. The core idea behind Hadoop is processing the data in parallel rather than in serial. Its goal is to address almost all concerns a data owner may have regarding his data storage and management. Indeed, the time has come for enterprises to reconsider their traditional policies and tools, and turn to frameworks like Hadoop, by which their data can be properly processed and managed within tolerable elapsed times. But what is exactly Hadoop and how can it help in solving the big data problem?

3.4 Hadoop

3.4.1 Overview

Hadoop is an open source distributed computing software based on Google File System (GFS) and MapReduce from Google. Hadoop was developed by Cutting and Cafarella in 2005 [45] and donated to the Apache Software Foundation[13] in 2006. The Apache Hadoop has rapidly emerged as an effective way of managing large volumes of both structured and unstructured data.

[13] https://en.wikipedia.org/wiki/Apache_Software_Foundation

Hadoop supports running of complex applications on big data using Hadoop multiple components. Hadoop enables distributed computing of big data across clusters of low cost computers, known as commodity computers, in an efficient, scalable, reliable, and cost-effective fashion. Hadoop is efficient due to its ability to perform a task in parallel, which increases the processing speed. Hadoop is highly scalable due to its ability to scale up from a single computer to thousands of servers, each supporting local processing and storage. That is to say, a Hadoop user, who has started with few computers, can easily increase the number of computers if his requirements change. Hadoop is reliable, meaning that it provides data replication to guarantee data availability, avoiding any interruption during data processing even if a computer goes down. And finally, Hadoop is inexpensive because it works on commodity computers using simple programming models, which makes it available to almost any kind of user. In addition, Hadoop offers redundant and fault-tolerant data storage, free and reliable computation framework, and job coordination.

Hadoop is based on Linux, so all the commodity computers have Linux as their operating system. To process data in parallel, Hadoop splits up both the data and the computation into smaller pieces and sends each piece of computation to each piece of data independently. Once all the computations are done, all the results from previous step are combined and sent back to the client as a final result.

Hadoop has a simple architecture, consisting of a set of tools like HBase, Hive, Pig, ZooKeeper, Mahout, etc. Actually, it is not only *one* project that a user can download on his computer and think it is done! Hadoop mainly consists of two components:

1. **Hadoop Distributed File System (HDFS)** is a distributed file system inspired by GFS. Actually, HDFS does the locating and replicating of all the data on clusters of computers.
2. **MapReduce** is the heart of Hadoop and provides a simple and effective way of processing large datasets residing in HDFS.

The other Hadoop tools provide additional services, which are usually built and run on top of these two main components in order to add higher-level abstractions.

In the rest of this chapter we explain the core components of Hadoop, i.e., HDFS and MapReduce. Note that you can find precious and up-to-date information about Hadoop as well as excellent Hadoop tutorials on the Apache Hadoop website [1].

3.4.2 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System is designed to store massive amounts of data (terabytes or even petabytes), running on large clusters having thousands of nodes while providing on-demand access to this data. When we say node, we are not talking about one big powerful server; nodes are actually numerous inexpensive commodity hardware.

The input file is first divided into smaller chunks, which are stored in a sequence of blocks of the same size (one block per file). Breaking the data into multiple chunks may increase the chance of hardware failure. To avoid it, HDFS supports data replication, i.e., it creates redundant replicas of the same block, scattering along different nodes. This ensures redundancy, reliability, availability, and fault tolerance in case any node fails. The block size and the number of copies of each file, namely the replication factor, can be specified by the user's application (normally three copies of each file with 16 MB to 64 MB block size).

Normally, an HDFS cluster has a single master node (aka *NameNode*) and a number of worker nodes (aka *DataNodes*). The NameNode regulates access to the data by users, so that the data in not accessible without the authorization of NameNode. The DataNodes, running on multiple nodes in the cluster, take care of storing and retrieving blocks, following the instruction dictated by the NameNode or the client itself. The NameNode is responsible to keep track of what data is residing on which DataNode. It periodically receives a Heartbeat, indicating that all DataNodes are properly working; and a report, containing a list of all blocks on each DataNode.

The master node has a JobTracker component and each worker has a TaskTracker component. Actually, HDFS takes care of NameNode and DataNode while MapReduce takes care of JobTracker and TaskTracker. The JobTracker component, running on the master, is responsible for breaking a big task into smaller tasks, and assigning each small

task to the TaskTracker of idle workers. TaskTracker then processes the small piece of task given to the particular worker node. Once the task is performed, the result goes to the JobTracker, which combines the whole results together and give it back to the client.

Figure 3.3 illustrates these components within the master/worker architecture.

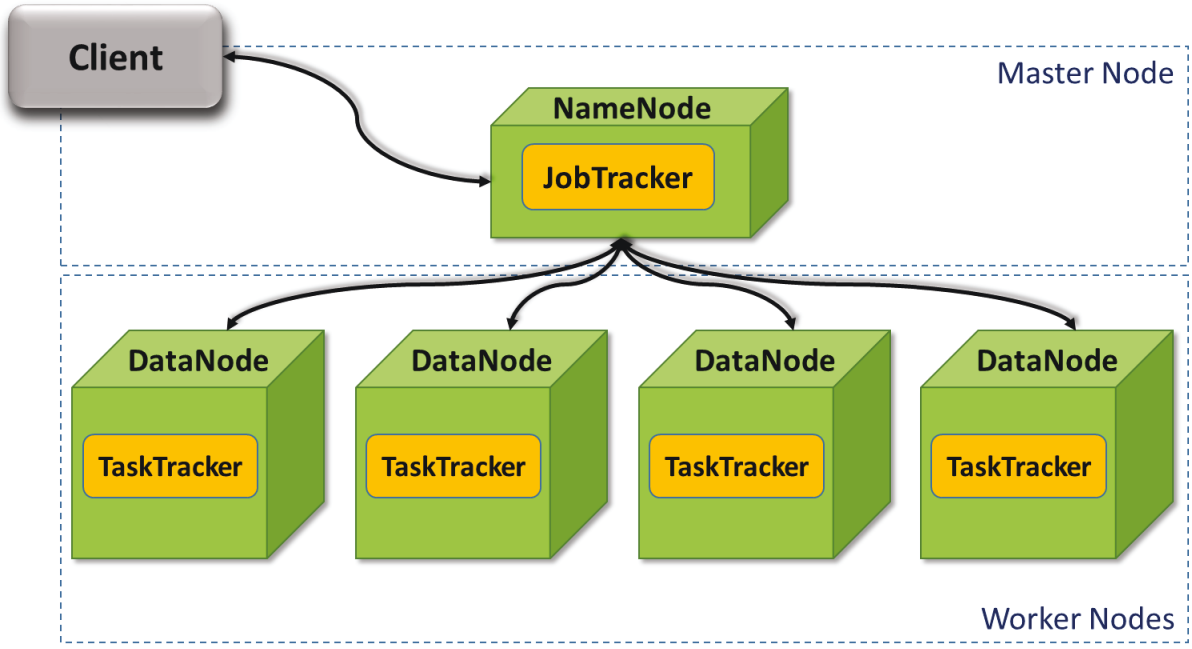

Figure 3.3: Hadoop master/worker nodes architecture

Enterprise version of Hadoop has two master, one is the main master and the other is the backup master in case of the main master dies.

3.4.3 MapReduce

The computational model used in Hadoop is called MapReduce [46], a programming framework for processing big data in parallel on multiple clusters of several computing nodes.

MapReduce programming not only provides a way to process a large scale of data over several machines, it also helps developers to simply use distributed and parallel systems, no matter if they have never worked with such systems.

MapReduce is also known as the "*paradigm shift*". It is in contrast to RDBMS database functions, by which data is passed to the function and the function does some computation over the data and returns the results; in MapReduce, the computation moves to the data rather than the data goes to the program. That is, the computation takes place where the data is, resulting in network load reduction and faster and more scalable computation (remember we are talking about petabytes of data). So, MapReduce codes written for megabyte data can easily scale up to terabytes and beyond.

A MapReduce program reads an input file from HDFS where the file is split into multiple chunks and passed to the different nodes. When a MapReduce computation runs, it consists of two key functions: *map* function, and *reduce* function. Each chunk is processed by the map function independently, which produces a set of intermediate (key, value) pairs. The reduce function then merges all intermediate values belonging to the same key according to the tasks the querier has specified for the reduce phase. This function finally releases the final aggregate output (either zero or one value per reduce invocation). Note that the mapper performs no aggregation, i.e., its job is mapping the data into (key, value) pairs for the reducer to aggregate. The data can be of any type like text file, audio, video, etc., almost all of which can be mapped into (key, value) pairs.

Actually, a MapReduce function can be a sequence of map and reduce tasks, fault-tolerant execution of which is guaranteed through this framework while scheduling them in parallel on any node. Normally, map and reduce functions are written by the querier.

To get a better understanding, we explain the MapReduce workflow through the common Word Count example, which counts the number of distinct words in a set of text documents. Using MapReduce, this computation can be done over a large collection of text documents in a reasonable amount of time. Suppose we have an input text file, including a couple of lines where each line is a *record*. As illustrated in Figure 3.4, the MapReduce algorithm has six phases, as follows:

1. The input text file is first read from the DataNode where the file is located.

2. The MapReduce job splits the file into multiple data chunks (records), each of which contains one line, so there would be three records as the file in this example has three lines.
3. Records are then passed independently to the mappers, which process received records in a completely parallel manner, and emit the list of (key, value) pairs. Here, the key is any single word and the value is its count, which is 1 per each word in every map output, i.e., (word, 1). After the map phase, a local combiner, as an optional processing step, can be applied to combine the same intermediate keys and their associated values produced from the map function.
4. In the shuffle and sort phase, all the emitted pairs from the map task are shuffled and sorted. That is, pairs with the same key will be grouped and passed to the same reducer. This phase involves many copies and coordination between nodes in the clusters.
5. The reduce task takes a set of (word, 1) pairs and simply sum up the values having the same key, and then prepares the final ordered list of words with their counts.
6. Eventually, the final result will be returned to the querier.

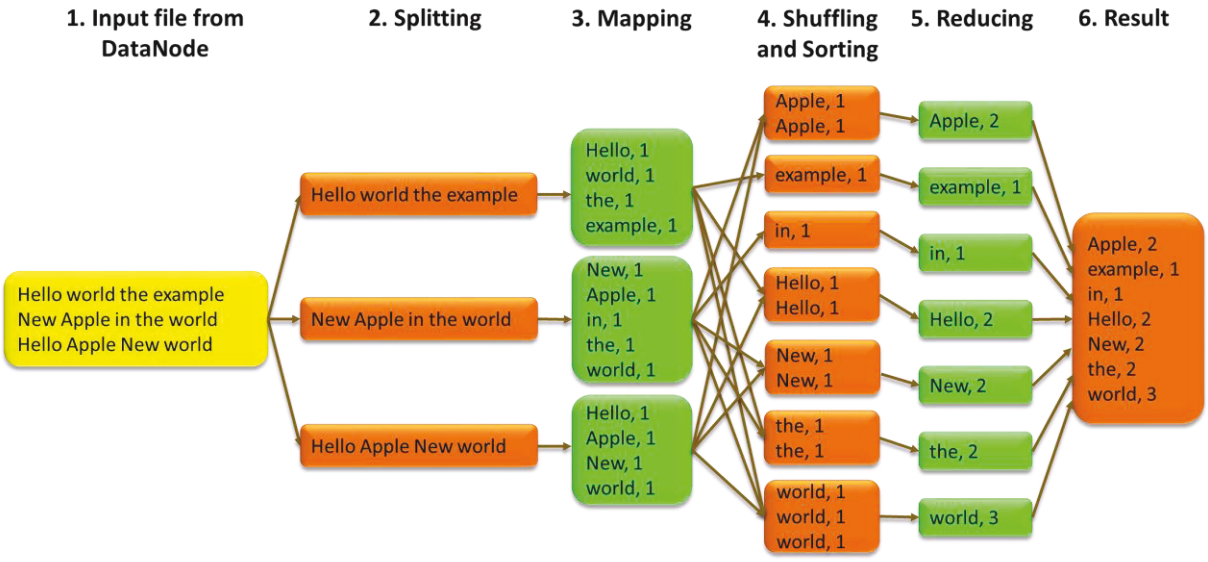

Figure 3.4: MapReduce process for Word Count

All the parts shaded in orange are handled by the framework itself, the querier only needs to provide green-shaded parts. In general, the MapReduce workflow may use multiple map and reduce stages.

