UNIVERSITÀ DEGLI STUDI
DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE
DIPARTIMENTO DI INFORMATICA (INF/01 INFORMATICA)

DOTTORATO DI RICERCA IN INFORMATICA
SCUOLA DI DOTTORATO IN INFORMATICA, XXV CICLO

Privacy-Preserving Query Processing by Multi-Party Computation and Encrypted Data Outsourcing

Tesi di dottorato di ricerca di:
Maryam Sepehri

Relatore:
Prof. Ernesto Damiani

Correlatore:
Prof. Stelvio Cimato

Coordinatore del Dottorato:
Prof. Ernesto Damiani

Anno Accademico 2012/13

Abstract

Privacy-preserving query processing (P^3Q) techniques are increasingly important on partitioned databases, where relational queries have to be executed on horizontal data partitions held by different data owners. To conduct queries on the entire data partitions, the data owners may jointly collaborate to one another for sharing their private data or delegate them to an external service provider. In the literature these two solutions are referred to multi-party computation (MPC) and data outsourcing (DO), respectively. On the other hand, when no data owner or external service provider can be trusted enough to know all the inputs, privacy becomes a primary concern. To this purpose, data owners are not willing to share plaintext data with other parties or outsource plaintext to the service provider as well. A traditional solution to ensure privacy protection consists in adopting encryption scheme in order to help preventing information leakage. Such traditional solutions however reduce query execution efficiency notably in MPC scenario with large size data. This introduces the need to develop efficient techniques for P^3Q, allowing data owners to respect data privacy when collaborating during the execution of queries.

Recently, many techniques for P^3Q have been developed in the multi-party context, which are based on the application of secure multi-party computation (SMC) protocols. While these solutions have focused on increasing the privacy, efficiency has been only marginally addressed. For this reason, in this thesis we describe a scalable approach for computing privacy-preserving queries on the entire relation(s) without sharing their private partitions. Our solution is applicable to a subset of SQL query language called SQL^{--} including *selection* and *equi-join* queries. In order to nicely scale with large size data, we show how computation and communication costs can be reduced via a novel bucketization technique. We consider the classical notion of query privacy, where the queries only learns as little as possible (in a computational sense) about the query. To ensure such privacy, our technique involves a trusted third party (TTP) only at the beginning of the protocol execution. Experimental results on horizontally partitioned, distributed data show the effectiveness of our approach.

We also consider the problem of encrypted data outsourcing (EDO) where the owners encrypt their sensitive data with their own keys and outsource their partitions to a cloud service provider. This case poses a significant challenge to a cloud service provider, since the queries should be

computed over data encrypted with different keys that the cloud must not know (we refer to this setting as multi-key data outsourcing). This problem has been addressed for instance with expensive techniques like of key sharing or exhaustive re-encryption by the server. In this thesis we address this challenge by introducing a novel security solution, which applies proxy re-encryption (PRE) scheme to bring data encrypted with different keys under the same key, making cloud service provider searching feasible (we refer to this setting as single-key data outsourcing). The adopted technique relies on proxy server to transform data encrypted with the public key of different owners, so that the query result on the entire partition(s) can be decrypted by the user. This is done without the need for single data owner to release its secret key, and more importantly the proxy server does not learn the content of data processed.

Overall, the key research area of this thesis is to conduct SQL^{--} queries without disclosing any data owner's private data by SMC and EDO paradigms to determine the advantages and drawbacks of each paradigm in terms of security and efficiency.

Aknowledgements

I feel greatly privileged that God gave me the patience, perseverance, and courage to finish this thesis.

First and foremost, I would like to thank my principal supervisor, Ernesto Damiani, for his guidance when I needed it. I am very grateful for his support and trust. I would also like to thank my co-supervisor, Stelvio Cimato, for his fruitful discussions and involvement with my research study. Beside my advisors, I am thankful to my thesis referees, Setsuo Tsuruta and Omar Hasan, for their insightful comments.

I am very grateful to Mohammad Sadegh Astaneh for introducing Università degli Studi di Milano to me. He helped me find my way and I hope that I can pay him back one day.

I would like to take this opportunity to thank Lorena Sala, Antonio Capoduro and Paolo Arcaini for helping me adjust to the new environment and made me feel comfortable especially during the first year of my stay in Italy.

I also thank Danilo Ardagna for giving me a chance to work with him as a lab assistant in the last year of my Ph.D. course. I value his friendship and support as well.

I would like to thank my brothers and sisters for their help and encouragement. They are always supportive on my choices.

Last but not least, I would like to express my deep appreciation to my best friend, Marco Frasca, for his support in every possible way. He continuously pushed me to do my best and reinforced my strength in times when I doubted myself.

I dedicate this thesis to my parents, I would not be here without their love.

Contents

Chapter 1

Introduction

The increasing need for computing user queries across databases belonging to multiple data owners, while keeping privacy, makes the Privacy-Preserving Query Processing (P^3Q) one of the most actual and intensively studied problems in the area of databases and cryptography. In real life, there are many cases that data is partitioned horizontally across multiple owners, and the owners may wish to perform common function on the entire of data without revealing private information of the involved participants.

In order to make accurate analysis and knowledge extraction on all partitions, data owners can share data with other parties, which refers to multi-party computation (MPC), or outsource data to a service provider, which refers to data outsourcing (DO). These two main paradigms can be briefly described as follows:

- MPC paradigm. This technique deals with two (or more) distrusting parties want to jointly compute a commonly agreed function, but with the restriction that none of the participants can learn anything more than their own input and the public output. The interest in MPC paradigm has now become very high, since collaborations among mutually distrustful partners are increasingly frequent: there are many cases where data on the performance or security of a coalition is independently gathered and held by multiple actors, who then wish to compute a common metrics on the union of data without revealing their local information.

- DO paradigm. In this technique, the main idea is to have an external server where each data owner outsources its own data for the storage, maintenance and retrieval. However, since this server is typically not fully trusted and the data are not under the control of data own-

ers, concerns with data privacy increased. Thereby, a widely adopted solution is encrypting the data before outsourcing. We call such a technique encrypted data outsourcing (EDO). This technique is well motivated in practice due to the increasing popularity of Cloud Computing [3].

The problem of conducting MPC while preserving data privacy is known as Secure Multi-party Computation (SMC), which was initially suggested by Andrew C. Yao in [68]. In the most general sense, SMC problem has a general solution by means of combinational circuits [68]. However, communication costs for circuits makes this solution impractical in multi-party setting. Another common strategy to solve SMC problem is to assume the existence of a trusted third party (TTP), to whom the inputs can be disclosed [43]. Nevertheless, since this party should be fully trusted, this solution may not be always acceptable. Therefore, an existing trend in recent years is the study of multi-party protocols, which avoid a central point of trust as far as possible. To this purpose, the research community has developed a wide range of P^3Q techniques for answering different types of queries, which are based on homomorphic encryption [25, 41] and oblivious polynomial evaluation [51]. While these techniques offer strong privacy guarantees, they do not scale well for large size data because of using heavy weight cryptographic operation or several rounds of interactions among parties. An innovative method in this context has been proposed by Agrawal et al. [1], which performs intersection and join operations in two-party setting with minimal information sharing. Their solution is based on commutative encryption without the need of third parties. However, it suffers from a large amount of encryption/decryption costs and accordingly increase the query response time.

In this thesis, we propose a solution to address these complexity deficiencies. In our solution, data owners use SMC to compute privacy-preserving queries on the virtual relation(s) they jointly hold, without sharing their private partitions. Our own query execution protocols support a small but significant subset of SQL query language we call SQL^{--}, which includes equality, range and equi-join queries. In order to nicely scale up with large size data, we show that both the computation and communication costs of our scheme can be reduced via a novel bucketizaton technique. Of course, due to the deterministic nature of encryption, multiple runs of our protocols may leak some partial information about queries; we discuss some countermeasures in Section 3.4. As far as privacy analysis is concerned , we consider an extension of the classical twofold notion of query privacy, where (i) the

querier only learns query results (and what can be inferred from it), and (ii) data owners learn as little as possible (in a computational sense) about the query. Namely, we add a notion of query anonymity: protocol participants must not be aware of the source of the data they are not allowed to read. To ensure such privacy, our technique involves a trusted party (TTP), deployed only at the beginning of the protocol execution[1]. Our extensive experimentation carried out on horizontally distributed data, shows the effectiveness of our approach in terms of computational and communication costs.

On the other hand, in data outsourcing scenario, the challenge is how to answer the problem of inquiring data encrypted with different keys (multi-key data outsourcing) managed by an untrusted cloud service provider without leaking information. There have been a wide variety of studies on P^3Q, which allow the server to search over encrypted data based on bucketization techniques [30, 34], specialized data structure [18, 60], and order preserving encryption method [1]. These techniques however are not applicable in our scenario as long as they have tailored to single-key setting where data are encrypted under the same key, and moreover they are rely on indexing methods, which are vulnerable to inference and linking attacks that can compromise the protection granted by encryption. An alternative solution is to allow the server to superencrypt data with the owners' keys, resulting data under the same key. While this solution is feasible in multi-key setting, it introduces heavy computation overhead of encryption at server side. To this purpose, in this thesis we propose multi-key protocols for computing SQL^{--} based on proxy re-encryption scheme. This solution relies on a proxy server to avoid workload at cloud service provider side. We transform information that was encrypted under the key of single data owner so that it can be decrypted by the user without the need for the owner to release its secret key. This approach introduces less computation overhead and also preserve privacy at proxy side.

Overall, the aim of this thesis is to develop SMC and EDO query processing techniques to understand their advantages and limitations in supporting the SQL^{--} language in terms of efficiency and privacy. The organization of the thesis is as follows:

> **Chapter 2** reviews the existing protocols for the private set intersection problem, which is a fundamental operation for performing equality test queries. It also presents techniques proposed for SQL^{--} queries

[1]While trusted computing is outside the scope of this thesis, we remark that our TTP executes simple operations, and lends itself to implementation via a secure coprocessor, in the line of a seminal work by Agrawal et al. [1].

in SMC scenario.

Chapter 3 describes a scalable protocol B-SMEQ to address the equality test problem in SMC scenario, which reduces the communication and computation costs adopting bucketization technique. It proposes an extension of B-SMEQ to execute privacy-preserving range and equi-join queries over partitioned data in scalable manner.

Chapter 4 reviews the state of the art of privacy-preserving query processing methods in data outsourcing paradigm. It describes techniques in the literature related to SQL^{--} queries and compares them in terms of privacy and efficiency.

Chapter 5 states an encrypted data outsourcing scenario in multi-key setting and presents a novel solution to manage SQL^{--} queries over data encrypted with different keys via proxy re-encryption scheme.

Chapter 6 summarizes the contributions of this thesis and outlines future work.

Chapter 2

Secure Multi-Party Computation

In this chapter, we review the known specialized protocols related to privacy-preserving query processing (P^3Q) in the area of Multi-party Computation (MPC). We start by looking at the importance of MPC in research community, and then study protocols related to the P^3Q to support SQL^{--} queries including equality test, range and join queries.

2.1 Introduction

Nowadays, large amounts of data are made available and shared in collaborative scenarios where multiple data owners put together the information they have on the purpose of making accurate analysis and knowledge extraction. When data privacy is a concern, however, data owners are not willing to share plaintext data with other parties. This is the case when legal constraints apply to the shared data (as for example data belonging to parties in hospital databases) or parties are at the same time competitor for commercial of financial reasons (as for example bank databases or list of customers from different competing companies). A solution to this problem comes from the application of multi-party computation (MPC) protocols, where two (or more) participants are enabled to jointly perform a computation over their inputs. Each party contributes its own data to the other participants without revealing any information irrelevant to the computation. Today this paradigm has emerged as an important area of research in privacy-preserving computation problems such as P^3Q. To execute query processing in MPC in a privacy manner, secure multi-party computation (SMC) techniques have been introduced, providing a general solution to execute any computation

by considering the transformation of the computed function in combinational circuit [69]. While this general solution is quite efficient in two-party case, it introduces high communication cost as the number of parties increases and the complexity of the computation grows. Over the years, the research community has developed a wide range of privacy-preserving techniques to realize different functions across the shared databases based on *homomorphic encryption* [41, 45], *oblivious polynomial evaluation* [25] and *commutative encryption* methods [1]. Although most of these techniques offer strong privacy guarantees, they do not scale well for large databases. Thus an important open problem is to design efficient SMC protocols. Next, we study a collection of techniques that are used to perform SQL^{--} queries over partition distributed data in SMC paradigm [69, 28].