With Hadoop, programmers no longer need to worry about where their data is located and how it is processed; how failures and data loss are actually handled; how to break computation into smaller pieces; and how to program for supporting the scalability (e.g., when a program is written for 1MB, it would also works well for 1000MB). Actually, Hadoop takes care of all of them and a programmer only focuses on writing the scale free program.

There exist other MapReduce frameworks such as Spark, Phoenix, Mars, and Blobseer. Spark[14], as an instance, runs programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark follows an advanced DAG (Directed Acyclic Graph) execution engine for Hadoop data that allows users to process large data sets with very high performance. It provides a simple API that support a wide range of programming languages, including Java, Scala, Python, and R.

Since Spark has some advantages over Hadoop, one may think Spark can replace Hadoop. However, it is not the case because spark is not able to properly work without Hadoop framework components like Hive, Pig, HBase, etc.

3.5 Summary

This chapter started with describing cloud computing as well as some of the most important concepts about cloud systems and providers, placing an emphasis on security and privacy issues for the cloud data. Next, we described the concept of Big Data as well as Hadoop MapReduce as a great framework to store and process it. Although MapReduce is a great

[14] http://spark.apache.org/

idea for handling big data, it raises new risk of leaking information; especially when it comes to sensitive data. Actually, an attack on the cloud where big data resides is not only possible, but likely. Therefore, strong privacy-preserving approaches are demanded in order to protect data privacy. In the next chapter, we study some privacy preserving methods over big and sensitive data. At the end of the next chapter, the reasons of choosing differential privacy among all other methods as well as its formal definition and properties will be discussed.

Chapter 4

Privacy-preserving Methods in Large-scale Data Analysis

Thousands of users outsource their data to cloud in the form of shopping transactions, healthcare data, clickstream, search logs, book and movie ratings, census, and so on. At the same time, protecting the privacy of the contributors in the data set is crucial; especially when it comes to sensitive data. The question here is how to release big data in a cloud computing environment without compromising its privacy.

Actually, cloud users need to be certain that their data stored in the cloud is still structured, regulated, and trustworthy. In case of healthcare data, for example, multiple third parties may want to mine the data and compute aggregate statistics in order to provide better services like investigating new drug, offering cheaper premium (by insurance companies), and so on. In this example, a participant might be willing to allow the insurance companies to compute aggregate queries by looking at the whole data, but he would be definitely unwilling to let those companies look at his specific data and figure out what kind of diseases he has and then misuse it to increase premium [3].

Hence, a challenge in this scenario is how to allow an untrusted program to run on the original data and ensure that the running program does not intend to leak individuals' information and violate their privacy. Many techniques have been proposed over the last decade to mine vast amounts of data in cloud systems without violating the privacy of its

individuals. One obvious way is to initially eliminate private and unique attributes, i.e., Personally Identifiable Information (PII), including names and Social Security Numbers (SSN), from the dataset and then release it. Clearly, this data de-identification alone is not sufficient to entirely protect data against linking attacks, which reconstructs individuals' identities using auxiliary datasets (recall re-identifying Massachusetts governor's record explained in Section 2.1). Therefore, after removing PII from data set, we need to apply other techniques that can successfully prevent disclosure of the identity information of individuals. In this chapter, we aim to study such techniques for privacy preserving data mining in cloud systems as well as their applicability and limitations.

4.1 Query auditing

Query auditing is a query restriction method with the goal of detecting and preventing leaks in a sensitive database. This method observes past behaviors of queries in order to determine whether answers to these kinds of queries can be used by an intruder to learn confidential information. Using this method, every new query is controlled and checked before running on the private database for any possible compromise and misuse of the data. The history of answered queries is kept and given the previous history of all the answered queries, the auditor decides whether a new query is allowed to be answered over dataset and returns accurate answer only when it is safe to do so [2].

However, providing privacy guarantees by this mechanism is not quite efficient and is a hard task due to the following reasons [3] [30] [47] [48] [49]:

- Data owner always worry if the querier is trustworthy or not, so the querier must convince the data owner that his query is safe enough to be run on data.
- To what extent is it practical to get all the untrusted MapReduce programs and audit them for correctness before executing them on private data? Obviously, it is impractical and very time consuming, because in reality, there exist so many programs whereas the auditor might have limited or even no access to their source code.

- By query auditing, a query is answered if it is safe to answer; otherwise the query is denied because it could potentially lead to compromise privacy. The problem arising here is that the refusal itself, combined with the answers to 'safe' queries, may be maliciously used to reveal sensitive data and leads to privacy breach, or even sometimes conservatively denies a safe query.
- It is computationally hard to decide whether an actual answer to a sequence of queries breaks any individual's privacy.

Therefore, many alternatives to query auditing have been proposed including adding random noise to the input data or the result, cryptographic techniques, and anonymization-based methods, we mention them in the following sections.

4.2 Data perturbation techniques

One way to preserve privacy of sensitive information is to mask input or output data through one of the perturbation approaches so that the computations will be run on the perturbed data. Perturbation techniques consist of two basic classes: (1) Input perturbation techniques, by randomly modifying the input data values, then running the queries on this modified data; and (2) Output perturbation, by running the queries on the original dataset, then adding random noise to the (actual) output of the computation. Both techniques have several pros and cons (described below) and they always need to control the trade-off between maximizing accuracy and minimizing privacy loss where smaller perturbation typically yields stronger violation of privacy [24].

4.2.1 Input perturbation

4.2.1.1 Keyword search on encrypted data

Data encryption in the cloud is widely used to encrypt sensitive data before outsourcing it to the cloud. This allows the cloud side to process encrypted data in order to maintain the secrecy of data against mischievous users [41]. PRISM (Privacy-Preserving Search in

MapReduce) is a secure search scheme compatible with any cloud computing environment utilizing Hadoop MapReduce. The idea behind PRISM is executing word search over encrypted data in parallel ("Map" phase), after which results of map phase are aggregated through the "Reduce" phase [50].

Similarly, Lin [51] implemented a system for preserving privacy of keyword search, comprising three parts that communicate via network:

1. *The data owner software*, which first encrypts images and tags (image decryption) owned by the data provider, and then emits search tickets as well as image decryption keys.
2. *The cloud software* that performs the search and image retrieval on encrypted data.
3. *The user software* that lets the end user perform tasks including search and image retrieval (and decryption) through a simple Graphical User Interface (GUI).

Suppose a data owner gives the search ticket to a user, who intends to issue a search query to all encrypted tags for a specific keyword. After performing this request by cloud and returning the encrypted result, the user needs to have the decryption key from the owner to access and see the image (see Figure 4.1).

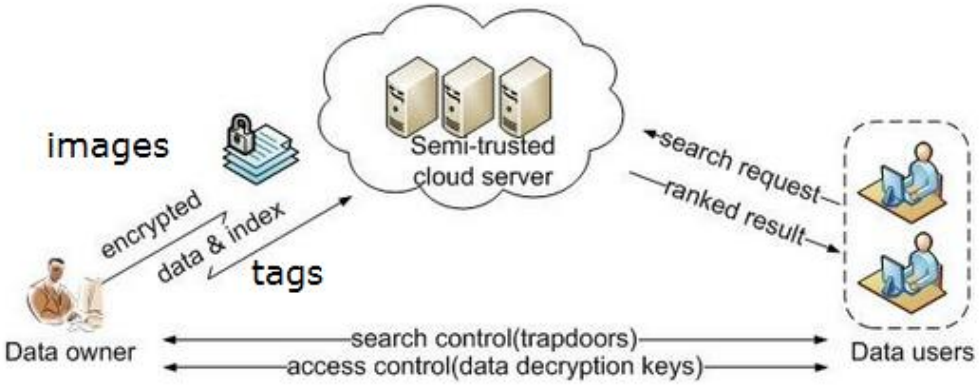

Figure 4.1: Architecture of the search over encrypted cloud data [52]

In both works, cloud is assumed to be "semi-trusted" but "curious", i.e., it is supposed to properly perform a given function, but may have intention to look at the individual data it works on. Both approaches are based on data encryption before uploading to the cloud and

the cloud side is not allowed to access unencrypted data, and thus it is not able to exploit any information from encrypted data it holds. Cloud only performs search queries on encrypted data and gives back encrypted results to the end user [50] [51].

However, encrypting all data sets, especially intermediate data that is frequently generated and recomputed during data processing, is very costly and time consuming. Because, most of the time, it is impractical to frequently en/decrypt the data sets while performing any computation over them. Instead of data encryption alone, Zhang et al. [53] proposed a novel approach that incorporates data encryption with anonymization techniques to reach cost-effective privacy preservation. This approach first identifies what intermediate data sets need to be encrypted and only encrypts that part of data, after which it anonymizes the rest of data to achieve privacy requirements. Compared to the approaches that encrypts all datasets, this approach can significantly reduce the privacy-preserving cost of intermediate data sets while the privacy requirements of data owners can still be achieved. This cost reduction is quite favorable for the cloud users who normally use cloud services in a pay-as-you-go fashion.

4.2.1.2 Input noise addition

As previously mentioned, data de-identification alone is not enough to preserve privacy of individuals. Hence, after the de-identification of data, other mechanisms such as noise addition must be employed to provide higher level of confidentiality for the remaining sensitive attributes (see Figure 4.2). To this end, several approaches add random noise to the inputs. Noise addition is captured by adding or multiplying a randomized value to private numerical attributes. After removing identifiable information from data sets as well as perturbing the remaining sensitive attributes through random noise, data usefulness is shrunk most of the time. In fact, input noise addition methods usually fall short of establishing a good balance between privacy and utility [54].

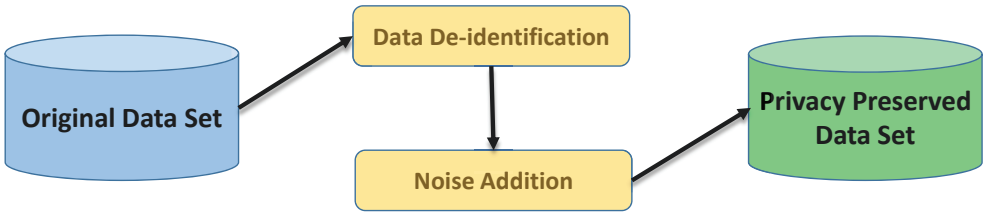

Figure 4.2: Preserving data privacy by data de-identification and noise addition

Another way to protect the identification of individuals in big data is anonymizing input data, using techniques such as *k*-anonymity, *ℓ*-diversity, *m*-invariance, and *t*-closeness. Data anonymization techniques anonymize large-scale input data sets before releasing them. An alternative anonymization-based method cannot be properly defined unless an understanding about the previous method is acquired. To this end, during the following sections, we study anonymization approaches based on the order in which they were invented.

4.2.1.3 *k*-anonymity

As we discussed in the beginning of this chapter, removing only PII from the database is not enough to completely prevent privacy violations. Therefore, Samarati and Sweeney [55] proposed *k*-anonymity, by which each record is syntactically indistinguishable from at least $k-1$ other records on every set of *quasi-identifier attributes*. Quasi-identifiers are sets of attributes (e.g., gender, date of birth, and zip code) that are not unique identifiers themselves, but can be combined with external data to create unique identifiers in order to re-identify individuals in the dataset [56].

In this method, each *equivalence class*[15] holds at least *k* records. Attributes are divided into two categories: 1. Sensitive attributes, are what the analysts require for their statistical analysis, so these attributes should be directly released (like Disease attribute in Table 4.1), 2. Non-sensitive attributes (aka quasi-identifiers) [7]. For instance, medical records are

[15] A set of records containing the same values for the quasi-identifiers [57].

shown in Table 4.1, and Table 4.2 shows a 3-anonymous version of the table. In this example, to achieve 3-anonymity, generalization and suppression techniques are applied on the quasi-identifier attributes, which are Age, Zip Code, and Gender.

	Non-sensitive			Sensitive
	Age	Zip Code	Gender	Disease
1	50	47677	Male	Flu
2	21	47602	Male	Heart Disease
3	28	47605	Female	Heart Disease
4	23	47606	Male	Flu
5	49	47678	Female	Heart Disease
6	30	47905	Female	Cancer
7	47	47673	Female	Flu
8	36	47909	Male	Cancer
9	35	47907	Male	Cancer

Table 4.1: Original patient's medical records.

	Non-sensitive			Sensitive
	Age	Zip Code	Gender	Disease
1	> 40	476**	*	Flu
5	> 40	476**	*	Heart Disease
7	> 40	476**	*	Flu
2	2*	476**	*	Heart Disease
3	2*	476**	*	Heart Disease
4	2*	476**	*	Flu
6	3*	479**	*	Cancer
8	3*	479**	*	Cancer
9	3*	479**	*	Cancer

Table 4.2: 3-Anonymous version of Table 4.1

k-anonymity can be seen as a remedy to defeat linking attacks, however, it provides no privacy guarantees against *homogeneity attacks* and also attacks using *background knowledge*. The background knowledge attack is due to the attacker's additional information about a *k*-anonymized dataset. Actually, attackers often have background knowledge about some of the individuals from somewhere outside the dataset such as websites, other databases, public records, newspapers, and so on. Based on additional knowledge, an attacker can conclude with near certainty whether an individual's information is contained in the dataset. A homogeneity attack occurs when information leakage is due to lack of diversity in an equivalence class [56]. For example, if an adversary knows a 35-year old male patient who has a record in the Table 4.2 and lives in the zip code 47907, then he realizes that the record number belongs to this patient can only be 6, 8, or 9. Since all of those records have the same disease (Cancer), the adversary learn with certainty this patient has cancer.

Using this method, each record must be indistinguishable among *k* records, which usually makes records too general to achieve accurate results [27]. Moreover, multiple publication

of the same dataset can completely violate the individuals' privacy. Also, *k*-anonymity can prevent *identity disclosure*[16] but not *attribute disclosure*[17].

Accordingly, a stronger definition of privacy has been proposed to consider diversity and background knowledge, it is called ℓ-diversity.

4.2.1.4 ℓ-diversity

ℓ-diversity model [56] was proposed to prevent some defects arising in *k*-anonymity model. With ℓ-diversity, privacy is preserved even if the data owner knows nothing about what kind of information an adversary has.

To resist homogeneity attacks, ℓ-diversity makes sensitive attributes "diverse" among each equivalence class, i.e., each equivalence class contains at least ℓ well-represented values for each sensitive attribute. In the previous example, manipulation techniques (i.e., generalization and suppression) can be differently employed on the quasi-identifier attributes to achieve 3-diversity (see Table 4.3).

	Non-sensitive			Sensitive
	Age	Zip Code	Gender	Disease
1	≥ 35	47***	*	Flu
5	≥ 35	47***	*	Heart Disease
9	≥ 35	47***	*	Cancer
2	≤ 30	47***	*	Heart Disease
6	≤ 30	47***	*	Cancer
4	≤ 30	47***	*	Flu
3	> 20	47***	*	Heart Disease
8	> 20	47***	*	Cancer
7	> 20	47***	*	Flu

Table 4.3: 3-diverse Version of Table 4.1

[16] Linking an individual to a specific record in a published dataset in order to re-identify it [57].
[17] Disclosing new information about an individual using released data, i.e., the characteristics of an individual in the released data can be derived more accurately than before releasing the data [57].

While ℓ-diversity is an important improvement to k-anonymity with respect to the homogeneity and background knowledge attacks, it has multiple limitations that we discuss below [57]:

- ℓ-diversity may be hard and unnecessary to achieve. Suppose we have a database of 1000 records, containing only one sensitive attribute that is the test result for HIV. It takes two values: positive and negative where 99% of the records are negative and only 1% are positive.
- ℓ-diversity is insufficient to protect data against attribute disclosure. This is because while it provides "diversity" for sensitive attribute values in each equivalence class, it does not consider semantic relationships among these values. From its point of view, values are only different points without having other semantic meaning.
- ℓ-diversity does not consider overall distribution of sensitive attribute values, which may affect privacy.

To prevent shortcomings arising in the two previous models, other approaches have been proposed including t-closeness and m-invariance that we briefly mention in the sequel.

4.2.1.5 *t*-closeness

Due to the limitations of ℓ-diversity discussed above, Li et al. [57] proposed t-closeness, which ensures that the distribution of each sensitive attribute in each equivalence is "close" to the distribution of the attribute in the entire database. More precisely, the distance among the two distributions must not be more than a threshold t, which is used to trade-off between utility and privacy.

In fact, requiring the closeness of two distributions would restrict the amount of useful information that can be learned by an adversary through published dataset. Because, it lessens the correlation between quasi identifier attributes and sensitive attributes, seeking to prevent attribute disclosure.

To apply t-closeness, a number of ways exist to measure the closeness (distance) between two probability distributions. However, most of the distance measures would not take into

account the semantic closeness of attribute values (recall the ℓ-diversity limitation related to the semantical closeness of sensitive values). To meet this requirement, the *Earth Mover Distance* (*EMD*) measure [58] is used, which considers the semantic distance between sensitive attribute values. EMD computes the distance among the two distributions. The EMD is actually based on the minimal amount of work that must be done to transform one probabilistic distribution to another, by transporting distribution mass between each other.

t-closeness protects against attribute disclosure by enforcing its requirement that limits an observer to obtain information about the correlation between quasi identifier attributes and sensitive attributes. Although this model addresses concerns found in ℓ-diversity and vulnerabilities in k-anonymity (like homogeneity and background knowledge attacks), it is not able to prevent identity disclosure.

4.2.1.6 *m*-invariance

With everyday and even every hour continuously increasing in datasets, we need techniques that support re-publication of data after any insertion in (or deletion from) it. Although the *k*-anonymity model and the related approaches developed novel solutions to data privacy preservation, they are limited only to static release of data. In these approaches, data is published only once and is not re-published after inserting into or deleting from it. In case of deletion, re-publication of dataset is even more critical and challenging than insertions of data. Anonymizing datasets statically may lead to poor data quality where queriers are deprived of accessing continuously updated data.

To solve this problem, Byun et al. [59] proposed an approach that dynamically anonymizes incremental datasets in a secure and efficient manner while considering data quality. They analyzed how to prevent inference channels where an adversary tries to leak information through comparing multiple re-publications of anonymized datasets. Then, they developed an algorithm that can efficiently insert new records into an anonymized dataset while satisfying the imposed privacy requirements. However, this approach supports only insertions of data not deletions.

m-invariance [60] was developed as the first privacy-preserving technique that supports anonymization of fully-dynamic databases in the presence of both insertions and deletions. The key idea of *m*-invariance is that, if a record *r* has been published several times, then all quasi-identifier groups including *r* must have the same set of sensitive values. This model prevents potential attacks with respect to re-publication and can effectively restrict the risk of privacy disclosure in data re-publication. Using this model, an adversary is prevented from inferring sensitive information by using possible correlations between multiple releases of a dataset.

With all above input perturbation methods, it is important to measure to what extent the real data and the masked data are similar; the more similarity, the less privacy but more utility, and vice versa. In addition, to what extent original data can be retrieved from distorted data. It is sometimes easy to get the original data from modified and perturbed data. All the anonymization-based techniques mentioned above intend to retain data utility while satisfying individual privacy. However, such intention for maintaining utility can help an adversary to drive patterns from anonymized data and use it to violate privacy of individuals with high likelihood. This is called *foreground knowledge attacks* that is in contrast to the background knowledge attack [61]. Furthermore, these approaches fail to withstand some other attacks such as *composition attacks*[18] [62].

Unfortunately, these anonymization-based methods do not support expressive privacy guarantees as well. AOL search logs [8] and the movie-rating records of Netflix subscribers [10] are two well-known fiascoes due to public releases of anonymized datasets. Sometimes, even faithful and secure execution of a query leads to disclose sensitive information about input [30]. Such failures motivate researchers to bring up a new approach to better protect data privacy, which is called *differential privacy* [3]. In the rest of this chapter, we discuss why differential privacy is a promising solution to provide stronger privacy for sensitive data.

---

[18] It occurs when multiple organizations independently publish anonymized data about overlapping individuals [62].

4.2.2 Output perturbation

A promising alternative to protect data privacy is scrubbing the output of the computation rather than input data. One of the well-known output perturbation methods is differential privacy that has recently emerged as a remedy to provide greater privacy guarantees. These guarantees hold independent of whatever auxiliary information an attacker has and also resist almost all vulnerabilities exploited through input perturbation techniques (e.g., composition attacks and so on). This method specifically distorts the results by adding random noise so that users submit queries to a dataset and get noisy answers.

4.2.2.1 Differential Privacy

Since input data may belong to multiple owners and come from various sources, access control alone is not sufficient to provide end-to-end privacy guarantees for big data stored in cloud environments. Furthermore, sensitive information about the inputs can be revealed through the output of an analyst's computation. Therefore, the result of an aggregate query is returned only when it does not disclose significant information about any single input.

As previously explained in Section 2.6, an individual's privacy can be violated even if that individual's record does not exist in the database. Thus, we need a notion of privacy that limits the risk of disclosing input sensitive information through the released results. The goal of differential privacy is to protect participants' information from disclosure, independent of any prior or posterior knowledge that an attacker may have. That is, an attacker learns nothing from the result of the computation and not being able to tell whether or not an individual has participated within the computation. In other word, any change to a single input has an insignificant and minor impact on the distribution of the computation's output. Thus, the presence or absence of any participant does not significantly alter the output of the computation. This is done by adding an appropriate amount of random noise to the result of an aggregate query in order to hide the actual result. Then by looking at this perturbed answer, an adversary is not able to derive any new knowledge about data [4].

Now we formally define differential privacy [4], [22]: Let D denotes a dataset whose individuals' privacy must be preserved. Two datasets D and D' are called *neighboring datasets* if they only differ in at most one item (or record), which exists in D but not in D'.

Definition 1 (ϵ-differential privacy [4], [22])**.** *A randomized algorithm or a function $\mathcal{F}$ is ϵ-differentially private if for all neighbor datasets D and D', and all possible outputs $S \subseteq Range(\mathcal{F})$,*

$$Pr[\mathcal{F}(D) \in S] \leq e^{\epsilon} \times Pr[\mathcal{F}(D') \in S] \quad (1)$$

The function $\mathcal{F}$ must be randomized (probability in the above definition is taken over the randomness of the function $\mathcal{F}$), and *Range($\mathcal{F}$)* denotes the output range of $\mathcal{F}$. If the input dataset is changed in one record, then the probability of any set of outputs *S* has only a very small multiplicative difference (e^{ϵ}). This formal definition considers any set of possible outputs and bounds the ratio of the probability that one of these outputs occurs when an individual is in dataset versus when it is not. It asserts that the results of the same query run on both D and D' should approximately be identical. Satisfying this definition addresses all concerns about any disclosure of personal information, regardless of presence or absence of any participant within the dataset, any auxiliary information that an adversary may have, and also the semantics of the records itself [4].

The parameter ϵ is a privacy parameter that is public and specified by the data owner (normally $0 < \epsilon < 1$) to manage the level of privacy. Sometimes, ϵ is interpreted as leakage, and hence its size is critical. Typically, a smaller value of ϵ provides greater privacy guarantees while with bigger values of ϵ we have less privacy but more utility. If ϵ is very small, then $e^{\epsilon} \approx 1 + \epsilon$ holds [24].

Composability: As we noted in Section 4.2.1.6, several anonymization-based methods such as *k*-anonymity and its variants, fail to provide privacy guarantees against composition attacks (recall that it leaks sensitive data through intersecting independent anonymized datasets). One of the most important features of differential privacy is that it is robust under composition, which means a composition of two differentially private queries is also

differentially private, i.e., a querier can submit query after query, without compromising privacy. Theorem 1 and Theorem 2 indicate sequential and parallel composition, respectively [30].

Theorem 1. *Let each randomized query Q_i preserves ϵ_i-differential privacy, then the sequence of $Q_i(D)$ over the dataset D preserves $(\sum_i \epsilon_i)$-differential privacy.*

Moreover, if a sequence of queries is issued on *disjoint* subsets of input, then the final privacy guarantee is acquired through the worst of privacy guarantees, instead of sum (see Figure 4.3).