This chapter describes the main techniques proposed in the literature for performing queries in SMC paradigm.

2.2 Background

The objective of this section is to look at some approaches relevant to the basic problems of SMC such as privacy-preserving query processing. To this purpose first we present the scenario of multi-party computation and then outline some solutions to support SQL^{--} over data horizontally partition among multiple data owners.

2.2.1 Secure Multi-party Computation

Generally speaking, SMC solves the problem of creating a privacy-preserving protocol for computing any function $f()$; that is, SMC techniques deliver, for a given function $f()$, a protocol that computes $f()$ over distributed inputs while revealing only the query result (and inferences that can be made from it). Many real world problems can be modeled in terms of SMC and can be solved applying the available SMC techniques. The SMC may be based on one of the following model :

- without third party: parties can jointly run their own SMC protocols without the help of third party;
- trusted third party: parties rely on trusted third party (TTP) for joint computation.

In theory, SMC problems can be solvable assuming the existence of a TTP, holding the secrets of the parties, computing the function and announcing

the result to the parties. However, this common strategy is risky in nowadays malicious environment, when disclosing the private input even to a TTP should be avoided, or when a TTP is not easy reachable. On the other side today there are many practical mechanisms that require TTP and it is natural to consider the possibility of implementing such a party using multi-party computation.

2.2.2 Description of SQL^{--} operators

In this section, we discuss the relational operations that need to be addressed under schemes proposed by research community to support query processing over encrypted data. We focus on the so-called SQL^{--} subset of the SQL query language including general database queries such as equality test, range and equi-join. To explain the implementation of each of these queries, we consider a data outsourcing mode where data owner O holds relations R that contains a set of searchable attributes[1] $A_1, \ldots, A_{l_1}$ and extra attributes[2].

Equality test. an equality test query selects items in a database that are equal to a given value. Let us suppose an authorized user asks for all $A_1.x$, where x is a plaintext value of attribute A_1. This type of query can be formulated as follows:

$$SELECT\ R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}\ FROM\ R\ WHERE\ A_1.x$$

or

$$\pi_{R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}}(\sigma_{A_1.x}(R))$$

Range. a range query selects items in a database that are equal to a given value fall in a specified range. Suppose an authorized user wants to perform a query with a range $\mathbf{r} = [x_1, x_2]$. This query can be formulated as follows:

$$SELECT\ R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}\ FROM\ R\ WHERE\ A_1.X_i\ s.t.\ x_1 \leq x_i \leq x_2$$

or

$$\pi_{R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}}(\sigma_C(R))\ s.t.\ C = \{A_1.x_i | x_1 \leq x_i \leq x_2\}$$

[1] A searchable attribute is an attribute that user wants to perform search on it. This attribute participates in the query condition.

[2] An extra attribute in a non-searchable attribute that is included in the query result.

Equi-join. an equi-join query combines tuples from two relations that have equivalent values for the specific attribute (join attributes). Suppose owner O holds another relation S with a set of searchable attributes $C_1, \ldots, C_{q_1}$ and extra attributes $D_1, \ldots, D_{q_2}$. An authorized user wants to join R and S on join attributes R_{B_1} and S_{D_1}: the resulting query can be formulated as follows:

$$SELECT\ R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}, R_{C_1,\ldots,C_{q_1},D_1,\ldots,D_{q_2}} FROM\ R\ , S\ WHERE\ R_{A_1.x} = S_{C_1.x}$$

or

$$\pi_{R_{A_1,\ldots,A_{l_1},B_1,\ldots,B_{l_2}}}(\sigma_{R_{A_1.x}=S_{C_1.x}}(R \bowtie S))$$

2.2.3 Querying over Secure Multi-party Computation

A number of techniques for SMC-based privacy preserving queries in distributed databases have been proposed along the years, delivering different performance levels in terms of security and scalability [40, 25, 41, 1, 16, 52, 23]. Different proposals assume different topology - for instance, a ring or a mesh - for the network interconnecting nodes that hold database partitions and different behavioral model - from honest-but-curious to malicious. Also, they put forward different SMC techniques - for instance, with or without support of a trusted computing unit - for preventing data leakage in query computation. Available proposals also differ in the expressive power of their SMC-supported query language: specific techniques have been put forward dealing with relational equality [45, 51, 43], range [68] and join queries [1, 52, 23]. In the following, we first review of studies on private set intersection protocol, and then describe the state of art on SQL^{--} query processing.

Set Intersection Problem

Consider a multi-party setting where each party has a set of items of size s. For the sake of simplicity, in the following we assume these items to be integer values. The set intersection problem is to compute the intersection between these sets without revealing any other information.

Freedman, Nissim and Pinkas [25] proposed a protocol based on the Paillier's cryptosystem [54]. In their protocol each party P_i, $i \in \{1, \ldots, m-1\}$, sends a polynomial F_i to P_m. The F_i polynomial has degree s and is rooted in P_i items. Then P_m, for each item y in her list, sends a $s \times (m-1)$ matrix built by evaluating in the point y of polynomials previously

received from other parties. Receiving parties decrypt and combine the evaluations to determine whether their items belong to intersection. The average computation cost for each party and the communication cost among all parties are $O(s(s+m)lgN)$ and $10s(m-1)^2lgN$, respectively.

Following Freedman's protocol, Kissner and Song [41] designed a technique for set intersection problem using polynomial representations with $O(cs^2lgN)$ computation complexity and $2cm(5s+2)lgN$ communication complexity, where c is a suitable constant and N is the Paillier cryptosystem RSA-modulus. Their protocol relies on randomly selected, irreducible polynomials zeroing at the data values. It supports not only intersection, but all set-theoretical operations. Each data owner encrypts its polynomials and broadcasts them to other data owners. Then, all data owners jointly perform decryption to compute the operation result. While Kissner-Song's polynomial technique looks effective and ingenious on paper, using it to support a (privacy-preserving) relational query system poses some hard scalability problems. Polynomials should be constructed to represent all relational tuples, which requires an amount of memory of the same size of the original database size. When the database is partitioned horizontally, not all nodes may have enough local resources to support the scheme.

Hazay and Lindell [32] proposed a different approach for privacy-preserving set-intersection where an oblivious pseudorandom function (PRF) is used instead of standard polynomial evaluation [50]. Hazay-Lindell's protocol is somewhat more efficient than Kissner-Song's polynomial approach, but it still requires heavyweight encryption/decryption operations that are impractical for distributed settings, where some nodes may have limited computational resources on-board. Also, it provides security against a smaller class of malicious adversaries[3].

Sang et al. [59] adopted a distinct through related approach with lower computation and communication costs with respect to [25, 41]. Their protocol needs $O(csmlgN)$ computation cost for each party and $2cm(4s+5)lgN$ communication cost among all parties. However, these costs are still too high for practical use.

A major step forward was taken by Agrawal et al. [1]. They introduce a two-party protocol for set intersection based on commutative encryption with linear complexity. The protocol has been extended to multi-party case [64] with lower complexity than [25, 41, 59], i.e., $O(smlgN)$ computation and $((m-1)s+(m-2)s+1+(m-1))lgN$ communication overhead.

[3]The authors show that their protocol can be modified so that malicious adversaries will be identified, but only with a probability that in turn depends on the overall protocol complexity.

Table 2.1: Time complexity comparison of solutions for set intersection problem (N is the Paillier cryptosystem RSA-modulus, c is a suitable constant , and m is the number of participants)

Method	Efficiency requirements	
	Computation cost	Communication cost
Freedman et al. [25]	$O(s(s+m)lgN)$	10s(m-1)2lgN
Kissner et al. [41]	$O(s^2lgN))$	$2m(5s+2)lgN$
Sang et al. [59]	$O(smlgN)+c$	$2m(4s+5)lgN$
Agrawal et al. [1]	$O(smlgN)$	$((m-1)s+(m-2)s+1+(m-1))lgN$

However, Vaidya's [64] extension only computes the size of the intersection, without identifying the intersection's elements.

In the reminder of this thesis we will show how the idea underlying Agrawal's protocol can be extended to tackle the **S**ecure **M**ulti-party **E**quality test **P**roblem (SMEP) problem, computing equality test queries in a privacy-preserving way.

Equality Test Problem

Even before multi-party SMC set operations were firmly in place, research started on techniques for querying partitioned databases using SMC. Indeed, the relationship between set operations and relational queries is rather straightforward: consider a simple case in a two-party setting where a receiver and a sender hold items a, d and b, c, respectively. Each party wants to know whether one or more values it holds are equal to values held by the other party without revealing its items. The same setting can be considered as a two-rows relation T where each row contains an item and partitioning has assigned two rows to each party. The SMC equality test problem is then equivalent to a (set-intersection) SQL query to the partitioned database.

A general way of solving this problem is using Yao's protocol [69], which represents the function as a combinational circuit to compare the corresponding bits through *and* operator with the complexity *O(lgN)* gates. However, this general solution is not efficient for multi-party case. Goldreich et al. [28] show that a multi-party version of Yao′s protocol needs a quadratic communication complexity in the number of parties.

Lindell et al. [45] propose another solution for two-party equality based on homomorphic encryption (e.g., Paillier [54]). For checking whether a is equal to b, the receiver sends a in encrypted form to the sender who uses homomorphic property to compute $(E_{p_k}(a) + E_{p_k}(-b)) \cdot r$, where r is a random value selected by the sender in order to prevent the receiver

from learning b. Assuming that the receiver and the sender have a private set of s elements, the computation and communication complexity of this technique amounts to $O(2s^2 lgN)$, where $2lgN$ is the length of each Paillier encryption and N is the RSA-modulus used for the encryption, which is the multiplication of two large prime numbers. The homomorphic encryption method is not straightforwardly extendable to the multi-party case because if multi-party equality test is executed as a number of two-party tests, each party will learn the intersection of its value with the remaining ones.

Naor et al. [51] presented a solution based on oblivious polynomial evaluation (OPE) protocol, where the sender who has a polynomial F and the receiver who has an input x want to jointly compute $F(x)$ such that the sender learns nothing about x and the receiver learns only $F(x)$ and nothing more. For comparing the items a and b through OPE protocol, the receiver and the sender should each generate a random linear polynomial $P(\cdot)$ and $Q(\cdot)$, respectively and obliviously compute the two values $P(a)+Q(a)$ (computed by the receiver) and $P(b) + Q(b)$ (computed by the sender). If the two obtained values are the same then $a = b$ otherwise they are different with high probability. For the case that the receiver and the server have a list of s inputs, the protocol requires each party to perform s oblivious evaluations of a polynomial of degree n with the communication cost of $O(n)$. For this reason, this protocol is considered too expensive to implement in the multi-party setting [35].

Li et al. [43] propose a protocol that involves a third party to compare the values held by two other parties. The protocol is based on homomorphic encryption scheme and similar to the solution proposed in [45]. Although this protocol is faster than OPE, it has two main drawbacks: 1) the TTP should be trusted by all parties; 2) when the number of parties increases, the solution does not scale because of the communication and computation bottleneck created at TTP.

Range Problem

A notable omission regards range queries. Much research have been developed for querying encrypted outsourced data and many techniques for performing equality and range queries over encrypted data have been developed [62, 34, 27, 14]. Encryption techniques used for database outsourcing usually do not preserve order and therefore query predicates with comparison operators can not be straightforwardly evaluated on encrypted values. For this reason, authors in [2, 9] proposed a solution based on the exploitation of order preserving encryption schemes for supporting range queries

on encrypted data, while others have put forward bucketization-based approaches [34, 30]. However, these solutions have been tailored to the data outsourcing paradigm. To the best of our knowledge, no applicable solutions for the execution of privacy-preserving range queries in multi-party collaborative scenarios exist in the literature.