Theorem 2. *The sequence of queries $Q_i(D_i)$ over a set of disjoint datasets D_i, each preserving ϵ_i-differential privacy, preserves $(\max_i \epsilon_i)$-differential privacy.*

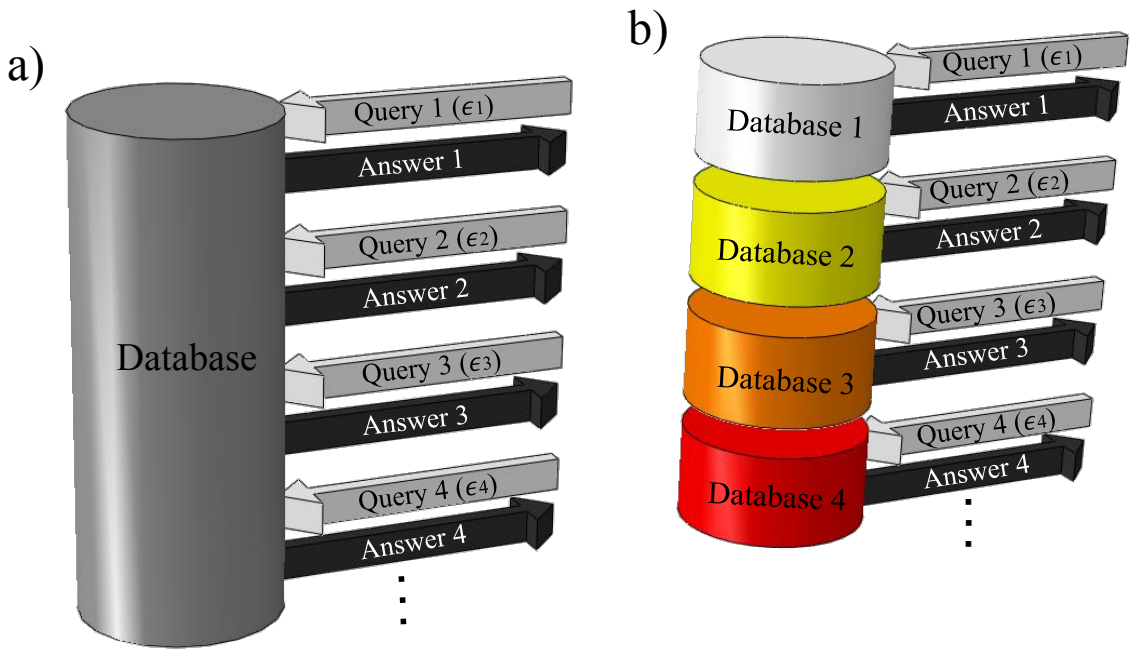

Figure 4.3: Composition properties: a) sequential composition and; b) parallel composition.

Group privacy: to achieve full privacy protection through differential privacy guarantees, we need to consider situations where input datasets differ not only in a single item, but in a group of items. Suppose an individual is part of a group of records that are collectively present or absent within a dataset, or the case where multiple records in the dataset contain

information of a single user. That is, in Definition 1, what if two datasets (here D and D') differ in more than one single item. The definition of differential privacy can extend to support group privacy using the composability properties. Therefore, to achieve ϵ-differential privacy for datasets D and D' differing in c elements [4]:

$$Pr[\mathcal{F}(D) \in S] \leq e^{\epsilon c} \times Pr[\mathcal{F}(D') \in S] \quad (2)$$

Note that we have stronger privacy guarantees for small c, and also when each participant has few records in the dataset [4].

To achieve ϵ-differential privacy, a properly chosen random noise must be added to the output $a = f(D)$, where f is a function (e.g., the average height of population in the dataset satisfying a given predicate) and D is the dataset. The question here is how much noise should be added to the result a? The magnitude of the noise directly depends on the *sensitivity* of the function f, which measures the maximum change of the function's result when any single item opts into, or opts out of, the dataset (i.e., maximum effect any single input can have on the output). The goal is to hide this effect to protect privacy [4], [24].

4.2.2.2 Function sensitivity

Dwork et al. [4] [24] define function sensitivity as follows:

Definition 2 (sensitivity)**.** *The sensitivity of a function $f : D \rightarrow \mathbb{R}^d$ under ℓ_1-norm is*

$$\Delta f = \max_{D,D'} \| f(D) - f(D') \|_1 \quad (3)$$

where the maximum is taken over adjacent datasets D and D', and d is the number of outputs.

The number of outputs is typically 1, i.e., $d = 1$ (computations that generate a single output). Normally, in calculating sensitivity, ℓ_1-norm[19] is used where $\| x \|_1 := \sum_{i=1}^{n} x_i$. By this definition, the sensitivity of a concatenation of *n* queries can be at most the sum up of the sensitivities of their individual.

Note that the sensitivity does not depend on the dataset and is actually a feature of the function, so it may be assumed that the sensitivity is known to the querier.

Many common functions have low sensitivity while some others have very high sensitivity. For example, counting the number of rows in a dataset has sensitivity 1 (i.e., $\Delta f = 1$), because removing any single row from the dataset can change the output by at most 1. By contrast, calculating the maximum height of the people in a database has high sensitivity, because single person's height can affect the final output by too much. Differential privacy works best (add the least noise) for small Δf, i.e., when the largest influence any single input can have on the output is low [4].

According to the probabilistic nature of differential privacy, its mechanism is necessarily randomness, i.e., it is a randomized method. Hence, to satisfy differential privacy, some mechanisms that rely on adding random noise are applied, such as *Laplace* mechanism. Dwork et al. [24] proposed Laplace mechanism, which takes a dataset D, a function f, and a parameter λ, based on which the amount of noise is specified. To mask the effect of a single input, this mechanism first calculates the actual answer $f(D)$. Then the actual answer is perturbed through adding noise according to a Laplace distribution, $Lap(\lambda)$. That is to say, $noisy\ output\ f'(D) = f(D) + Lap(\lambda)$. In doing so, the magnitude of noise is computed according to the sensitivity of the function as well as the value of epsilon.

Theorem 3. *Given a function* $f : D \rightarrow \mathbb{R}^d$*, the sensitivity of the function* (Δf)*, and the privacy parameter* ϵ*, Laplace mechanism adds random noise with distribution* $Lap(\Delta f/\epsilon)$ *to each of the* d *outputs, satisfying* ϵ*-differential privacy* [24]:

[19] https://en.wikipedia.org/wiki/Norm_(mathematics)

$$f'(D) = f(D) + \left(Lap(\Delta f/\epsilon)\right)^d \quad (4)$$

As an example, we want to compute noisy SUM. In general, the sensitivity of the function SUM is computed by the largest value in input dataset. Indeed, consider the case where all the inputs are either 0 or 1, then the sensitivity of the SUM is equal to 1 because the largest value within this input dataset is 1. Hence, to achieve differential privacy, we need to add a small amount of noise (i.e., $Lap(1/\epsilon)$) to the output of the SUM.

As a result, more sensitive queries lead to more data leakage about the presence or absence of an individual, resulting in adding more random noise to the output. Obviously, the outputs of such queries are most likely inaccurate. In general, as we have seen before, there is always a trade-off between minimizing privacy loss and maximizing utility in all differentially private computations.

4.2.2.3 Privacy budget

Different execution of the same differentially private query returns different answers. That is, if one asks the same query over and over again, then the average answers to the repeated query will finally converge to the true answer with high probability despite the noise. That is, the guarantee provided by differential privacy is degraded gradually under repeated queries. One possible solution might be giving the same answer to the same query, but it is not always easy to detect repeated queries. This is because with a sufficiently rich query language, the same query can be asked in many different ways, and consequently it is computationally undecidable whether two queries are identical [25].

Accordingly, we cannot issue any number of queries over a sensitive dataset and expect to have an absolute privacy guarantees as well. For that reason, the number of queries should be restricted in order to prevent an adversary from obtaining too much information about given dataset [3]. Moreover, differential privacy is composable, which allows the sequence of queries to be run on the dataset. This leads to gradually lessen guarantees of differential privacy [63]. To solve this problem, data owners impose a limit on how many queries can be answered on their database, ensuring that multiple queries do not degrade privacy.

The need to limit the number of computations coupled with the composability feature of differential privacy yield a notion of privacy, called "*privacy budget"*. Privacy budget actually limits the amount of information that a sequence of queries can obtain about any individual's information within a dataset.

The value of the privacy budget is normally fixed and set by the data owner up front to dictate his privacy requirements. Its value is then consumed gradually as queries are answered [3], [24]. Every time a new query is issued, the system first checks the remaining balance on the privacy budget and if it is adequately high, then allows the query to run on data. The budget is then deducted depending on how "private" each query is. Once the budget is exhausted, answering any more queries is refused and the querier can no longer get the results [64].

Having done so, an adversarial query "*Has Peter been diagnosed with HIV?"* could not acquire any accurate answer because its cost would exceed any reasonable privacy budget. For instance, if ϵ is a privacy budget, then a sequence of queries $Q_i(D)$ where each query satisfies ϵ_i-differential privacy, can be securely and independently answered as long as $\sum_i \epsilon_i \leq \epsilon$, without any privacy violation due to the aggregation of these queries. In this example, after submitting a ϵ_i-differentially private query, the system subtracts ϵ_i from the privacy budget (here ϵ), and then returns answer only if $\epsilon \neq 0$ holds [64].

Intuitively, to have a stronger privacy, a lower privacy budget should be specified, which in turn can impact several limitations on the data utility. However, almost all the robust and practical privacy mechanisms come at a price of loosing utility. Further, the way to properly set a privacy budget, as well as efficiently distributing this limited budget between multiple queries, is another issue [63] [65]. In fact, an inappropriate allocation of the privacy budget would cause incorrect data analysis and decrease the number of queries that can be securely answered on the database. In this regard, Mohan et al. [65] proposed a new model that efficiently distributes the limited privacy budget across various queries, so that more queries can be answered on the dataset.

4.3 Summary

This chapter covered the most important privacy-preserving techniques over big data in the cloud environments. We reviewed techniques including query auditing, input perturbation, and output perturbation afterward. Among all these approaches, we gave a detailed overview of differential privacy and pointed out why it is the "right" choice for protecting individuals' privacy in the cloud. In the next chapter, we will study existing differentially private platforms (PINQ and Airavat) and discuss their strengths and weaknesses, existing attacks to these systems as well as possible defenses.

Chapter 5

Differential Privacy Platforms

Airavat and *Privacy Integrated Queries (PINQ)* are two well-known platforms that are designed to perform differentially private data mining over sensitive datasets, without the need of human experts to use them.

5.1 Privacy Integrated Queries (PINQ)

McSherry [30] proposed PINQ, an extension platform that provides a programming interface to compute on unperturbed data using LINQ[20], while enforcing differential privacy requirements before running a query. That is, the programming language of PINQ and its run-time checks enforce differential privacy requirements.

Note that the main characteristic of this approach is that differential privacy guarantees are provided by the platform itself and both queries and data providers can use PINQ through a simple infrastructure and interface even without any particular privacy training and the help of differential privacy experts [30].

[20] Language Integrated Queries (LINQ) is a declarative language extension to the .NET framework that adds powerful query capabilities to the language syntax of C# for parallel computations. This language supports multiple SQL-like operations such as *Select*, *Where*, *GroupBy*, and *Join* [30].

This platform provides a thin layer in front of raw input data to not allow the querier to directly access the original data. The querier is rather remote and only able to submit queries to the system over a network (see Figure 5.1). However, McSherry [30] admitted this indirect access as the main limitation of his approach.

Compared to the previous methods, PINQ has two main advantages [30]:

1. PINQ enables the querier to run his computation against unmasked and unaltered data that gives high fidelity outputs. This is in contrast to anonymization-based methods, where the querier is not allowed to touch raw data.
2. PINQ runs the analyst's computation without any need to audit it. This is in contrast to interactive analysis where the querier must convince the data provider that his query is safe and the auditor decides which queries are OK.

The queries are thus constrained to differentially private implementations of current transformations and aggregations written in LINQ that return aggregate results to the querier. The queriers pose whatever queries through allowed transformations, and set the accuracy for aggregations. The data provider based on his privacy policies implements a *PINQAgent* that has an Alert method, where the *PINQAgent* ensures that the privacy budget is not exceeded. Each *PINQAgent* is supposed to return a boolean value that determines whether the request will be accepted or rejected. After forwarding each request, the budget is decremented [30].

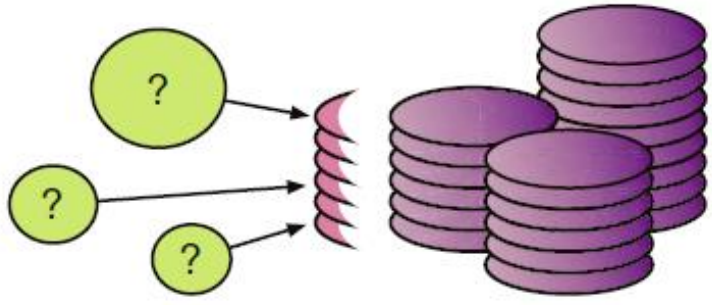

Figure 5.1: PINQ provides as a thin protective layer in front of raw input data [30].

Through this platform, the queries do not need to use a set of fixed aggregations which limit further usage; instead, they benefit from a programming language that enables them to

express almost any set of computations using a rich set of transformations followed by differentially-private aggregations. Main data operations supported by PINQ are divided into two types: transformation operators including ***Where***, ***Select***, ***GroupBy***, and ***Join***; aggregation operators that return the aggregate results after adding noise through differential privacy including ***NoisyCount***, ***NoisySum***, ***NoisyAverage***, and ***NoisyMedian*** [30].

To achieve differential privacy, ***NoisyCount*** and ***NoisySum*** apply Laplace Mechanism [24] while ***NoisyMedian*** and ***NoisyAverage*** apply exponential mechanism [26]. In PINQ, each aggregation takes parameter ϵ to satisfy ϵ-differential privacy regarding its underlying dataset. Before performing any differentially-private aggregation, the associated *PINQAgent* is checked through invoking the Alert with respect to the value of epsilon, after which the aggregation can be conducted only if the privacy cost of the query does not exceed the specified budget [30].

Another important operator supported by this platform is the ***Partition*** operator that is useful when a single dataset is shattered into multiple data sets. In this case, *Partition* operator can be used rather than using multiple *Where* in a query. As discussed in Section 4.2.2.1, Theorem 2 indicates that the final privacy cost associated with disjoint subsets of input is the maximum of privacy costs, rather than their sum. The *Partition* operator can be followed by any aggregation as well as other differentially-private computations, enabling a powerful recursive descent programming paradigm. For example, we can partition a database based on its distinct IP address, and conduct independent queries on each part with only the maximum cost within all parts. Accordingly, the privacy guarantee of a query depends on both "what" the query is going to return and "how" to express it [30], [66].

5.2 Airavat

Some of the privacy failures such as Netflix and AOL have been a motivation to Roy et al. [3] to propose a system for large-scale distributed computations. This system, namely *Airavat*, is a MapReduce-based platform to support end-to-end privacy guarantees. Airavat

is the first platform that incorporates mandatory access control with differential privacy and enables wide range of MapReduce computations on sensitive data (both trusted and untrusted). It provides strong privacy guarantees for any individuals' record within statistical databases particularly against untrusted code. It aims at maximizing the accuracy of answers to queries, while minimizing the probability of detecting any particular record.

As illustrated in Figure 5.2, Airavat has three main entities [3]:

1. *Data provider:* it is considered as the trusted party. Data provider sets several privacy parameters (e.g., epsilon, privacy budget, etc.) as well as different access control labels for their own data. This is done before uploading the data to Airavat without any need to audit all the submitted MapReduce codes.
2. *Computation provider:* this can be seen as an insurance company, a university researcher, or a drugstore person who is mostly considered as an untrusted party. This entity might attempt to access input values, intermediate values, or even the final output through malicious data mining algorithms written in a familiar MapReduce framework.
3. *Airavat framework:* it is a trusted cloud-based environment that comprises modified MapReduce, distributed file system, Java Virtual Machine (JVM), and SELinux as the underlying operating system.

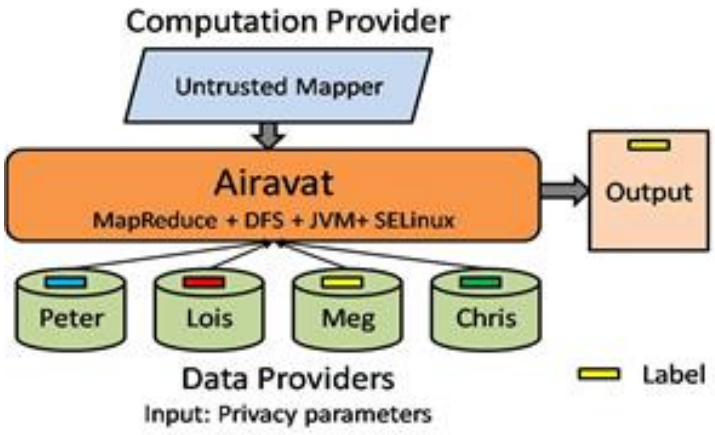

Figure 5.2: An overview of the Airavat architecture [3].

Data providers first specify privacy policies for their sensitive information through setting appropriate values of multiple privacy parameters. Then, Airavat enforces those pre-defined policies to confine computations performed by computation providers and ensures comprehensive protection for data against any leaks beyond those policies. Once the privacy budget has been exhausted, the mandatory access control will be triggered so that the output is no longer public and MAC is the only privacy protection mechanism. To demonstrate the practically of Airavat and its flexibility, the authors evaluated this system by performing several case studies including AOL search queries, a recommender system using Netflix data, and clustering and classification algorithms.

This work aims at addressing the main following questions:

1. What is the programming model?
2. How to enforce robust privacy guarantees to sensitive data?
3. What computations can be supported in Airavat?

Programming model

The programming model of Airavat is similar to standard Mapreduce framework, thus all developers and programmers, who are already familiar with this interface and programming language, can easily work with Airavat.

Airavat modified MapReduce framework through splitting it into "untrusted mapper" and "trusted reducer". Rather than writing the reducers by computation providers, Airavat provides them a few reducers and they are supposed to use those allowed reducers to write their MapReduce programs. The core goal behind Airavat is to confine the untrusted code and run it over un-modified dataset instead of auditing the code. Therefore, the only challenge here is how to confine the untrusted *mapper* codes, which can be any piece of Java code.

Privacy enforcement mechanisms

Untrusted mappers might exploit system resources to disclose information like copy data and send it over the network. To prevent any disclosure through system resources (e.g.,

files, network connections, names of running programs, etc.), Airavat runs on SeLinux [67] as the underlying operating system and modifies MapReduce framework in order to enforce SELinux's mandatory access control and support labels. The output of the computation itself might be an information channel as well. Suppose a query outputs 1 million if Alex bought some kind of drug; otherwise, outputs zero. In these cases, access control alone is not sufficient to prevent leaks through the computation's output. Hence, Airavat enforces differential privacy to ensure that the result of an aggregate computation will not reveal any information about any single input.

Both mentioned mechanisms (MAC and Differential privacy) have their own importance and Airavat needs to have them together to give an end to end privacy and security guarantee. To enforce them, the authors modified Java Virtual Machine (500 lines of code), MapReduce framework (2,000 lines of code), HDFS (3,000 lines of code), and SELinux policy (450 lines of code). In this system, differential privacy and MAC work interactively, it means that: *"if a MapReduce computation is differentially private, the security level of its result can be safely reduced."* Using Airavat, neither the data provider nor the computation provider requires to understand the complexity of implementation and enforcement of differential privacy [3].

As the computations computed by untrusted mappers and their corresponding sensitivity are unknown, Airavat requires computation providers to declare the range of their mapper outputs in advance. Airavat then uses it to estimate sensitivity. To that end, as illustrated in Figure 5.3, each mapper is equipped with a *range enforcer*. It ensures that if an output is produced outside the declared range by a malicious mapper, it will be replaced by a value inside the range while the computation provider is never notified of this change, because the notification itself would be a channel of information leak [3].

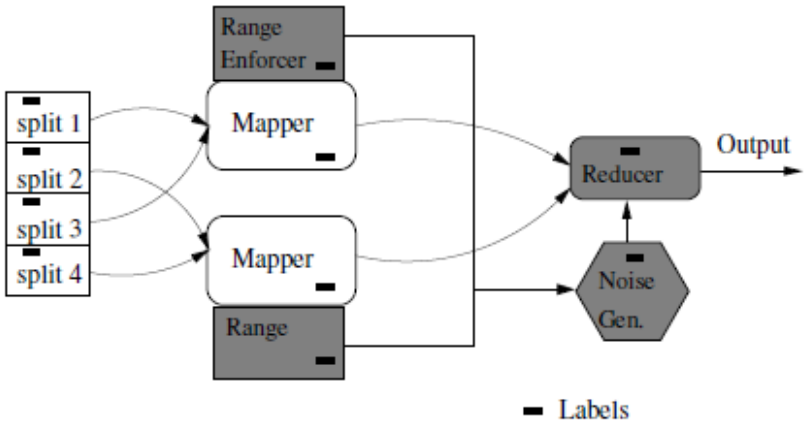

Figure 5.3: An overview of range enforcers. Trusted parts are shaded [3].

The range enforcer prioritizes privacy over accuracy in order to preserve privacy; however, the results may no longer be accurate or even meaningful. To enforce differential privacy, Airavat reducers add random noise to the output of MapReduce computations based on both the estimated sensitivity (exact sensitivity in case of trusted mappers) and the parameter ϵ (set by data provider) [3].

Supported computations in Airavat

Airavat supports some trusted reducers including *SUM*, *COUNT*, and *THRESHOLD* that can run directly on the mappers' output and are responsible for enforcing differential privacy through adding an appropriate amount of random noise to the output. One may consider this set of reducers very restricted but the authors declared that they would be sufficient to perform almost any data mining algorithms, process the search logs, and even write recommendation systems like Netflix [3]. In this regard, Tran and Sato [68] improved the utility of Airavat by extending it through allowing users to also write reducers by themselves, rather than using a fixed set of reducers provided by Airavat.

In general, both trusted and untrusted mappers are supported in Airavat. Many queries can be answered using untrusted mappers, e.g., the query *"How many iPhone 6s were sold today?"* can be answered with an untrusted mapper because the key ("iPhone 6s" in this case) is declared as part of the query prior to release the answer. By contrast, the query that directly outputs keys as part of its output must be produced only by trusted mappers.

Because outputting a particular key can leak information about an individual input. In this case, Airavat returns (noisy) answers associated with a key or list of keys only if the computation provider declares these keys as part of his computation. For example, the query *"List the top K items and their quantities sold in a store"* requires trusted mapper. The reason is that the query, in this example, prints item names in which keys are text strings that can open a storage channel for malicious mappers to leak information. Thus, Airavat does not return non-numeric values when the query outputs keys as the essential part of the result. With trusted mappers, more general queries are possible using *exact* sensitivity instead of range based estimates [3].

Systems like PINQ and Airavat provide robust privacy guarantees that allow any programmer to submit a batch of queries in a differentially private way without needing to know how to do Laplace or exponential mechanism. Among these two, Airavat seems to be more powerful than PINQ. The reason is that, PINQ provides a restricted programing language, through which the computation provider must rewrite his computation using trusted PINQ primitives. In contrast, Airavat takes untrusted user's computation, confine it, and then run it without modifying it. However, Airavat requires the querier to write his query in a MapReduce programming paradigm. Furthermore, Airavat offers end-to-end guarantees, while the guarantees of PINQ are mostly language level [3].

However, both systems are vulnerable to several covert channels by adversarial queries. Covert channels can break differentially private systems, because leaking even one bit of sensitive data can be enough to violate guarantees provided by these systems. As an example, a malicious query can easily send a bit to the adversary through sleeping the query for a long time when a specific record is found in the sensitive database [64].

Thus, differential privacy needs a defense against covert channels. Even restricting information leakage by refusing computations that leak more than a threshold number of bits does not provide full privacy protection. As we said, even one bit is too much. Furthermore, the refusal itself can leak one bit to disclose the presence of a given individual in the input data, resulting in an unacceptable loss in privacy [66].