In the remainder of this thesis, we will show how the idea underlying equality test problem can be extended to tackle this problem, computing range queries respecting data privacy.

Join Problem

The problem of join queries in MPC while preserving privacy has been studied in [1, 52, 23]. A general way for solving this problem is using Yao's protocol [69], which is not efficient for multi-party case [29].

Recently, several schemes have been proposed in the literature to perform equi-join query via set intersection protocol [40, 25, 1, 41]. Agrawal et al. [1] show a two-party protocol based on commutative encryption method to process join protocol. They consider two parties - a sender and a receiver- with tables T and $T^{'}$, respectively. The scheme ensures that the receiver will only learn information corresponding to the intersection of join attributes of tables T and $T^{'}$ but nothing else. While their protocol answers private equi-join query in a simple and effective way, it is costly in multi-party setting in terms of computation and communication time.

Following Agrawal's protocol, Emekci et al. [23] designed a new technique using Shamir's secret sharing to speed up query response time. According to their scheme, a sender and a receiver creates n shares for every element in their join attribute and distribute them to n third parties. Consequently, each third party calculates bitmap lists corresponding to the shares, which allows parties to determine elements of the intersection set. Then, a sender sends all tuples $t \in T^{'}$ where $t.A \in T \cap T^{'}$ without revealing any extra information except join result. Although this scheme is faster than Agrawal's protocol, it suffers from bottleneck created on third parties when the number of parties increases.

Narayan et al. [52] presented a solution for join query across multiple databases in which they introduce a system called *Djoin* that first converts join queries into set intersection queries where possible and then jointly executes them in a distributed fashion. While the proposed solution is much better than general purpose solution in MPC and suitable for offline analysis, not all queries can be translated in such a way.

The mentioned solutions have been tailored to the scenario of two-party

case where each party owns different table, say T_R for receiver and T_S for sender. In the remainder of this thesis, we will focus on the situation where each party has two shares associated to T_R and T_S in multi-party setting.

2.3 Chapter summary

The aim of SMC scenario is to enable players to carry out distributed computing tasks on their private information in a privacy way. In this chapter, we first defined the concept of SMC, and then we studied techniques concerning P^3Q over data horizontally partitioned among multiple data owners. We pointed out that set intersection problem can be considered as a fundamental operation for performing SQL^{--} query processing. As Table 2.1 shows that Agrawal's set intersection technique provides less computation and communication costs with comparison to [25, 41, 59]. Therefore, our goal in the next chapter is to extend the treatment of two-party Agrawal's set intersection protocol into multi-party setting, and develop it by exploiting bucketization technique, resulting less computation and communication costs.

Chapter 3

Querying by Secure Multi-party Computation

In this section, we describe three different protocols, for equality-test, range and join queries respectively for the multi-party scenario that, while preserving privacy, reduce the computation and communication cost by adopting bucketization technique.

3.1 Introduction

SMC is an interesting research area, though few real world applications have been described in recent years. The research community has developed a wide range of techniques based on homomorphic encryption for computing set-theoretical operations on multiple datasets [25, 41]. Other similar proposals [51] rely on oblivious polynomial evaluation. While all these techniques offer strong privacy, they do not scale well for large size data because of using heavyweight cryptographic operation and/or several rounds of interactions among parties. Therefore, finding a secure protocol for P^3Q is a major requirement for real word application. To this purpose, this thesis proposes an approach for the multi-party scenario to support SQL^{--} queries, relying on a commutative encryption scheme to avoid information from being revealed among data owners, and on a bucketization technique to reduce the time complexity. Moreover in order to realize the bucketization scheme, a trusted third party (TTP) is involved just at the beginning of the protocol. The TTP does not participate in the query processing, thus avoiding computation and communication bottleneck at TTP.

This chapter is organized as follows. Next section states some preliminaries and definitions behind our approach, including Agrawal's set intersec-

tion protocol in two-party setting. In Section 3.3, we present our protocol B-SMEQ adopting bucketization technique. Section 3.4 discusses the security of our protocol in the presence of malicious participants. In Section 3.5, the problem of privacy-preserving range queries is introduced, and Section 3.6 shows a SMC-based protocol for performing equi-join queries on big partition tables.

3.2 Preliminaries

3.2.1 Set Intersection Protocol

The set intersection protocol proposed by Agrawal et al. [1] can be summarized as follows:

Input. Two parties S(sender) and R(receiver) with set of values V_S, V_R and keys k_s, k_r, respectively.

Output. $V_S \cap V_R$

Step 1. Both S and R apply hash function h to their own values and encrypt the result with their secret keys, $f_{k_s}(h(V_S)$ and $f_{k_r}(h(V_R))$.

Step 2. Sites S and R exchange $f_{k_s}(h(V_S)$ and $f_{k_r}(h(V_R))$ after randomly reordering their values to prevent possible inference attacks.

Step 3. Site S encrypts each value of the set $f_{k_r}(h(V_R))$ with k_s to obtain $Z_R = f_{k_s}(f_{k_r}(h(V_R)))$ and sends back the pairs $(f_{k_r}(h(V_R)), Z_R)$ to R.

Step 4. Site R creates pairs (v,Z_R) by replacing $f_{k_r}(h(V_R))$ with the corresponding v where $v \in V_R$.

Step 5. Site R selects all v for which $Z_R \in f_{k_r}(f_{k_s}(h(V_S))$ for $V_S \cap V_R$.

3.2.2 System Model

We consider a Multi-party System $MS = \langle D, m, U, q, R \rangle$ for query computation involving five different entities as illustrated in Figure 3.1: a common database D composed of a set of tables $\{T, T', T'', \ldots\}$, which have been horizontally partitioned among a set of (mutually distrustful) m data owners O; a set of authorized users[1] U enabled to query the system; a query q and the

[1]Since authentication and access control are not the main focus in this paper, we assume access authorizations between data owners and users are appropriately managed.

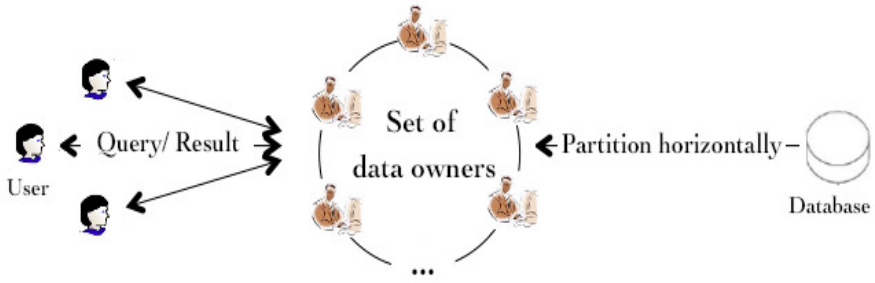


Figure 3.1: Basic Scenario

result R of the query computation across multiple tables/partitions. The data owners $O = \{O_1, ..., O_m\}$ hold data partitions $\{T_1, ..., T_m\}$, $\{T_1', ..., T_m'\}$ and so on. For the sake of simplicity, let us assume that each share T_i includes one searchable attribute $T_{i,A}$ with a set of values V_A and at least one extra attribute $T_{i,B}$ with a set of values V_B. Moreover, for each $v \in T_{i,A}$, $ext_i(v)$ denotes the values occurring in V_B where $T_{i,A} = v$.

Let us recall that in SMEP, the m data owners must jointly compute the result R to the equality test query q and return it to an authorized user $u \in U$, without revealing their private data to each other. Moreover, MS must satisfy the following properties:

Data privacy: the user learns only the result of the query,

Query privacy: the data owners do not learn the query,

Query anonymous result: the user does not know to whom the results belong.

In our model, participants are arranged in a ring topology and one data owner is designated as the master site (initiator). The communication among owners is realized via secure channels. We assume data owners are *honest-but-curious*, meaning that they follow the protocol as exactly specified (they are honest), but they try to learn extra information by analyzing the transcript of messages received during the execution.

3.2.3 Definitions

In this section, we use *hash function* and *commutative encryption* scheme for mapping the original values to achieve data privacy across multiple data owners.

One-way hash function.

A one-way hash function is a simple and very straightforward method for hiding data [16]. However, employing only hash function to data owners' values is not effective for preserving data privacy with *honest-but-curious* behavior. In fact, when the domain of values is small, the attacker can exhaustively go over all possible values and compute their hashes to learn the real values.

Commutative encryption scheme.

Informally, a commutative encryption is a pair of encryption functions f and g such that $f(g(v))$=$g(f(v))$. Thus by using the combination $f(g(v))$ to encrypt v, we can ensure that one data owner cannot compute the encryption of value without the help of other data owners. In the following we give a formal definition of commutative encryption scheme.

DEFINITION 1 (INDISTINGUISHABILITY). Let $\Omega_k \subseteq \{0,1\}^k$ be a finite domain of k-bit numbers and D_1 and D_2 be distributions over Ω_k. Let $A_k(x)$ be an algorithm that, given $x \in \Omega_k$, returns either true or false. We define distribution of D_1 of random variable $x \in \Omega_k$ to be *computationally indistinguishable* from distribution D_2 if, for any family of polynomial-step (w.r.t. k) algorithms $A_k(x)$, $P_r\left[A_k(x)|x \sim D_1\right] - P_r\left[A_k(x)|x \sim D_2\right] < \frac{1}{p(k)}$, for each polynomial $P(k)$ and sufficiently large k. Here $x \sim D$ denotes that x is distributed according to D, and $p_r[A_k(x)]$ is the probability that $A_k(x)$ returns true.

DEFINITION 2 (COMMUTATIVE ENCRYPTION). A commutative encryption F is a computable (in polynomial time) function $f = key\ F \times Dom\ F \rightarrow Dom\ F$, defined on finite computable domains that satisfies all properties listed below. We denote $f_e(x) \equiv f(e,x)$, and use $\in_r$ to mean "is chosen uniformly at random from"

1. Commutatively for all $e, e' \in key\ F$ we have $f_e \circ f_{e'} = f_{e'} \circ f_e$,
2. Each $f_e : Dom\ F \rightarrow Dom\ F$ is a bijection,
3. The inverse f_e^{-1} is also computable in polynomial time given e,
4. The distribution of $\langle x, f_e(x), y, f_e(y)\rangle$ is indistinguishable from the distribution of $\langle x, f_e(x), y, z\rangle$ where $x, y, z \in_r Dom\ F$ and $e \in_r key\ F$.

With respect to security concern, we apply the commutative encryption function to hash values instead of actual values to guarantee the indistinguishability between data distribution and uniform random values. We assume that hash function is ideal i.e., every hash function is evaluated for a new value, an independent random value is chosen.

Therefore, the input for the encryption function appears random and is able to satisfy the property 4.

3.3 A Multi-Party Protocol for Privacy-Preserving Equality test Queries

This section deals with equality test query among data of multiple data owners without revealing anyone's private data to others. In order to nicely scale with large size data, we show how communication and computation costs can be reduced via bucketization technique. In the following, we introduce a naive solution to the equality test problem in MPC followed by the description of B-SMEQ (**B**ucketized **S**ecure **M**ulti-party protocol for **E**quality test **Q**ueries). The protocol correctness, the security issues and the computation and communication cost analysis end the section.

3.3.1 A naive approach: SMEQ

In this section, we extend the two-party set intersection protocol given by Agrawal et al. (see 3.2.1) for solving the SMEP problem by a naive approach, which does not ensure scalability. the protocol, named **S**ecure **M**ulti party protocol for **E**quality test **Qu**eries (SMEQ), extends the treatment of set intersection protocol from two-party to multi-party setting, where the role of sender is played by each of m data owners. SMEQ is simple to describe and implement but suffers from high communication and computation.

Input. A multi-party system with the data owner O_1 as initiator[2].