5.3 Attacks on PINQ and Airavat

This section discusses several kinds of attacks to the mentioned system using covert channels. As previously discussed, revealing even a single bit of sensitive information may completely break differential privacy guarantees. Thus, covert channels can significantly destroy the systems, at which privacy mechanisms are based on differential privacy. The authors of both platforms discussed possible exploitable covert channels and tried to address them to some extent; however, neither system presented a fully protective defense. The reason is that both systems assume that the only thing a querier can acquire from dataset is the output of his query, however, this is not the case where the querier can obtain further observations from his query such as execution time, power consumption, query approval/refusal, and so on. Using these observations, different kinds of covert-channel attacks can be mounted against the underlying dataset. Therefore, the combination of all the things that can be observed by the querier must be differentially private [64].

To resist these channels, first the querier should be forced to be remote with no physical access to the system that holds dataset, so that he will be able to pose arbitrary queries only from a distance and receive his answer over a network. This is the *threat model* used in both PINQ and Airavat to close side-effect channels such as memory and power consumption, electromagnetic radiation, processor usage, disk activity, sound, light, and so on. By doing so, there remain only three channels: the decision whether or not the remaining budget is enough to answer a query, the time until the query is answered, and any modification to the value of global variables [64]. Now, we review and analyze these three channels as well as discussing the feasibility of these channels in PINQ and Airavat as the case studies.

Privacy budget attacks:

As previously mentioned, when a ϵ_i-differentially private query is issued, the system subtracts ϵ_i from the whole privacy budget ϵ, and executes the query only if ϵ remains nonzero. Therefore, once the budget is exhausted, the system refuses to answer further queries. However, this refusal itself can be exploited as a channel to leak private

information. That is, a malicious query might run some sub-queries that consume a lot of (or even exhaust all the remaining) privacy budget when a given condition is satisfied. Once the answer is returned, the adversary simply detects the change in the budget by checking how many more queries he can submit [64].

Timing attacks

This kind of attack uses *computation time* as a side-channel to leak information. A query can simply send a bit to an attacker through pausing for a specific time or invoking an infinite loop when a certain predicate is satisfied [64]. Suppose a query that always returns noisy sum as its output but takes 5 minutes if Bob has cancer and 1 micro second otherwise; then based on the considerably change in the query completion time, the adversary can infer that Bob has cancer. In the context of differentially private systems such as Airavat and PINQ, this assumed query is considered safe and differentially private. However, the adversary can learn private data with certainty by only looking at the execution time not the result itself (he actually might not care about the result of his query!). Infinite loops (non-termination) results in timing channel attacks as well.

Mapping operations are considered as *microqueries*, which computes over a single database row at a time. *Macroqueries* are some reducing operations that merge the outputs of running microqueries on each row of the database, and finally perturb the total output through random noise. In simple words, microqueries and macroqueries can be seen as mappers and reducers respectively. In both PINQ and Airavat, each query may comprises a sequence of chained microqueries along with macroqueries. Possible solutions such as *simple* microquery timeout or reducing the bandwidth of the covert timing channels by making all clocks noisy [69] are not completely effective. Moreover, billing information provided to the cloud users containing execution time can be viewed as a timing channel to leak private data, which cannot be completely closed through quantizing billing units (e.g., billing in multiples of $20) or grouping billing over a period of time (e.g., monthly) [3] [64].

Although, batch-oriented systems like MapReduce where most programs do not depend on time can result in a reduction in the utility of the timing channels; however, it is unrealistic to expect a system to thoroughly rule out all time-based channels [3].

State attacks (global variable attacks)

State attacks uses *global state variable* to create a channel between microqueries, so that microqueries can communicate using these variables. It happens when an adversarial query is able to access and modify the value of a static variable. In contrast to timing channel attacks at which the result of the query still remains differentially private, by this channel the result of the microquery is most likely changed if a particular record is detected by *any previous* microquery. Thus, the result no longer satisfies differential privacy [64].

As shown in Figure 5.4, the initial value of the state variable in this example is set to *false*. The result of each microquery can be influenced by the previous microquery, which satisfy the embarrassing condition while processing the row *r*. that is to say, if Bob has Cancer, then the value of the state variable is set to *true*, and all subsequent records return 1.

```
Boolean found = false;
Function f(Record r){
    if(found)       return 1;
    if(r.name = Bob && r.disease = Cancer){
            found = true; return 1;
    } else return 0;
}
```

Figure 5.4: an example of the state attack [64].

Thus, privacy can be violated in the systems, such as PINQ, through which global variables are accessible and modifiable by the querier. In this regard, Airavat takes a different way, which marks a microquery as “not differentially private” when it is about to alter a static variable. However, this way is not fully effective. The reason is that the adversary can observe whether he receives the result of his query or the quote “Sorry, that’s not differentially private.” A better solution would be to halt the microquery and give a default result and let the rest of the query proceed on the dataset [64].

The feasibility of all the above channels is acknowledged in both systems, among which PINQ is vulnerable to all of these channels. Although Airavat protects against budget attacks through pre-calculating sensitivity, it is vulnerable to both state attacks (to some extent) and timing attacks (completely). So, we need stronger systems that can completely close these kinds of channels [64].

5.4 Defence mechanisms against attacks

Some mechanisms have been proposed to resist covert-channel attacks discussed above. In this regard, Haeberlen et al. [64] proposed a system, namely "Fuzz", that supports a simple and special programming language based on the Caml Light runtime system [70], [71] for writing differentially private queries. The authors declared that this system could completely rule out all mentioned remotely exploitable channels where its functionality is approximately similar to PINQ and Airavat. An overview of Fuzz system is illustrated in Figure 5.5, containing three basic components [64]:

1. A *type checker* to close budget-based channels.
2. A *special programming language* to close global state-based channels, simply through not supporting global variables in its language.
3. A *predictable query processor* to rule out timing-based channels.

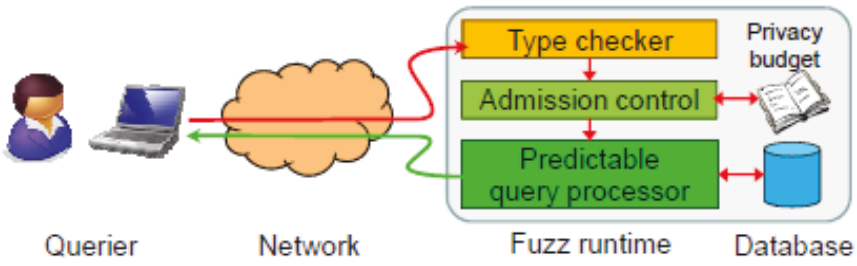

Figure 5.5: an overview of Fuzz system [64].

We now explain how these three components contribute to close discussed channels.

Defending against timing attacks: to avoid leaking information via query completion time, there are two scenarios [64]:

1. Time can be viewed as an additional output of the query so that a query would have two outputs: the output value, and the time taken by the query to complete. Therefore, the same mechanisms, i.e., sensitivity estimation and appropriate perturbation with noise already used for data outputs, would be taken into account for query time as well. It seems to be a promising and interesting approach, however, this is not an easy task to clearly calculate the sensitivity of a query in time domain.
2. An alternative is to ensure that a given computation takes the *same* amount of time that the result arrives for all possible datasets of a given size. To do so, the time has to depend only on the size of the dataset, which is supposed to be public. This can be achieved through making the completion time of the query predictable using the *predictable query processor* component in Fuzz, which enforces a fixed amount of time for each microquery. Making the query completion time depend only on the size of the database, and not its contents, avoids a potentially adversarial query from learning private information by looking at the execution time.

For each microquery m, in this scenario, the querier specifies a timeout T, then each time m processes a row r in the database [64]:

- If it is completed earlier than T, then Fuzz waits until time T elapses and produces the output of m afterward.
- If it is not completed within T, then Fuzz aborts the microquery and proceeds to the next row in order to deallocate any resources associated with that microquery before terminating T. However, such aborting itself can leak information to an adversary. In this case, the authors decide to return a *default value d* instead, i.e., Fuzz returns the answer to the query if the query completes within T, or d otherwise.

Accordingly, one may have two questions in mind [64]:

1. Do default values impact on utility?

2. Do default values violate privacy?

To address first question, in case of non-adversarial queries, default values are never returned if the querier choose timeouts properly. Therefore, default values impact on no queries, but adversarial queries. According to the second question, it may seem that default values open a data channel through changing the query's answer, but this is not the case: remember that Fuzz and therefore *d* must not depend on the contents of the dataset. Besides, timeouts are enforced on the microqueries not the whole query and default values should be chosen from the expected output range that a microquery terminates without a timeout. At the end, the final answer is noised (based on the sensitivity) before it is returned [64].

Defending against privacy budget attacks: since vulnerable systems to this kind of attack *dynamically* determine the privacy 'cost' of a query, the system's decision whether or not to execute a query can be influenced by an adversarial query. The reason is that an adversary can associate this decision with the contents of the private dataset and not with the queries. To close budget-based channels, queries are first type-checked through Fuzz that *statically* checks the cost of each query (rather than dynamically) before running it using the type system from [72]. Having done so, the deduction from the privacy budget does not rely on the contents of the database at all [64].

Defending against state attacks: to avoid microqueries from communicating through global variables, a microquery should be prevented from updating a static variable. To do so, language design of Fuzz does not support global variables, i.e., Fuzz disallow the querier to use global variable in his query. Thus, global variables can be neither accessed nor modified by any microquery [64].

Haeberlen et al. [64] pointed out that the implementation of Fuzz is practical and expressive enough to handle realistic queries and prevent all discussed covert-channel attacks at the same time. However, defensive mechanisms used in Fuzz come at the price of a greater query completion time, which is negligible compared to the defenses it provides.

In addition to Fuzz, Mohan et al. [65] designed GUPT, a platform for differentially private data mining, which is safe to all previously mentioned side-channel attacks. In the rest of this chapter, we mention some of the main specifications of GUPT very shortly. Note that for more details about this system and how its defence mechanisms exactly work, we recommend that you read the full version of the GUPT paper.

GUPT enhances the accuracy of data analysis while not shrinking data privacy, using a new model for data sensitivity called *aging of sensitivity*. The aging model of data sensitivity makes private data less sensitive over time through considering weaker privacy requirements for older records. GUPT uses this aged data to appropriately distributes the given privacy budget ϵ across multiple queries with respect to the querier's desired accuracy of the final output. This provides more accurate answer without shrinking the privacy guarantees. Less sensitive data can then be used to set optimal privacy parameters for executing queries on the newer data [65].

Contrast to PINQ, GUPT supports a wider range of unmodified computations with no need to rewrite them to be differentially private. Unlike Airavat, in which only mappers are any piece of codes written by the querier while reducers must be trusted and provided by the system; GUPT assumes the entire computation as a black box that can be completely untrusted or even malicious.

In addition, it eliminates many defects that exist in mentioned differential privacy systems, i.e., PINQ and Airavat, without sacrificing neither accuracy nor privacy. GUPT guarantees high performance and fast runtime through parallelizing the program across a cluster, ensuring a fine level of scalability for concurrent programs [65].

To defend against state attacks, GUPT computes the entire computation in an *isolated execution environment* in a manner that lets the querier observe only the final output and not any intermediate output or static variables. In case of privacy budget attacks, managing the privacy budget is handled by GUPT itself, rather than untrusted program. Defensive mechanisms in GUPT to handle timing attacks are the same as in Fuzz. Evaluation of GUPT demonstrates that GUPT boost the accuracy of data analysis and properly manages

the limited privacy budget allocation across various data analysis. Even non-expert programmers can easily work with GUPT [65].

5.5 Summary

This chapter first covered two famous differentially private systems, i.e., PINQ and Airavat. PINQ platform supports a limited set of primitives for data analysis while Airavat is the first system that tries to execute unmodified computations in a differentially private way; however, it requires that the querier breaks his program into an “untrusted” map program and a “trusted” reduce aggregator. We next showed that even differentially private systems like PINQ and Airavat can leak information through side-channels such as timing, state, and privacy budget attacks. Afterwards, we studied Fuzz and briefly GUPT, which are two proposed frameworks to successfully close these exploitable channels.

In the following chapter, inspired by discussed differential privacy platforms, specifically Airavat, we will think of an algorithm that integrates differential privacy with role-based access control (mentioned in Section 2.4).

Chapter 6

Integrating Differential Privacy and RBAC

Motivated by Airavat [3], we think about a combination of role-based access control and differential privacy. The motivation behind this idea is providing several levels of accuracy for both the input data and the results with respect to different users having multiple roles. That is, more accurate answers for more trusted roles while more general and noisy answers for others depending on what role(s) a user has in the system. The system normally returns only aggregate result that cannot under any circumstances reveal information about an individual unless the role that made this request has a high level of security. We have not tried our algorithm on any dataset to see its evaluation.

As previously discussed in Section 2.4, in RBAC model, policies will not be changed in case of reassigning people to different roles within the organisation, because permissions are specified in terms of roles rather than people.

Another motivation for using RBAC is role hierarchies, a way of structuring roles through the inheritance relation among the roles, by which a role (often a senior in the organisation) is allowed to inherit permissions from junior roles. In fact, role hierarchies can be viewed as an accumulation of permissions of other roles. For example, as illustrated in Figure 6.1, the physician role inherits the permissions of both the nurse and lab technician roles, as they are junior roles [14].

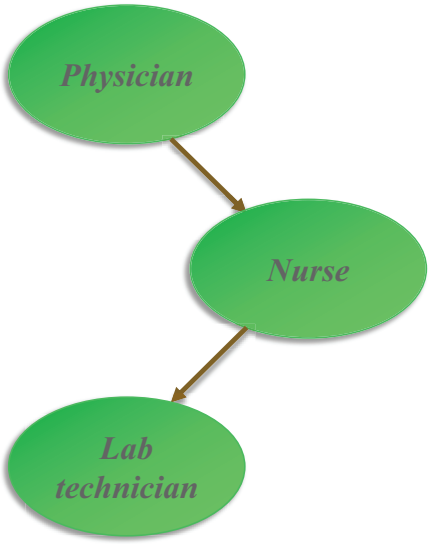

Figure 6.1: An example of role hierarchies.

This component of RBAC considerably simplifies access control administration by reusing the specification of a role through hierarchical relation between roles, which avoids the need to re-specify permissions for each role [73].

6.1 Who has access to what type of data?

Based on the role(s) a user has within an organization, he can access to certain type of data and submit either a limited or unlimited number of queries. In fact, the accuracy of answers to the users' queries depends on their role(s) in the system. So, the amount of random noise added to the result to achieve differential privacy and the number of answered queries varies according to their roles. An organization first needs to define different roles in terms of what information can be accessed by each role, and then assign permissions to each role. Thereafter, differential privacy guarantees will be enforced based on these roles' permissions.

Managing privacy budgets is an administrative issue for all differential privacy systems. In case of inefficient distribution of the privacy budget where a single privacy budget is specified for all roles, one role may consume budget more than it actually needs or

deserves. Some restrictions must be enforced on submitted users' queries in terms of different levels of access for different users' roles.

Therefore, to achieve differential privacy together with RBAC, the data owner needs to define different roles depending on what type of data he possesses, and then specify the following parameters:

- Different values of ϵ to control the level of privacy. As previously mentioned, smaller ϵ results in more privacy but less accuracy. For example, for computing the average of the scholarships and education grants on a university dataset, data provider set bigger ϵ for more trusted roles (e.g., director of department) leading to add less amount of noise to the output, and thus providing more accurate result.
- Different values of the privacy budget to ensure fairness. Same privacy budget can be set for all roles where system only subtracts different values of ϵ according to different roles from this budget after submitting a query. A better way to preserve privacy might be considering different values of budget based on roles is.

When a querier sends a request to access data, the system first checks his role(s) to review his access levels to decide how to accomplish his request. for example, an external querier may have a fixed and limited budget to submit his queries; while an internal querier not only have more access to all product and data, he also can run more queries (in the form of more privacy budget) while receiving more accurate answers (in the form of greater value of epsilon).

We use the same paradigm as Airavat, where users can write their queries in a familiar MapReduce framework.

Since a user can belong to different roles, the system enforces one of the following situations after he logs in [14] [20]:

1. He will obtain all of the permissions associated with his multiple roles at the same time (core RBAC).

2. He will gain either all permissions of junior roles or only inherits from immediate junior role based on either general or limited hierarchies respectively (hierarchical RBAC).
3. He will gain only permissions of the role by which he logs in, and during this session, his other roles' permissions are not accessible by him (constrained RBAC).

6.2 Noise addition method

As discussed above, for achieving differential privacy together with RBAC, different values for epsilon and privacy budget should be considered according to multiple roles within the system. To ensure higher level of confidentiality, we can also apply a noise addition method to the confidential quantitative attributes before publishing the dataset.

Therefore, more sensitive or confidential numeric attributes are first identified, after which these attributes are transformed through adding or multiplying a random noise according to the level of security considered for each role. Afterwards, aggregate queries can be issued to this perturbed database and Laplacian noise will be added to the final answer to achieve differential privacy.

At the same time, there would be some attributes in the dataset that are reserved and kept unperturbed for some specified roles. For instance, the system administrator role's queries would be run against raw database while satisfying differential privacy. This role can perform more operations on the database with less amount of noise for his final output, or even being able to have direct access to some or all the attributes of the database.

However, the noisy output may seem to be inappropriate to many applications that require accurate output, e.g., data mining and analysis over medical experiment. In such cases, we can consider less or zero amount of noise to the output of the computation only for trusted roles.

For illustrative purposes, we provide a simple example of combining differential privacy and RBAC shown in Algorithm 6.1. This algorithm is intended to calculate the average salary of the $male$ employees in a company who are more than 30 years old. In this

example, $salary$ attribute is assumed to be more private than Age attribute within the database D. So, we apply either additive or multiplicative noise addition methods to the values of the $salary$ attribute.

The data owner first defines multiple roles for his own database D, which contains employees' information. The Querier provides Map and Reduce codes to compute the average salary, which is computed by calculating SUM and dividing it by COUNT. *ConfAtts* is the set of confidential attributes (here $salary$).

After logging the querier to the database, his role is first checked and then permissions for his role are updated by inheriting from low-grade roles in case of using role hierarchies feature. According to the role, ϵ and privacy budget P_B is set, and also confidential attributes values will be transformed by adding or multiplying a randomized number.

The Map phase first checks if the amount of P_B is enough to proceed and then subtracts ϵ from this budget. Afterwards, each record r is passed to a map task that maps r into a list of key/value pairs where key is $gender$ and value is the perturbed salary amount satisfying the given condition, i.e., when $age \geq 30$.

In the Reduce phase, the reducer accepts the intermediate key (i.e., gender emitted from the Map phase) and a set of values for that key. It then sums up these values together. Then the true sum as well as the true count should be perturbed through adding Laplace noise in order to achieve differential privacy. And finally, noisy average will be calculated and returned to the querier.

Algorithm 6.1: an example of combining differential privacy and RBAC to calculate **NoisyAvg**.

Input:

- **The Data Owner**: $\boldsymbol{D}$, $\boldsymbol{Roles}$
- **The Querier**: **Map**, **Reduce**
- **ConfAtts**

Output:

- **NoisyAvg**

Process:

Check *Role* and **Update** *permissions*;
Set ϵ and P_B based on the *Role*;
Apply random noise to *ConfAtts*;
Map phase
Map (Integer key, String values) {
//key : record number
//values : record contents
if ($P_B < \epsilon$) {
TERMINATE
}
$P_B = P_B - \epsilon$;
foreach record r in D **do** {
While $r.gender$ **is** $male$ **and** $r.age$ **is** more than 30 **do** {
Emit (key: $gender$, $value$: $salary$)
}
end While
}
end foreach
}
Reduce phase
Reduce (String key, Iterator values) {
//key: $gender$
//values: list of $salary$ values val
Set COUNT and SUM to $zero$
foreach val in values **do** {
SUM+= val;
COUNT++;
}
end foreach
NoisySUM = SUM + $Lap(\Delta f/\epsilon)$;
NoisyCOUNT = COUNT + $Lap(1.0/\epsilon)$; // COUNT has sensitivity 1
NoisyAvg = (NoisySUM / NoisyCOUNT);
Emit (NoisyAvg);
}

Also, some other situations can be considered in this algorithm such as:

- Allowing some direct queries and returning true results for some roles while only aggregate queries and returning noisy or perturbed results for other roles.
- Enforcing a set of rules for all roles to prevent them from issuing queries that return highly private information like individuals' SIN numbers or bank accounts.

6.3 Using both private data and non-private data

Mechanisms based on differential privacy normally assume that data is entirely sensitive. This assumption often results in adding too much noise to the output, and thus affects its accuracy and usefulness. Note that this assumption is not always the case. Consider a database D consisting of both sensitive and non-sensitive attributes (D is not totally private). D might also include some records that are less sensitive than others (e.g., from customers who sign open-consent agreements to reveal publicly their data). In this case, we can add a new attribute that determines whether a record is highly sensitive, sensitive, or non-sensitive, and then we consider it for the computation. Besides, there might exist a public database about the same individuals that does not include any confidential information.

In these scenarios, if we treat the whole dataset as completely private, then we pay extra cost for preserving privacy of non-private data. Therefore, in differentially private algorithms, it would be reasonable to use non-private records or available public dataset as a post-processing step. This can significantly decrease the amount of augmented noise to the final output and boost the quality and accuracy of data mining without sacrificing privacy [28] [74].

6.4 Summary

This chapter investigated the feasibility of integrating differential privacy with role-based access control. In doing so, privacy parameters defined by data owners as well as the amount of noise added to either the input attributes or the final output, can vary considerably from one role to another. The goal was to provide better access to database

and more accurate result for some key and trusted roles while denaturing the results according to the significance of other roles within the system. Section 6.3 described that using non-private and public data can improve accuracy yet achieve differentially private outputs. The next chapter is dedicated to the conclusion of this dissertation as well as the suggestions for possible future research directions.

Chapter 7

Conclusion and Future Work

Protecting big data stored in the cloud environments is vital, especially when it comes to confidential information. Multiple researchers have proposed various approaches to increase security and privacy of the individuals in a large amount of data. They all have one main goal in their mind: preventing a malicious user from obtaining information more than intended. Reaching this goal will successfully result in reducing privacy breaches, increasing customers' confidence for releasing their sensitive data, and motivating people and organizations to adopt cloud computing services.

In this work, we have studied the most important privacy-preserving approaches over a large amount of sensitive data while picking differential privacy among all to give its detailed overview. The reason of choosing differential privacy is that privacy guarantees provided by the previously done work are either difficult to achieve and program, or vulnerable to different kinds of attacks and disclosures.

Although differential privacy seems promising, it introduces noise that may be inappropriate to many computations requiring accurate results. In addition, this approach may apply strong noise addition for high sensitivity function in order to protect privacy, shrinking data utility most of the time. That is to say, the guarantees of differential privacy are rather robust and direct, but can come at the expense of accuracy.

Therefore, we have proposed an idea of combining differential privacy and role-based access control to benefit from the strengths of both approaches. This combination can preserve privacy in a computation system using the MapReduce framework without the

need to audit untrusted code while providing different levels of accuracy according to the users' roles. Compared to other differentially private systems like Airavat, we expect to have better usability for our proposed algorithm, because we considered raw input dataset as well as more accurate results for trusted roles. Moreover, non-private records in a dataset and even public dataset about the same individuals can be used to further improve the accuracy of the output.

However, this dissertation has not demonstrated to what extent our algorithm is practical since we have not evaluated it on any data mining computations. We leave this part to the future work and we hope that this idea leads to a different and new approach to privacy-preserving researches.

Also, it may be worthwhile to study what else can be done to optimize the usability of differential privacy. While this technique is an important step beyond anonymizing-based techniques, it has a number of issues that need to be addressed.

For example, unique values within a dataset (values that appear one time) can create an exploitable channel to leak information about a specific item in the dataset. For example, an adversary may attempt to encode to a unique value. To address this problem, one may think that potentially malicious unique values can be replaced by a random number within a pre-defined range, or simply removed. However, there would be a possibility that the dataset has a lot of unique values. So, in such case, the result will be no longer correct. This is a disadvantage of unique value replacement way. Hence, we believe some scope of improvements still remain over differential privacy.

Another promising direction for future work can be developing more efficient privacy preservation techniques through combining two or more already existing techniques. The emphasis can be placed on providing an optimum level of data perturbation in order to maintain a perfect balance between data privacy and utility.

Bibliography

[1] "Welcome to Apache™ Hadoop®!" Available: http://hadoop.apache.org/.