Output. The result of user query q to partitions $\{T_1, \ldots, T_m\}$ where $[(T_{1,A} = v) \vee \ldots \vee (T_{m,A} = v)]$.

Step 1. Both user (receiver side, R) and data owners O (sender side, S) apply hash function h to the value v and to the set of their values of attribute A, respectively. Let $v' = h(v)$ be the result of hashing from the receiver side and let $T'_{i,A'} = h(V_{i,A})$, for each $i \in \{1, \ldots, m\}$, be the hashing of the set values $T_{i,A}$. Sites S and R randomly choose a secret key, k_r for R and $\langle k_i, k'_i \rangle$ for data owner O_i.

Step 2. R spans the encrypted hash value $y_R = f_{k_r}(v')$ to all data owners at site S.

Step 3. At site S, each data owner O_i, $1 \leq i \leq m$, does the following:

3.1 Computes $f_{k_i}\left(V_{i,A'}\right) = Y_i = \{y_i = f_{k_i}(x) | x \in V_{i,A'}\}$,

[2] Any data owner(for instance, the one holding the largest data partition) can be chosen as initiator.

3.2 Generates a set of new keys, one for each value of attribute B, as K_i^B=$\{k_{ix} = f_{k_i'}(x)|x \in V_{i,A'}\}$,

3.3 Encrypts each value x in $T_{i,B}$ with the corresponding key k_{ix} to obtain $Y_i^B = \{E_{k_{ix}}(u)|u \in V_{i,B}\}$, where E is an encryption function which can be efficiently inverted given key k_{ix},

3.4 Computes $I_i = f_{k_i'}(y_R)$ for the purpose of decrypting the values of attribute B at site R,

3.5 O_i randomly reorders the tuples Y_i and Y_i^B and sends them along with I_i to the next owner $O_{(i \bmod m)+1}$,

3.6 Data owner $O_{(i \bmod m)+1}$ encrypts only Y_i with the key $k_{(i \bmod m)+1}$ and sends the triple $\langle f_{k_{(i \bmod m)+1}}(Y_i), Y_i^B, I_i\rangle$ after reordering to the next participant in the ring. This process is repeated until Y_i is encrypted by all keys of m data owners, obtaining $Z_i = f_{k_1}(f_{k_2}(...(f_{k_m}(Y_i))))$, up to a permutation of the encryption keys[3].

Step 4. Each data owner O_i sends $\langle Z_i, Y_i^B, I_i\rangle$ to O_1.

Step 5. O_1 receives all tuples from Step 4 in order to initiate a two-party Agrawal's set intersection protocol between the user as a receiver site and the initiator as a sender site.

Step 6. O_1 passes y_R through the ring in order to have it encrypted by all keys $k_1, \ldots, k_m$ for obtaining $y_R' = f_{k_1}(\ldots(f_{k_m}(f_{k_r}(v'))))$, and then sends back y_R' together with $\langle Z_i, Y_i^B, I_i\rangle$ to the user, for all $i \in \{1, \ldots, m\}$.

Step 7. First, R decrypts y_R' with her decryption key to obtain $y_R'' = f_{k_1}(\ldots(f_{k_m}(v')))$, and then for each i, $1 \leq i \leq m$: 1) Finds tuples in Z_i whose entry related to the searchable attribute is equal to y_R''; 2) Considers the entry corresponding to attribute B of those tuples; 4) Decrypts I_i with k_r, obtaining $f_{k_i'}(v')$; 5) Uses $f_{k_i'}(v')$ to decrypt the corresponding entry in Y_i^B.

Although this protocol is simple and effective, it has high cost in terms of communication and computation over large data sets. Moreover, it suffers from high query computation workload at user side. In the next section we show how to overcome these drawbacks.

[3]The keys $k_1, k_2, \ldots, k_m$. represent a commutative set of keys.

3.3.2 From SMEQ to B-SMEQ

B-SMEQ makes use of a TTP which is not involved in the query processing, but only in the realization of a bucketization scheme. Before the protocol starts, the TTP builds an interchange matrix W and sends the row vectors of W to the data owners. The owner that receives W_1 is called the initiator. Then, data owners are arranged into a ring; each holds a permutation[4] of the bucket order $\{1, \ldots, s\}$ and a vector from matrix W. In the first part of the protocol, each owner sends her buckets in encrypted form to the next participant. When a user wants to submit an equality query, she gets from the TTP the bucket number of the query value according to the TTP's own permutation, $\bar{\Pi}$. This number is sent to the initiator, who uses it to identify the bucket related to the user value from her predecessor in the ring she should overencrypt with her key. The procedure is repeated for each node of the ring. In the second phase of the protocol, selected buckets (i.e. the ones corresponding to the user value) are propagated along the ring until they have been encrypted by all keys. Once all data owners hold the encrypted buckets, a two-party Agrawal's protocol is executed between the initiator and the user. In the next sections we will provide a detailed description of the algorithm.

Defining Buckets. We divide the range of values, in the domain of each searchable attribute A, into buckets so that each bucket is assigned a unique label (ID). This label is then stored alongside the encrypted version of the searchable attribute. We adopt equi-width strategy for selecting buckets, which divides the range of possible values of searchable domain into s buckets of the same size l, i.e. $l = \frac{A_{max} - A_{min}}{s}$, where A_{max} and A_{min} are the maximum and minimum values in the domain of A, respectively. Thus, we define bucket boundaries of the searchable attribute as $BU = \{B_1 : [A_{min},\ l],\ \ldots,\ B_s : (l(s-1),\ A_{max}]\}$. BU is called *public bucketization scheme* and it is accessible to all data owners and authorized users as well.

Example.

If the observed values of a given searchable attribute (e.g. age) are between 0-80, and we suppose $s = 4$, the protocol creates 4 buckets of width $(80 - 0)/4 = 20$ as $B_1 : [0, 20],\ B_2 : (20, 40],\ B_3 : (40, 60],\ B_4 : (60, 80]$.

[4]We assume that partitions are abundant meaning that the number of permutations is much greater than the number of participants.

The bucketization improves the efficiency of SMEQ by considering only records relative to the buckets containing the value searched by the user. In this way, the data owners work only on a subset of their data at each round, leading to a more scalable protocol.

The protocol. B-SMEQ has two phases: computation of matrix W and query protocol. First, in order to preserve data privacy, each data owner O_i separately computes a local permutation $(B_{\pi_{i_1}}, \ldots B_{\pi_{i_s}})$ of the public bucketization scheme for the searchable attribute. Each data owner i chooses a local permutation $\Pi_i = (\pi_{i_1}, \ldots, \pi_{i_s})$ of bucket indices $(1, 2, \ldots, s)$. Moreover, the TTP chooses her own permutation $\bar{\Pi}$ and sends it to the user posing the query.
Let's now assume that the user query value v falls into bucket B_i of public bucketization. Then, the user sends to a pre-set data owner (henceforth, the *initiator*) a query for the bucket $B_{\bar{\pi}_i}$. In this way, when data circulate along the ring, data owners will not know which bucket ID the user is looking for. We will now present our matrix-based mechanism, which allows each data owner to correctly select the local bucket corresponding to $B_{\bar{\pi}_i}$.

Phase1. Computation of matrix W

Step 1. Each data owner O_i sends her permutation Π_i to the TTP.

Step 2. The TTP builds the matrix Π containing the received permutation vectors and generates a $m \times s$ interchange matrix W, where the matrix elements are defined by Eq. (3.1).

$$\Pi = \begin{pmatrix} \Pi_1 \\ \Pi_2 \\ \vdots \\ \Pi_m \end{pmatrix} = \begin{pmatrix} \pi_{11} & \pi_{12} & \ldots & \pi_{1s} \\ \pi_{21} & \pi_{22} & \ldots & \pi_{2s} \\ \vdots & & & \vdots \\ \pi_{m1} & & & \pi_{ms} \end{pmatrix}$$

$$W = \begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_m \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1s} \\ w_{21} & w_{22} & \ldots & w_{2s} \\ \vdots & & & \vdots \\ w_{m1} & & & w_{ms} \end{pmatrix}$$

In the following equation, we denote by $\bar{\Pi}^{-1}(l)$ the position in vector $\bar{\Pi}$ that contains value l.

$$
\begin{aligned}
& w_{ij} = \pi_{i\delta_i} \quad \forall i \in \{1,2,\ldots,m\},\ j \in \{1,2,\ldots,s\}, \text{ where} \\
& \delta_1 = \bar{\Pi}^{-1}(j) \\
& \delta_2 = \bar{\Pi}^{-1}(k_1) \quad \text{with } k_1 \text{ s.t. } w_{1k_1} = j \\
& \vdots \\
& \delta_i = \bar{\Pi}^{-1}(k_{i-1}) \\
& \qquad \text{with } k_{i-1} \text{ s.t. } w_{1k_{i-1}} = k_{i-2}, \\
& \qquad \text{with } k_{i-2} \text{ s.t. } w_{2k_{i-2}} = k_{i-3}, \\
& \qquad \vdots \\
& \qquad \text{with } k_1 \text{ s.t. } w_{i-1k_1} = j
\end{aligned}
\tag{3.1}
$$

The rationale behind Eq. (3.1) is generating the entries of matrix W by identifying a corresponding entry of matrix Π. This entry, for the first row ($i = 1$) is obtained by looking for the position of index j in the TTP permutation. For the other rows, the entry is obtained by looking for the position of index j in the previous rows of matrix W itself.

For instance to compute δ_2 for $w_{21} = \pi_{2\delta_2}$, TTP follows this procedure: a) Find $j = 1$ in the previous line of matrix W b) Take the position of 1 and read the value in this position of the TTP permutation.

Step 3. The TTP sends the row vectors of W to the data owners and her permutation $\bar{\Pi}$ to the user.

Phase 2. Query Protocol

Steps 1-3 of B-SMEQ correspond to Steps 1-3 (excluding Step 3.5) of SMEQ

Step 4. The user sends $B_{\bar{\pi}_k}$ to the initiator, where k is the number of the bucket where the query value v falls.

Step 5. Let us recall that, at this step, each data owner O_i holds data Y_{i-1} (which corresponds to T_{i-1} encrypted with the key k_{i-1}) of O_{i-1}. With this arrangement, O_2 is the algorithm's initiator[5]. O_2 sets $h_2 = w_{1\bar{\pi}_k}$, selects $Y_1(h_2)$ i.e. the bucket in Y_1 whose ID is h_2, and encrypts it with her own key. Then O_2 sends h_2 to the next owner. When data owner O_i receives h_{i-1} from O_{i-1}, she sets $h_i = w_{(i-1)h_{i-1}}$, selects the corresponding bucket $Y_{i-1}(h_i)$ and sends the position h_i to

[5]This is not a limitation, because any node can be selected as initiator by changing the order of permutations at the time of W generation by TTP.

the next owner. This step iterates until all data owners have selected their bucket.

Step 6. The data owners apply the procedure described in Step 3.5 of SMEQ just on the selected buckets obtained at Step 5.

Steps 7-10 of B-SMEQ correspond to the Steps 4-7 of SMEQ

B-SMEQ: A worked-out example Consider 4 data owners on a ring, where each has a searchable attribute A with range $[0, \ldots, 100]$ and s=5 with the same size.

Phase 1. Computation of matrix W According to the Phase 1 described in Section 3.3.2, each data owner O_i takes a local permutation Π_i of bucket order $\{1, \ldots, s\}$ on searchable attribute regarding the public bucketization $BU = \{B_1 : [0, 20], B_2 : (20, 40], B_3 : (40, 60], B_4 : (60, 80], B_5 : (80, 100]\}$ and sends Π_i to the TTP (shown in Figure 3.2(a)).

$$\Pi_1 = (3, 2, 1, 4, 5), \Pi_2 = (5, 2, 1, 3, 4), \Pi_3 = (3, 2, 1, 5, 4), \Pi_4 = (1, 3, 2, 4, 5)$$

First, the TTP builds a 4×5 matrix Π and chooses her own permutation $\bar{\Pi}$ randomly, and then TTP generates an interchange matrix W by following the Eq. (3.1):

$$\Pi = \begin{pmatrix} 3 & 2 & 1 & 4 & 5 \\ 5 & 2 & 1 & 3 & 4 \\ 3 & 2 & 1 & 5 & 4 \\ 1 & 3 & 2 & 4 & 5 \end{pmatrix} = \begin{pmatrix} \Pi_1 \\ \Pi_2 \\ \Pi_3 \\ \Pi_4 \end{pmatrix}$$

$$\bar{\Pi} = (5, 3, 1, 4, 2)$$

$$W = \begin{pmatrix} 1 & 5 & 2 & 4 & 3 \\ 1 & 2 & 5 & 3 & 4 \\ 1 & 2 & 5 & 4 & 3 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix} = \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{pmatrix}$$

For instance, we compute w_{34} where $w_{34} = \pi_{3\delta_3}$ and the integer δ_3 is calculated as follows:

$\delta_3 = \bar{\Pi}^{-1}(k_2)$, with k_2 s.t. $w_{1k_2} = k_1$ and k_1 s.t. $w_{2k_1} = 4$, so $k_1 = 5$

$\delta_3 = \bar{\Pi}^{-1}(k_2)$, with k_2 s.t. $w_{1k_2} = 5$, so $k_2 = 2$

By $\delta_3 = \bar{\Pi}^{-1}(2) = 5$, we get the solution $w_{34} = \pi_{35} = 4$.

Once W is computed, TTP sends the row vectors of W to the data owners and her permutation to the user. Note that W_i corresponds to Π_i. However, since in the first part of *query protocol* phase each data owner sends her buckets in encrypted form to the next participant, TTP sends W_i to O_{i+1} for searching user query. In our example O_2 is an initiator who receives W_1 (see Figure 3.2(b)).

Phase 2. Query Protocol According to the first part of query protocol, each data owner O_i holds her data along with her permutation, encrypted buckets of her upstream and vector row W_{i-1}. Suppose now that user wants to find value 82 that falls in B_5 of public bucketization. Then she sends to O_2 a query for the bucket $B_{\overline{\pi}_5} = 2$. The initiator O_2 follows the procedure: (a) Takes the value of position of 2 in W_1 and gets 5, (b) Overencrypts data corresponding to bucket 5 of her preceding with her key O_1, and (c) Sends the value 5 to the next participant. This procedure is repeated for each node in the ring (shown in Figure 3.2(c)). Once every data owner holds the buckets corresponding to the user query, a multi-party Agrawal protocol can be executed.

3.3.3 Correctness and Privacy Issues

In order to show that B-SMEQ computes query results correctly, we need to prove that at Step 5 data owners always select the position corresponding to the bucket ID related to the user value.

General Scheme

- User chooses bucket a and sends $\overline{\pi}_a$ to O_2 (initiator);
- O_2 computes $h_2 = w_{1\overline{\pi}_a}$, selects $T_{1(h_2)}$ and sends h_2 to the O_3;
- O_i receives h_{i-1}, computes $h_i = w_{(i-1)h_{i-1}}$, selects $T_{i-1}(h_i)$ and sends h_i to the next owner in the ring.

Proof of Correctness.

- O_2, who has the data Y_1 of O_1 receives $\overline{\pi}_a$, computes $h_2 = w_{1\pi_a}$, selects $Y_1(h_2)$ and sends h_2 to the O_3. Observe that by definition $h_2 = w_{1\pi_a} = \pi_{1\delta_1}$, where $\delta_1 = \Pi^{-1}(\overline{\pi}_a)$. Hence, $h_2 = \pi_{1a}$, which means O_2 chooses the correct bucket.

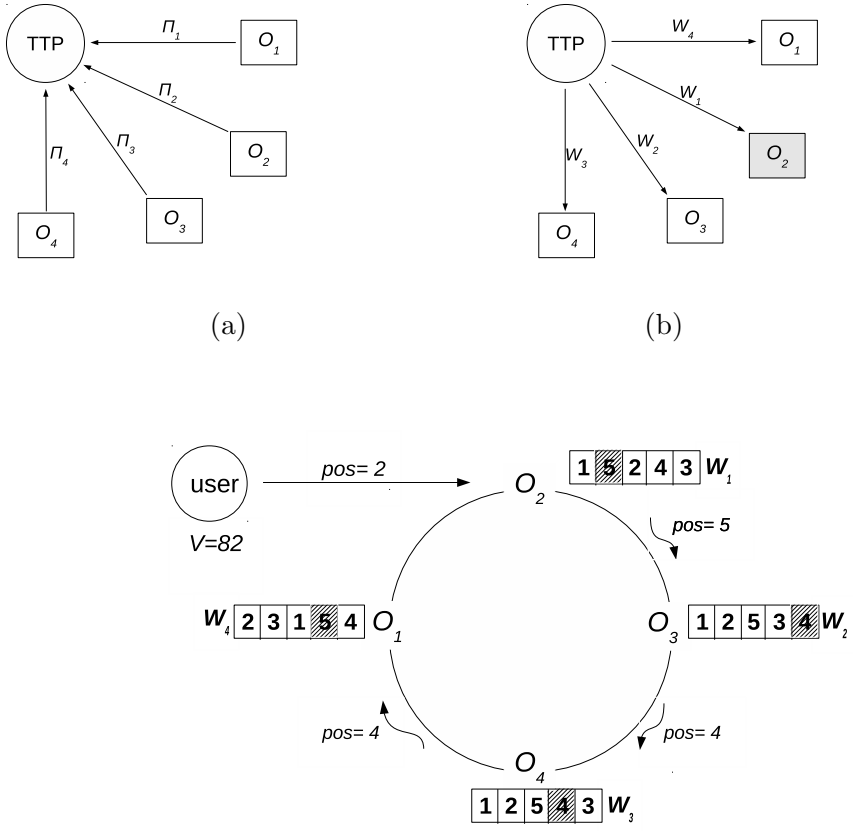


Figure 3.2: (a) Each data owner sends her permutation to TTP, (b) The TTP sends row vectors of matrix W to data owners and selects O_2 as an initiator and (c) Data owners select the correct buckets corresponding to the user query (Phase 2- Steps 4, 5)

- O_i, who has data Y_{i-1} of O_{i-1}, receives h_{i-1} from party $i-1$, computes $h_i = w_{(i-1)h_{i-1}}$, selects $Y_{i-1}(h_i)$, and sends h_i to next owner. Observe that by definition, we have $h_i = w_{(i-1)h_{i-1}} = \pi_{(i-1)\delta_{i-1}}$, where

$$
\begin{aligned}
&\delta_{i-1} = \Pi^{-1}(k_{i-2}) \\
&k_{i-2} \text{ s.t. } w_{1k_{i-2}} = k_{i-3} \\
&k_{i-3} \text{ s.t. } w_{2k_{i-3}} = k_{i-4} \\
&\vdots \\
&k_2 \text{ s.t. } w_{(i-3)k_2} = k_1 \\
&k_1 \text{ s.t. } w_{(i-2)k_1} = h_{i-1}
\end{aligned}
\tag{3.2}
$$

Since by definition $h_{i-1} = w_{(i-2)h_{i-2}}$, from (3.2) we have $w_{(i-2)k_1} = w_{(i-2)h_{i-2}}$, hence $k_1 = h_{i-2}$. By definition, $h_{i-2} = w_{(i-3)h_{i-3}}$ and from (3.2) $w_{(i-3)k_2} = w_{(i-3)h_{i-3}}$. This implies $k_2 = h_{i-3}$. By iterating this procedure, we obtain $k_{i-3} = h_2$, that impliese quation (3.2) $w_{1k_{i-2}} = h_2 = w_{1\pi_a}$. This means $k_{i-2} = \overline{\pi}_a$ and we have $\delta_{i-1} = a$ that prove the correctness for data owner O_i. □

Privacy issues. For the moment, we just consider processing a single query. We focus on the privacy of Steps 5 and 6 where the records of each data owner are selected according to the user bucket ID and circulated in order to be encrypted by all keys. At each round of the query protocol, every data owner receives a new set from her predecessor via the ring. Since each value of the received set has been hashed and encrypted based on *commutative encryption* function, the distribution of encrypted hash values are indistinguishable from the uniform random distribution Moreover, in each round of Step 6, data owner i receives a data set encrypted with different keys, so that party i can not infer the relationship between received data sets. In addition to the size of the whole upstream data set (revealed in the set intersection protocol), in Step 5 each party can learn the size (number of tuples) of the upstream selected bucket. Nevertheless, when a data owner O_i overencrypts the encrypted tuples received from her predecessor, O_i does not know which bucket ID corresponds to the received tuples. The size of the selected position does not reveal any further information to the data owner. Our protocol is thereby secure against semi-honest parties as long as no two data owners collude.

3.3.4 Time complexity analysis

Theoretical Cost Analysis

Here, we analyze the computation and communication costs of the protocols proposed in Section 3.3.1 and 3.3.2.

Computation Cost. The main computation cost during the execution of both protocols belongs to hashing and encrypting the set of values V_A corresponding to n distributed records among m data owner[6].
Let C_h be the cost of evaluating the hash function h, C_f be the cost of encryption/decryption by function f, and C_E be the cost of encryption/decryption by function E. For the sake of simplicity in the calculations below, we do not consider the cost of reordering encrypted values and assume a unitary cost when applying C_h, C_f and C_E to a single record.

SMEQ. We will now follow SMEQ step-by-step (Section 3.3.1) in order to quantify the complexity of each step.

Step 1. User and data owners hash their values V_A once, with the cost: $C_h(|T_{1,A}|) + \ldots + C_h(|T_{m,A}|) + C_h(v) = n + 1$[7].

Step 2. $C_f(v^{'}) = 1$ for encrypting user hash value $v^{'}$ to get $f_{k_r}(v^{'}) = y_R$

Step 3. For a single data owner O_i, we have:

(3.1) $C_f(|T^{'}_{i,A'}|) = \frac{n}{m}$, for encrypting a set of hash values with key k_i

(3.2) $C_f(|T^{'}_{i,A'}|) = \frac{n}{m}$, for creating a set of new keys for attribute B

(3.3) $C_E(|T_{i,B}|) = \frac{n}{m}$, for encrypting the set of values V_B with keys generated in Step 3.2

(3.4) $C_E(y_R) = 1$, for overencrypting the user encrypted value y_R with $k_i^{'}$. This value will be used as a decryption key for the query result.

(3.5) $C_f(\sum_{i=1}^{m-1} |T^{'}_{i,A'}|) = (m-1)\frac{n}{m}$, for encrypting values $V_A^{'}$ of O_i with $m-1$ keys of other data owners in the ring.

The cost of Step 3 for O_i is $C_f(n + \frac{n}{m}) + C_E(\frac{n}{m} + 1)$.

Steps 4, 5. No computation cost

Step 6. The user encrypted value is overencrypted with all keys k_i of m data owners through the ring. Total cost is $C_f(f_{k_1} \ldots (f_{k_m}(y_R))) = C_f(y_R^{'}) = m$

[6]We compute the total computation cost of the two mentioned protocols with the assumption that each data owner has $\frac{n}{m}$ expected records (uniform distribution).

[7]Note that $C_h(v)$ refers to the hashing cost of user value v to obtain $v^{'} = h(v)$.