[2] F. Y. Chin and G. Ozsoyoglu, "Auditing and Inference Control in Statistical Databases," *IEEE Trans. Softw. Eng.*, vol. SE-8, no. 6, pp. 574–582, 1982.

[3] I. Roy, S. T. V Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat : Security and Privacy for MapReduce," *Proc. 7th USENIX Conf. Networked Syst. Des. Implement.*, vol. 19, no. 13, p. 20, 2010.

[4] C. Dwork, "Differential privacy," *Proc. 33rd Int. Colloq. Autom. Lang. Program.*, pp. 1–12, 2006.

[5] D. Sun, G. Chang, L. Sun, and X. Wang, "Surveying and analyzing security, privacy and trust issues in cloud computing environments," *Procedia Eng.*, vol. 15, pp. 2852–2856, 2011.

[6] C. Dwork, "The promise of differential privacy: A tutorial on algorithmic techniques," *Proc. - Annu. IEEE Symp. Found. Comput. Sci. FOCS*, pp. 1–2, 2011.

[7] L. Sweeny, "k-anonymity: A model for protecting privacy," *Int. J. Uncertainty, Puzziness Knowledge-Based Syst.*, vol. 10, no. 5, pp. 557–570, 2002.

[8] M. Barbaro and T. Zeller, "A face is exposed for AOL searcher No. 4417749," *New York Times*, pp. 1–3, 2006.

[9] D. Kawamoto and E. Mills, "AOL apologizes for release of user search data," *CNET, August*, vol. 7, 2006.

[10] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," *Proc. - IEEE Symp. Secur. Priv.*, pp. 111–125, 2008.

[11] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays," *PLoS Genet.*, vol. 4, no. 8, p. e1000167, 2008.

[12] P. Samarati and S. De Capitani, "Access Control : Policies , Models , and Mechanisms," *Found. Secur. Anal. Des.*, vol. 2171, pp. 137–196, 2001.

[13] R. S. Sandhu and P. Samarati, "Access control: principle and practice." pp. 40–48, 1994.

[14] S. Harris, *All in One CISSP*. 2008.

[15] S. Osborn, R. Sandhu, and Q. Munawer, “Configuring role-based access control to enforce mandatory and discretionary access control policies,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 85–106, 2000.

[16] M. Nyanchama and S. Osborn, “Modeling Mandatory Access Control in Role-Based Security Systems,” *Database Secur. VIII Status Prospect.*, pp. 129–144, 1995.

[17] R. Sandhu and R. Sandhu, “Role Hierarchies and Constraints for Lattice-Based Access Controls,” *Proc. Forth Eur. Symp. Res. Comput. Secur.*, pp. 65–79, 1996.

[18] R. Sandhu and Q. Munawer, “How to do discretionary access control using roles,” *Proc. third ACM Work. Role-based access Control - RBAC ’98*, pp. 47–54, 1998.

[19] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta, “Privacy-aware role-based access control,” *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, pp. 1–31, 2010.

[20] R. S. Sandhu, D. Ferraiolo, and R. Kuhn, “The NIST Model for Role-Based Access Control: Towards A Unified Standard,” *5th ACM Workshop on Role Based Access Control*. pp. 47–63, 2012.

[21] C. Dwork, “Ask a better question, get a better answer a new approach to private data analysis,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4353 LNCS, pp. 18–27, 2006.

[22] C. Dwork, “Differential privacy: A survey of results,” *Theory Appl. Model. Comput.*, vol. 4978, pp. 1–19, 2008.

[23] C. Dwork, “An Ad Omnia Approach to Defining and Achieving Private Data Analysis,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4890 LNCS, pp. 1–13, 2008.

[24] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” *Proc. 3rd Theory Cryptogr. Conf.*, pp. 265–284, 2006.

[25] C. Dwork, “I’m in the Database, but Nobody Knows | Berkman Center,” 2010. Available: https://cyber.law.harvard.edu/interactive/events/luncheon/2010/09/dwork.

[26] F. McSherry and K. Talwar, “Mechanism Design via Differential Privacy,” *48th Annu. IEEE Symp. Found. Comput. Sci.*, pp. 94–103, 2007.

[27] X. Han, M. Wang, X. Zhang, and X. Meng, “Differentially Private Top-k Query over Map-Reduce,” *Proc. fourth Int. Work. Cloud data Manag.*, pp. 25–32, 2012.

[28] G. Jagannathan, C. Monteleoni, and K. Pillaipakkamnatt, “A Semi-Supervised Learning Approach to Differential Privacy,” *2013 IEEE 13th Int. Conf. Data Min. Work.*, pp. 841–848, 2013.

[29] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. Cambridge, MA, USA: MIT Press, 2006.

[30] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," *Proc. 2009 ACM SIGMOD Int. Conf. Manag. data*, pp. 19–30, 2009.

[31] Z. Wang, "Security and Privacy Issues within the Cloud Computing," *Int. Conf. Comput. Inf. Sci. (ICCIS), 2011*, pp. 175–178, 2011.

[32] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1–11, 2011.

[33] M. Hogan, F. Liu, A. Sokol, and J. Tong, "NIST Cloud Computing Standards Roadmap," *NIST Spec. Publ.*, p. 35, 2011.

[34] S. Nepal and M. Pathan, "Security, Privacy and Trust in Cloud Systems," *Springer Berlin Heidelb.*, 2014.

[35] A. F. Barsoum and M. A. Hasan, "On Verifying Dynamic Multiple Data Copies over Cloud Servers," *Eprint Arch.*, pp. 1–30, 2011.

[36] S. Kaur and A. Singh, "The Concept of Cloud Computing and Issues Regarding its Privacy and Security," *Int. J. Eng. Res. Technol.*, vol. 1, no. 3, pp. 1–5, 2012.

[37] T. Altameem, "A Replication-Based and Fault Tolerant Allocation Algorithm for Cloud Computing," vol. 4, no. 12, pp. 395–399, 2014.

[38] K. Popovic and Z. Hocenski, "Cloud computing security issues and challenges," *MIPRO, 2010 Proc. 33rd Int. Conv.*, pp. 344–349, 2010.

[39] A. Jacobs, "The Pathologies of Big Data," *Commun. ACM*, vol. 52, no. 8, pp. 36–44, 2009.

[40] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (IAAS)," *Int. J. Eng. Inf. Technol.*, vol. 2, no. 1, pp. 60–63, 2010.

[41] P. K. Tiwari and B. Mishra, "Cloud Computing Security Issues , Challenges and Solution," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 8, pp. 306–310, 2012.

[42] R. Magoulas and B. Lorica, "Introduction to big data. Release 2.0.," *O'Reilly Media, Inc.*, no. 11, 2009.

[43] P. Zikopoulos, D. DeRoos, C. Bienko, R. Buglio, and M. Andrews, *"Big Data Beyond the Hype" A Guide to Conversations for Today's Data Center*, vol. 33. 2014.

[44] Z. Majkić, *Big Data Integration Theory: Theory and Methods of Database Mappings, Programming Languages, and Semantics*. Springer Science & Business Media, 2014.

[45] “Apache Hadoop.” Available: https://en.wikipedia.org/wiki/Apache_Hadoop.

[46] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[47] I. Dinur and K. Nissim, “Revealing information while preserving privacy,” *Proc. twenty-second ACM SIGMOD-SIGACT-SIGART Symp. Princ. database Syst.*, pp. 202–210, 2003.

[48] J. Kleinberg, C. Papadimitriou, and P. Raghavan, “Auditing Boolean attributes,” *J. Comput. Syst. Sci.*, vol. 66, no. 1, pp. 244–253, 2003.

[49] K. Kenthapadi, N. Mishra, and K. Nissim, “Simulatable auditing,” *Proc. Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, pp. 118–127, 2005.

[50] E. O. Blass, R. Di Pietro, R. Molva, and M. Önen, “PRISM - Privacy-preserving search in MapReduce,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7384 LNCS, pp. 180–200, 2012.

[51] E. Lin, “Towards Privacy-Preserving Keyword Search via MapReduce Report on Project Completion,” pp. 1–4, 2012.

[52] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-preserving multi-keyword ranked search over encrypted cloud data,” *Proc. - IEEE INFOCOM*, vol. 25, no. 1, pp. 829–837, 2011.

[53] X. Zhang, C. Liu, S. Nepal, S. Pandey, and J. Chen, “A privacy leakage upper bound constraint-based approach for cost-effective privacy preserving of intermediate data sets in cloud,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1192–1202, 2013.

[54] K. Mivule, “Utilizing Noise Addition for Data Privacy , an Overview,” *Proc. Int. Conf. Inf. Knowl. Eng. (IKE 2012)*, pp. 65–71, 2012.

[55] P. Samarati and L. Sweeney, “Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement Through Generalization and Suppresion.,” *Proc IEEE Symp. Res. Secur. Priv.*, pp. 384–393, 1998.

[56] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, “l-diversity: Privacy beyond k-anonymity,” *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, p. 3–es, 2007.

[57] N. Li, T. Li, and S. Venkatasubramanian, “t-Closeness : Privacy Beyond k-Anonymity and ℓ-Diversity,” *Data Eng. 2007. ICDE 2007. IEEE 23rd Int. Conf. on. IEEE*, no. 3, pp. 106–115, 2007.

[58] Y. Rubner, C. Tomasi, and L. J. Guibas, “The earth mover’s distance as a metric for

image retrieval," *Int. J. Comput. Vis.*, vol. 40, no. 2, pp. 99–121, 2000.

[59] J.-W. Byun, Y. Sohn, E. Bertino, and N. Li, "Secure anonymization for incremental datasets," in *Secure Data Management*, Springer, 2006, pp. 48–63.

[60] X. Xiao and Y. Tao, "M-invariance: towards privacy preserving re-publication of dynamic datasets," *SIGMOD Conf.*, pp. 689–700, 2007.

[61] R. C.-W. Wong, A. W.-C. Fu, K. Wang, Y. Xu, and P. S. Yu, "Can the Utility of Anonymized Data be used for Privacy Breaches?," *ACM Trans. Knowl. Discov. from Data*, vol. 5, no. 3, p. 11, 2009.

[62] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith, "Composition Attacks and Auxiliary Information in Data Privacy," *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. data mining. ACM*, pp. 265–273, 2008.

[63] C. Clifton and T. Tassa, "On syntactic anonymity and differential privacy," *Trans. Data Priv.*, vol. 6, no. 2, pp. 161–183, 2013.

[64] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential privacy under fire," *USENIX Secur. Symp.*, p. 33, 2011.

[65] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. E. Culler, "GUPT: Privacy preserving data analysis made easy," *Proc. 2012 ACM SIGMOD Int. Conf. Manag. Data*, pp. 349–360, 2012.

[66] F. McSherry and R. Mahajan, "Differentially-private network trace analysis," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 123–134, 2010.

[67] B. McCarty, *SELinux: NSA's Open Source Security Enhanced Linux*. O'Reilly, 2005.

[68] Q. Tran and H. Sato, "A Solution for Privacy Protection in MapReduce," *Comput. Softw. Appl. Conf. (COMPSAC), 2012 IEEE 36th Annu.*, pp. 515–520, 2012.

[69] W.-M. Hu, "Reducing timing channels with fuzzy time," *Proceedings. 1991 IEEE Comput. Soc. Symp. Res. Secur. Priv.*, 1991.

[70] "Caml Light website." Available: http://caml.inria.fr/caml-light/.

[71] X. Leroy, "The ZINC experiment: an economical implementation of the ML language," no. 1, p. 100, 1990.

[72] J. Reed and B. C. Pierce, "Distance makes the types grow stronger," *ACM SIGPLAN Not.*, vol. 45, no. 9, p. 157, 2010.

[73] N. Damianou, A. K. Bandara, M. Sloman, and E. C. Lupu, "A survey of policy specification approaches," *Dep. Comput. Imp. Coll. Sci. Technol. Med. London*, vol. 4, no. April, pp. 1–37, 2002.

[74] Z. Ji, S. Diego, S. Wang, and L. Ohno-machado, “Differentially private distributed logistic regression using private and public data,” *Kdd’13*, vol. 7, no. Suppl 1, p. S14, 2013.

[75] I. Foster, “What is the Grid ? A Three Point Checklist,” *GRID today*, vol. 1, pp. 32–36, 2002.