Table 3.1: Computation and Communication costs of SMEQ and B-SMEQ with m=10 data owners, s=5 buckets and t equal to 10% of the number of records

Number of records	C_{SMEQ}	C_{B-SMEQ}	C'_{SMEQ}	C'_{B-SMEQ}
50000	$660.022 \cdot 10^3$	$300.022 \cdot 10^3$	$5450.645 \cdot 10^3$	$991.145 \cdot 10^3$
100000	$1320.022 \cdot 10^3$	$600.022 \cdot 10^3$	$10900.645 \cdot 10^3$	$1981.145 \cdot 10^3$
200000	$2640.022 \cdot 10^3$	$1200.022 \cdot 10^3$	$26160.774 \cdot 10^3$	$4753.375 \cdot 10^3$
300000	$3960.022 \cdot 10^3$	$1800.022 \cdot 10^3$	$39240.774 \cdot 10^3$	$7129.374 \cdot 10^3$
500000	$6600.022 \cdot 10^3$	$3000.022 \cdot 10^3$	$65400.774 \cdot 10^3$	$11881.374 \cdot 10^3$

Step 7. Let t be the number of tuples for which user finds an equality match. The cost of this step including decrypting y'_R and I_i is $tC_f(1) + tC_e(1) = 2t$

The total computation cost of SMEQ is given by:

$$\begin{aligned} C_{SMEQ} &= (n+1) + 1 + (nm+n) + (n+m) + m + 2t \\ &= nm + 3n + 2m + 2 + 2t \in O(mn) \end{aligned} \tag{3.3}$$

B-SMEQ. The major difference of the two protocols in terms of computation cost lies in Steps 3.5 and 7 of the SMEQ and B-SMEQ , respectively. In SMEQ the whole set of data of each data owner must pass through the ring in order to be encrypted with all keys, whereas in B-SMEQ only tuples that correspond to the bucket ID of user must circulate. For this reason, here we report just the complexity of Step 7 of B-SMEQ which corresponds to Step 3.5 of SMEQ as follows:

Step 7. For a single data owner O_i, we have $C_f(\sum_{i=1}^{m-1} \frac{|T'_{i,A'}|}{s}) = (m-1)\frac{n}{ms}$ for encrypting values V'_A of O_i with $m-1$ keys of data owners where s is the number of buckets.

Since the cost of other steps in B-SMEQ are equal to the SMEQ the computation cost of B-SMEQ is:

$$\begin{aligned} C_{B-SMEQ} &= (n+1) + 1 + (\tfrac{(m-1)n}{s} + 2n + m) + n + m + 2t \\ &= \tfrac{(m-1)n}{s} + 4n + 2m + 2 + 2t \in O(mn) \end{aligned} \tag{3.4}$$

Communication Cost. Communication cost can be computed as the total number of bits transmitted during the protocol execution. We compute

the total communication cost of our protocols regard the assumption that each data owner has $\frac{n}{m}$ expected records (uniform distribution). We consider k the length of each encrypted codeword of the domain of encryption function f and $k^{'}$ the length of the encryption function E on other attribute.

SMEQ. The communication cost of SMEQ can be computed by looking each step as was due to computational complexity.

Step 1. No communication cost

Step 2. The user broadcasts the encrypted hash value, y_R, to m data owners, giving $m \cdot k$

Step 3. For single data owner O_i, we have the following cost:

(3.1), (3.2), and (3.3). No communication cost

(3.4) $(|T^{'}_{i,A'}|+1)\cdot k+(|T_{i,B}|)\cdot k^{'}=(\frac{n}{m}+1)\cdot k+(\frac{n}{m})\cdot k^{'}$, for sending tuples $\langle Y_i, Y_i^B, I_i\rangle$ to the O_{i+1}

(3.5) Since this step needs to pass $m-2$ times the data of O_i through the ring in order to be encrypted by all $m-1$ keys, the cost is equal to $\sum_{i=1}^{m-2}(|T^{'}_{i,A'}|\cdot k+|T_{i,B}|\cdot k^{'}+1)$. Therefore, the cost of Step 3 for O_i is $\sum_{i=1}^{m-1}(|T^{'}_{i,A'}|\cdot k+|T_{i,B}|\cdot k^{'}+1)=(n-\frac{n}{m}+m-1)\cdot k+(n-\frac{n}{m})\cdot k^{'}$, and accordingly for m data owners we have $(nm-n+m^2-m)\cdot k+(nm-n)\cdot k^{'}$

Step 4. The communication cost of this step for sending the data of $m-1$ data owners to the O_1(initiator) is $(|T^{'}_{2,A'}|+\cdots+|T^{'}_{m,A'}|)\cdot k+(|T_{2,B}|+\cdots+|T_{m,B}|)\cdot\ k^{'}+(m-1)\cdot k=(n-\frac{n}{m}+m-1)\cdot k+(n-\frac{n}{m})\cdot k^{'}$

Step 5. No communication cost

Step 6. $m\cdot k$ is the cost of passing the user value through m data owners to be encrypted by all m keys and for sending all n records from O_1 to the user the cost is as follows:
$(|T^{'}_{1,A'}|+\cdots+|T^{'}_{m,A'}|)\cdot k+(|T_{1,B}|+\cdots+|T_{m,B}|)\cdot k^{'}+m\cdot k=(n+m)\cdot k+n\cdot k^{'}$

Step 7. No communication cost

Overall we have

$$\begin{aligned} C^{'}_{SMEQ} &= (m^2+nm-\tfrac{n}{m}+3m+n-1)\cdot k \\ &\quad +(nm-\tfrac{n}{m}+n)\cdot k^{'} \end{aligned} \tag{3.5}$$

So, the communication cost is asymptotically observed by $O(mn)$.

B-SMEQ. The difference between our two protocols with regard to the communication cost lies mostly in Phase 1- Steps 1, 3 and Phase 2- Steps 4, 5, 6,8 of B-SMEQ, that we describe below:

Step 1. $s{\cdot}k$, for sending $\Pi_i = (\pi_{i1}, \ldots, \pi_{is})$ of bucket indices $(1, 2, \ldots, s)$ from O_i to TTP, and accordingly for m data owners is $ms \cdot k$

Step 3. $ms \cdot k$, for sending row vectors of W to m data owners

Step 4. k, for sending bucket $B_{\bar{\pi}_k}$ from user to the owner O_2 (initiator), where $B_{\bar{\pi}_k}$ contains the position of bucket ID of user query

Step 5. $(m-1) \cdot k$ for sending the position h from O_2 to O_m to select the records in each data owner corresponding to the user bucket ID

Step 6. The cost of this step is calculated similar to the Step 3.5 of SMEQ except that here only a portion of data of the owner O_i, which is related to user bucket ID is sent through the ring. Thus, for a single owner O_i we have $\sum_{i=1}^{m-2}(\frac{|T^{'}_{i,A'}|}{s} \cdot k + \frac{|T_{i,B}|}{s} \cdot k^{'} + 1) = (m-2)(\frac{n}{ms} + 1) \cdot k + (m-2)\frac{n}{ms} \cdot k^{'}$, and accordingly for m data owners the cost is: $(\frac{n}{s} + m)(m-2) \cdot k + \frac{n}{s}(m-2) \cdot k^{'}$.

Step 8. This step contains the Steps 4-7 of SMEQ. In Step 4, Each data owner O_i sends $\langle Z_i, Y_i^B, I_i \rangle$ to the owner O_2. So, the communication cost for sending the data of $m-1$ data owners to O_2 is $(\frac{|T^{'}_{1,A'}|}{s} + \frac{|T^{'}_{3,A'}|}{s} + \ldots + \frac{|T^{'}_{m,A'}|}{s}) \cdot k + (\frac{|T_{1,B}|}{s} + \frac{|T_{3,B}|}{s} + \ldots + \frac{|T_{m,B}|}{s}) \cdot k^{'} + (m-1) \cdot k = (\frac{1}{s}(n - \frac{n}{m}) + (m-1)) \cdot k + \frac{1}{s}(n - \frac{n}{m}) \cdot k^{'}$, and in Step 6, for sending all records from O_2 to the user, we need the communication cost equal to $(\frac{|T^{'}_{1,A'}|}{s} + \ldots + \frac{|T^{'}_{m,A'}|}{s}) \cdot k + (\frac{|T_{1,B}|}{s} + \ldots + \frac{|T_{m,B}|}{s}) \cdot k^{'} + (m) \cdot k = (\frac{n}{s} + m) \cdot k + (\frac{n}{s}) \cdot k^{'}$

The total communication cost of B-SMEQ is:

$$\begin{aligned} C^{'}_{B-SMEQ} &= (m^2 + 3m + \tfrac{1}{s}(mn - \tfrac{n}{m}) + 2ms + \tfrac{n}{m} - 1) \cdot k \\ &\quad + (\tfrac{n}{m} + \tfrac{1}{s}(nm - n - \tfrac{n}{m})) \cdot k^{'} \end{aligned} \tag{3.6}$$

Again, the overall communication cost is $O(mn)$. Table 3.1 presents the communication costs $C^{'}_{SMEQ}$ and $C^{'}_{B-SMEQ}$ of SMEQ and B-SMEQ. We assumed k and $k^{'}$ to be equal to the smallest integer greater than $\lg(n)$. As Table 3.1 shows, the B-SMEQ protocol has lower communication cost; also, in this range, the difference between the two protocol increases with the number of records.

Practical Cost Analysis

Here we verify the scalability of our protocol via some experimental tests. We report the results in terms of communication time using Castalia on a Linux machine with dual Intel CPU running at 2.26 GHz. and 2GB Ram. It should be noted that Castalia[8] is a simulator for Wireless Sensor Networks (WSN), Body Area Network (BAN) and generally networks of low-power embedded devices. In particular, for our simulation, we deploy 4 and 5 nodes, respectively for SMEQ and B-SMEQ. For both protocols, we create a ring of three nodes with numbers from 0 to 2 for the three data owners and one node (number 3) for the querier[9].

To encrypt the set of searchable attribute of each data owner's table, we implement a simple *commutative encryption* protocol based on *exponentiation modulo p*. First, all data owners agree on a common large prime p; second, each data owner randomly chooses number e as an encryption key from the group Z_p, so that the greatest common divisor of e and $(p-1)$ is 1; third, each data owner calculates the decryption key d, which is the inverse of e modulo $(p-1)$ by using extended Euclidean algorithm. Since the most communication time of the two protocols is devoted to exchanging data of each data owner along the ring in order to be encrypted by all keys, we only focus on Steps 3.4 and 3.5 of SMEQ, and Steps 5 and 6 of B-SMEQ. We run two different experiments: in the first one, we compare the two protocols; in the second one, we evaluate the effect the number of buckets has on the communication time for B-SMEQ.

In the first experiment, we set five different triples $(\theta 1, \ldots, \theta 5)$ of number of records as $\theta 1 = \{5, 6, 7\}$, $\theta 2 = \{50, 60, 70\}$, $\theta 3 = \{500, 600, 700\}$, $\theta 4 = \{5000, 6000, 7000\}$ and $\theta 5 = \{50000, 60000, 70000\}$, where the position j in each triple is the number of records for data owner j, $j \in \{1, 2, 3\}$. Note that for B-SMEQ we divide the searchable attribute domain $[1 \ldots 100]$ into s=5 number of buckets of the same size $l = 20$. Figure 3.3(a) shows the result of our simulation, the solid line displays the result from SMEQ, whereas the dotted line displays the result from the B-SMEQ. The difference in communication time between the SMEQ and B-SMEQ increases fairly slowly when the number of records of data owners is relatively small, but it grows much faster as the number of records increases. The results come from the fact that for each query, in SMEQ all records of data owners must pass through the ring, while in B-SMEQ, from Step 6, only records

[8] http://www.omnetpp.org/component/content/article/8-news/3478

[9] It should be noted that for B-SMEQ we need to deploy an initiator node for the role of TTP.

corresponding to the bucket ID of user's query are taken into consideration. In the second simulation, we study the effect of increasing the number of buckets on communication time for B-SMEQ. We fix the number of records given by $\theta 5$ and searchable attribute with range $[1 \dots 100]$. We repeat the experiment with $s{=}5k$, where $k \in \{1, \dots, 13\}$. In Figure 3.3(b), the x axis shows the number of buckets and y axis shows the total communication time of the Steps 5 and 6. Interestingly, with respect to varying the number of buckets in ascending order, we can see a progressive communication time decreasing. For instance, when the number of buckets is $s = 5$, B-SMEQ provides about 2 times improvement over SMEQ (which can be seen corresponding to $s{=}1$). The reason is that, when we increase the number of buckets, the expected number of records "falling" in each bucket decreases. Moreover, the improvement due to bucketization is higher for $1 \leq s \leq 10$, since, in this setting, when $s > 10$ the expected number of records in each bucket and for each data owner does not considerably change. In order to further clarify this concept, in Figure 3.3(c) we show the results of the same experiment when $s = \{1, 2, 3, 4, 5\}$. Our results show that bucketization decreases communication time dramatically at first; then, the marginal contribution of additional buckets to speed-up tends to decrease. This behavior suggests to find the optimal number of buckets, i.e. the number of buckets where the marginal contribution speed of an additional bucket is negligible. This behavior happens regardless of data distribution (See Appendix A).

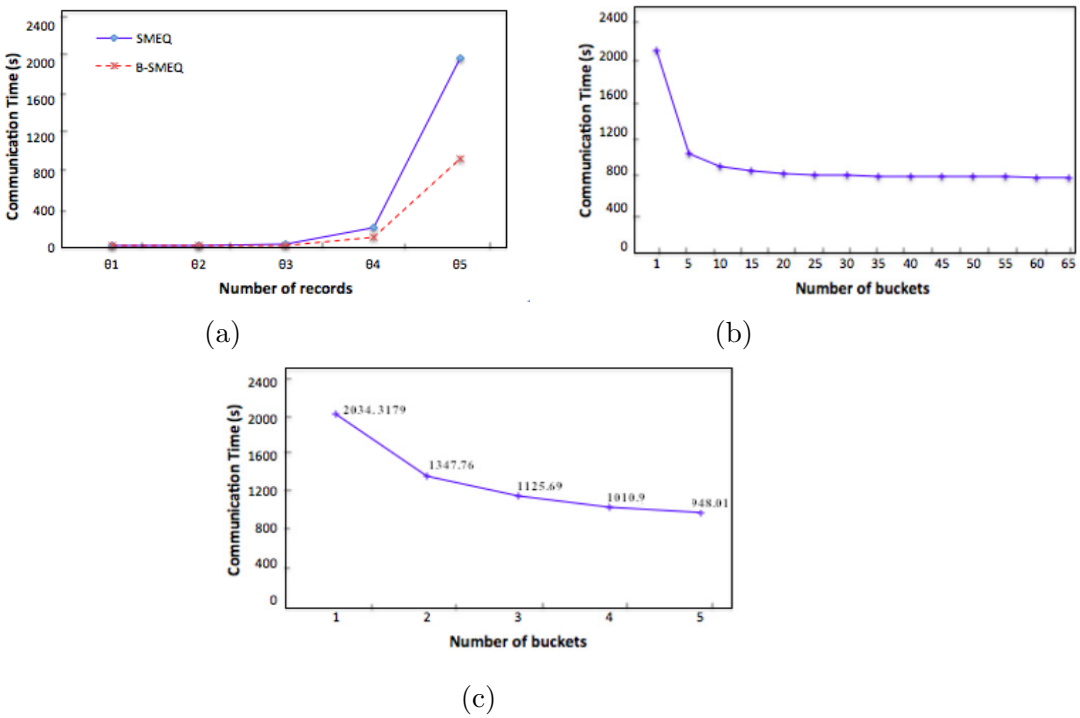


Figure 3.3: (a) Comparison of SMEQ and B-SMEQ Protocols based on communication time (m=3, s=5), (b) Effect of increasing number of buckets on communication time (θ_5, m=3, $1 \leq s \leq 65$) and (c) Effect of increasing number of buckets on communication time (θ_5, m=3, $1 \leq s \leq 5$)

3.4 Statistical analysis attack

In this section, we study the security of our protocol in the presence of malicious participants. Consider a case where a malicious owner aware of the distribution of query values. In such case, with small number of buckets[10] she will guess permutations of the following parties along the ring by observing sufficient number of equality test queries. To illustrate this scenario, let us consider an example of 3 data owners $O = \{O_1, O_2, O_3\}$ with $s = 3$ number of buckets where a malicious party knows the query distribution from the user as shown in Figure 3.4. In our running example, we assume that each owner O_i holds a data partition T_i with the set of attributes $\{B\_id, A\}$, where B_id and A are bucket number and searchable attribute respectively. We define $T_{i,A}$ the column corresponds to attribute A and for each value $v \in T_{i,A}$, $B\_id_i(v)$ denotes the bucket number occurring in Π_i where v falls in. Although the malicious party O_2 does not know which index of TTP's permutation (user requested bucket) corresponds to which interval in her permutation, she can determine the correspondence by observing numerous query execution. For instance, the user sends values 1, 2 and 3 to the malicious party 10, 5, and 15 times, respectively. Then O_2 can infer that the most frequent value 15 corresponds to interval $(30, 60]$ of the permutation she holds, and accordingly she deduce value 2 is likely map to the interval with lowest observed frequencies $(60, 100]$ and so on. Therefore, she finds the correspondence between her own permutation and user requested bucket that allows her to infer the permutations of the following parties after 3 rounds of protocol if parties deploy immediate forwarding i.e., each owner forwards data to the next owner as soon as she receives data.

Let's consider an example to see how this inference occures (see Figure 3.5(a)). Recall that at round 2 of the protocol each owner holds the encrypted data of its own predecessor in encrypted form as shown in Figure 3.5(b). Now suppose that user holding Π_{TTP} wish to look for bucket 2 including the interval $(30, 60]$, she submits $pos = 3$ corresponding to the bucket 2 of public bucketization to owner O_2. Then, owner O_2 finds the third position in $W1$ and gets 2 to select data whose bucket IDs=2. At this point, O_2 can realize that the interval $(30, 60]$ of data owner O_1 is located in bucket 2 (see Figure 3.5(c)). This process is repeated for all owners along

[10]In general, with large number of buckets the malicious party can hardly guess the permutations of participants because the difference of values frequencies between adjacent buckets in query distribution histogram is small.

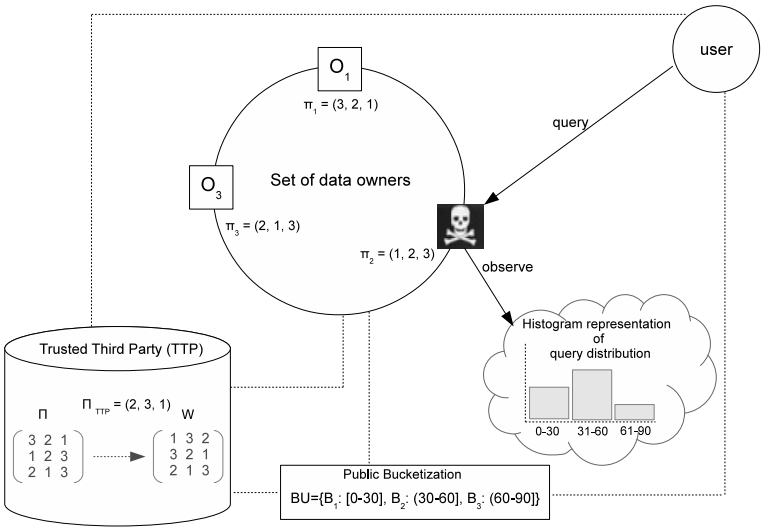


Figure 3.4: A scenario where a malicious participant knows the query distribution

the ring (see Phase 2- Step 5). Consequently, in the next round of protocol execution, each owner encrypts data corresponding to the user bucket ID and sends it to the next data owner, therefore, owner O_2 receives data of O_3 in encrypted form. Observing the bucket ID of the received data, she obtains 1, which indicates that the interval $(30, 60]$ of data owner O_3 is located in bucket ID=1 (see Figure 3.5(d)). Clearly, O_2 can determine the permutations of other participants after observing numerous query execution in intermediate forwarding setting.

In the following, we propose two main approaches for defending against malicious behavior in B-SMEQ protocol.

3.4.1 Countermeasures

- **Lazy Forwarding.** In lazy forwarding implementation, data owner O_i can forward data to O_{i+1} after a random period of time not as soon as she receives data from her predecessor. Since our protocol must run in immediate forwarding for rounds 1 and 2, this solution can apply for a set of m data owners where $m \succ 4$. In order to make it clear, here we consider a scenario of 4 data owners with 3 buckets. At round 1 of the protocol execution, each owner encrypts its data with its own key as shown in Figure 3.6 (a) and sends the data to the next participant in the ring (see Figure 3.6 (b)). At round 2, data owners select correct buckets corresponding to user bucket ID,

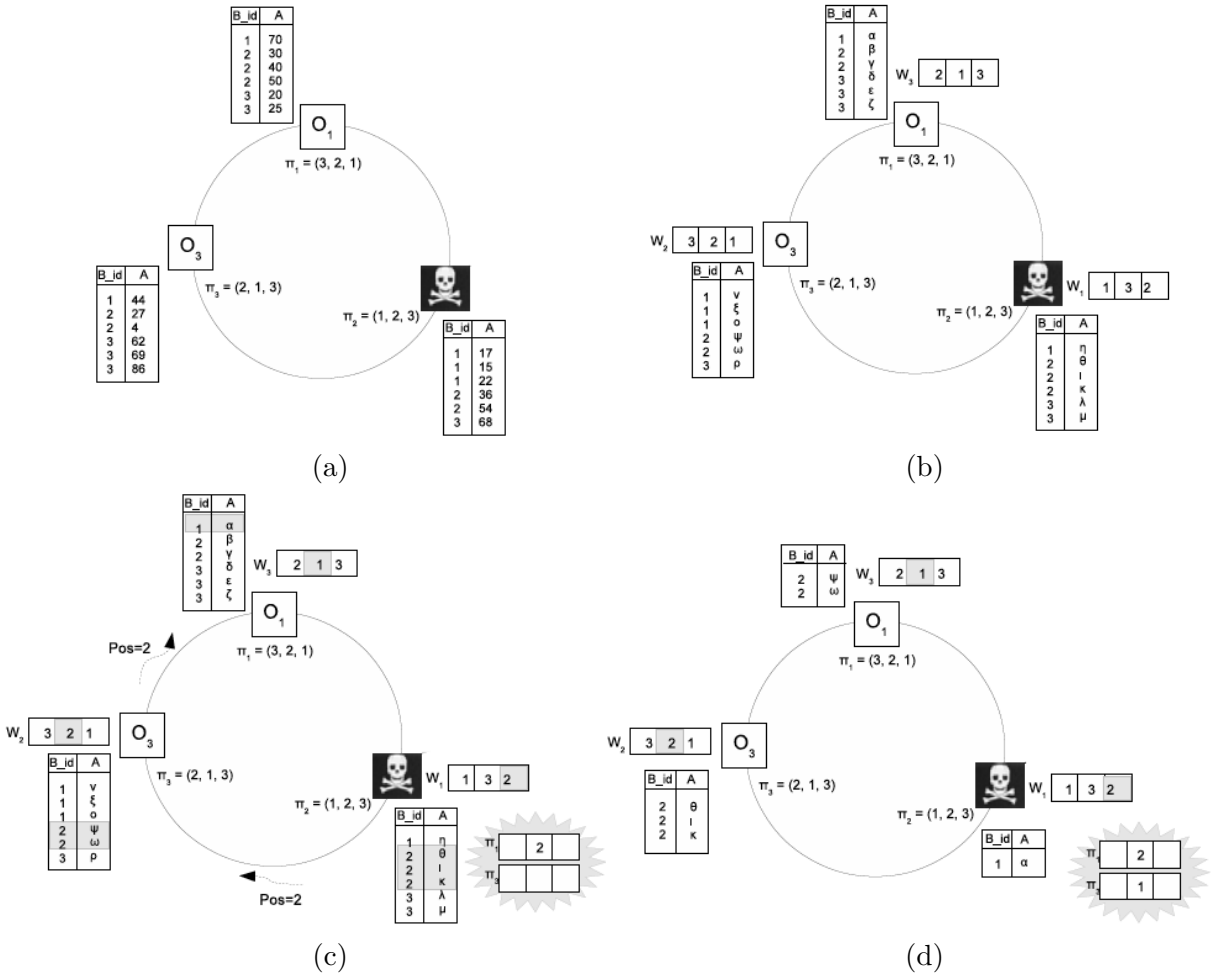


Figure 3.5: (a) data owners with 3 buckets, where $\Pi_{TTP} = (2, 3, 1)$. (b) Each data owner encrypts data and sends it to the next party. (c) Each data owner selects bucket corresponding to the user query $pos = 3$. (d) Selected buckets circulate among data owners to be encrypted by all keys

lets say bucket 1 of public bucketization ($pos = 3$), as described in the previous example. At round 3, data owners except O_1 encrypt the received buckets with their keys and send the data to the next participant as shown in Figure 3.6 (c). In other words, owner O_1 does not follow immediate forwarding. After a short time, O_1 sends the data it holds to the next owner O_2, however, O_2 can not distinguish between data of O_1 and O_4 (see Figure 3.6 (d)).

- **Regenerating Matrix W.** In order to deter malicious party from deducing other permutations along the ring, TTP can regenerate matrix W after a random period of time. Consider round 2 of the previous example where data owners select the correct buckets corresponding to user bucket ID (see Figure 3.7(a)). During the rounds 3 and 4, O_2 infers the permutation of owners $O1$ and O_4 as shown in Figure 3.7(b). Before executing the round 5, TTP asks owner O_4 to change her permutation and accordingly updates interchange matrix W (see Figure 3.7(c)). Since after updating matrix W, the permutation that O_2 holds (W_1) is remained the same, she can not realize the change and incorrectly infers the permutation of O_4 (see Figure 3.7(c)).

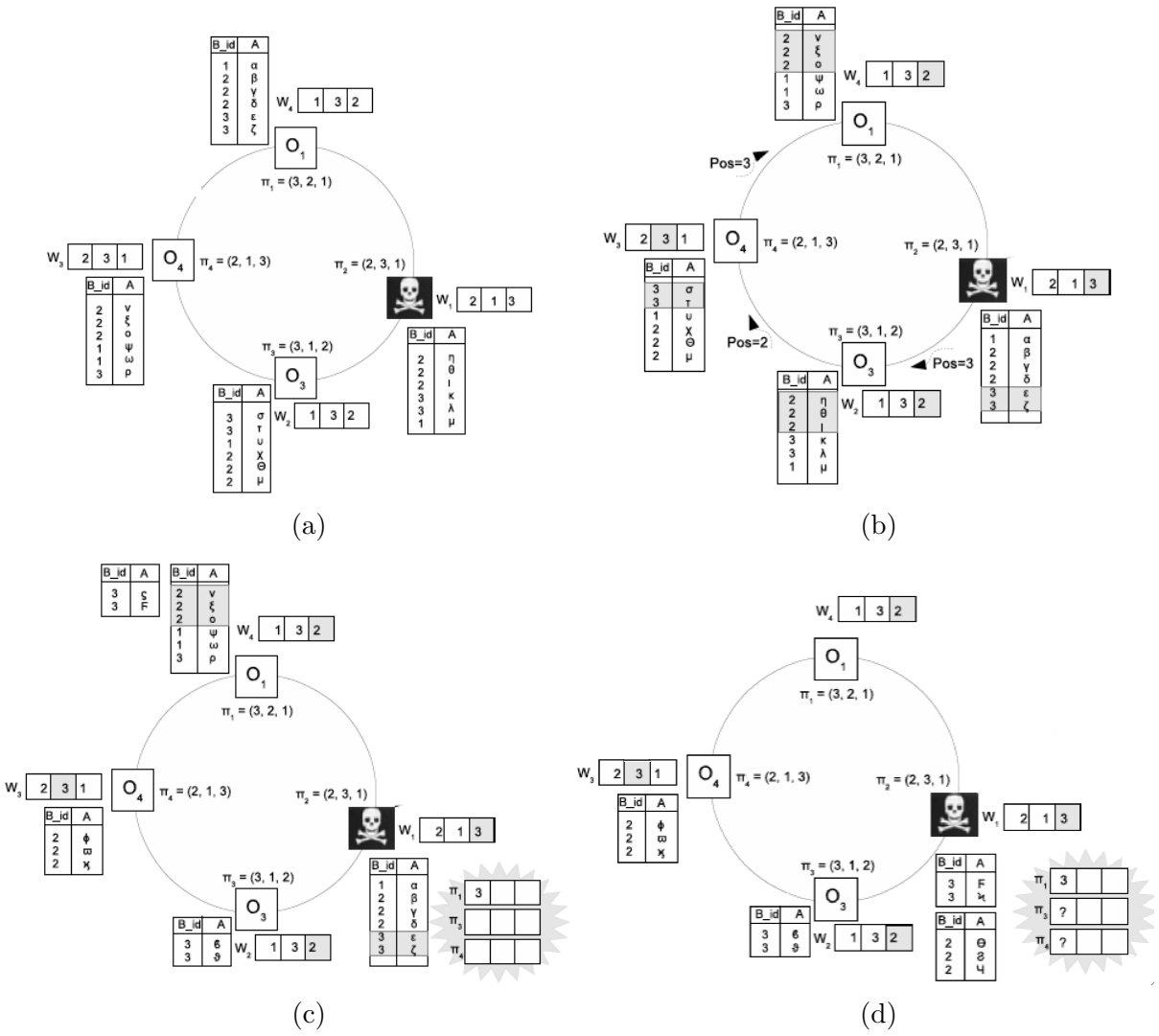


Figure 3.6: (a) 4 data owners with 3 buckets, where $\Pi_{TTP} = (3, 1, 2)$. (b) Each data owner selects bucket corresponding to the user query $pos = 3$. (c) Each data owner encrypts data and sends to the next party. (d) O_2 cannot distinguish between data of O_1 and O_4

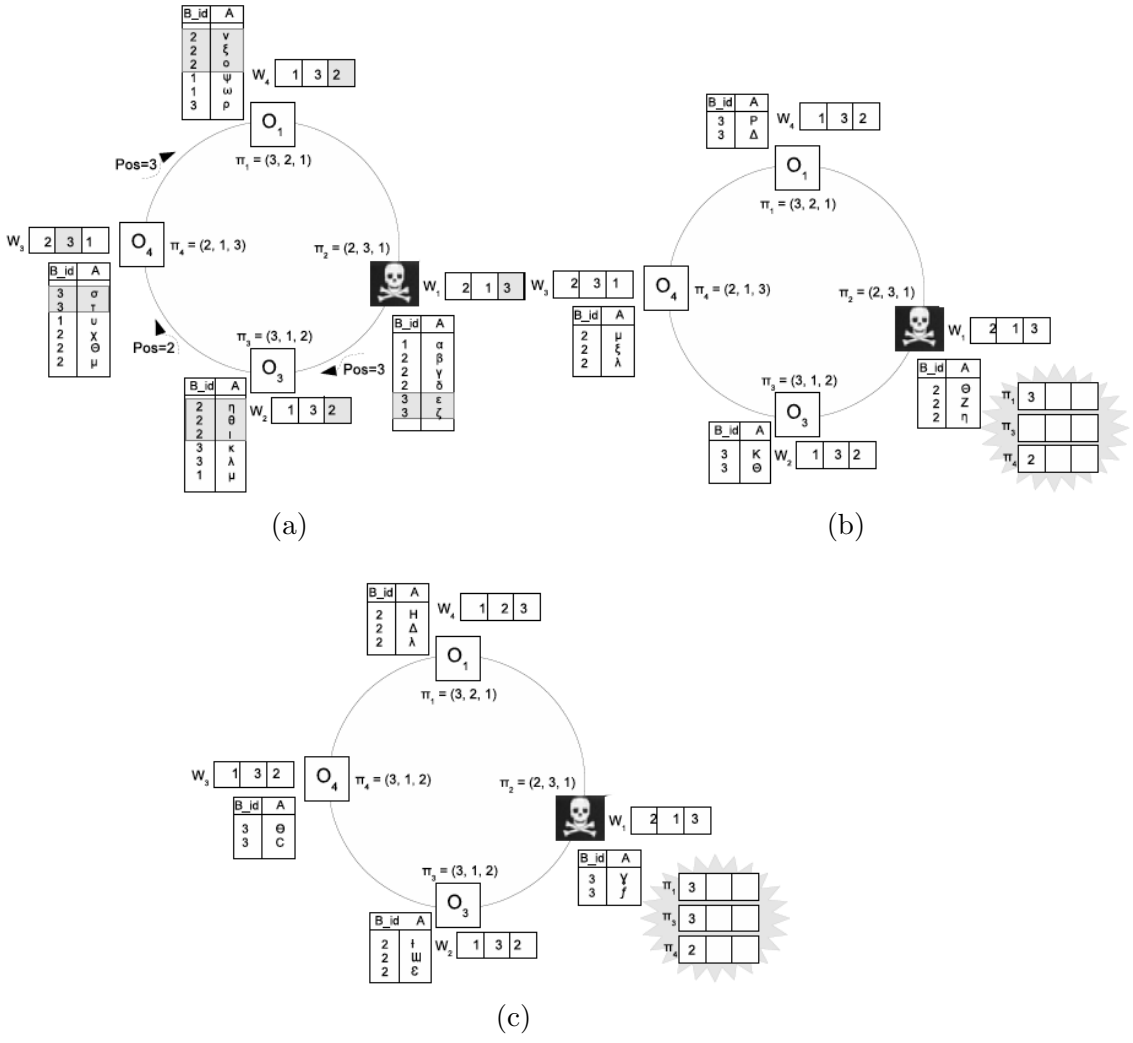


Figure 3.7: (a) 4 data owners with 3 number of buckets $\Pi_{TTP} = (3, 1, 2)$. (b) Each data owner selects bucket corresponding to the user query $pos = 3$. (c) O_2 infers incorrectly the bucket ID of O_4 corresponding to user bucket ID

3.5 A Multi-Party Protocol for Privacy-Preserving Range Queries

In this section, we propose a protocol for privacy-preserving range queries over partitioned data in scalable manner. The idea is to transform a range query in a sequence of equality test queries, each resolved using B-SMEQ. We present two protocols, the first designed to work for range queries on integers, and the second for range queries on real numbers.

3.5.1 Range query over Integer Domain (RDR)

We assume that phase 1 of B-SMEQ protocol has already been executed, and each data owner is holding the vector W_i that can be used to correctly select the buckets containing the values falling in the range query. Accordingly, to describe the protocol we only focus on the query processing phase. Let us assume that data owners $\{O_1, O_2, \ldots, O_m\}$ have received row vectors $\{W_m, W_1, \ldots, W_{m-1}\}$ from matrix W, respectively.

Input. A multi-party system MS as described in Section 3.2.2, with the data owner O_2 as initiator; query range $\mathbf{r} = (r_{min}, r_{max})$; user query values $V_{\mathbf{r}} = \{x \in N | r_{min} \leq x \leq r_{max}\}$

Output. The set of tuples $R = \{t \in T | t.A \in V_{\mathbf{r}}\}$.

Step 1. Both user u and data owner O_i apply the hash function h to their sets. Let $\bar{V}_r = h(V_{\mathbf{r}})$ be the result of hashing for the user and let $\bar{T}_{i,\bar{A}} = h(V_{i,A})$, for each $i \in \{1, \ldots, m\}$, be the hashing of the set values $T_{i,A}$. Two randomly secret keys are selected, k_r for u and $\langle k_i, k_i' \rangle$ for data owner O_i.

Step 2. u encrypts $\bar{V}_r$ and sends $\bar{\bar{V}}_r = f_{k_r}(\bar{V}_{\mathbf{r}})$ to all data owners.

Step 3. Each data owner O_i, $1 \leq i \leq m$, does the following:

3.1. Computes $f_{k_i}\left(\bar{T}_{i,\bar{A}}\right) = Y_i = \{y_i = f_{k_i}(x) | x \in V_{i,\bar{A}}\}$.

3.2. Generates a set of new keys, one for each value of attribute B, as $K_i^B = \{k_{ix} = f_{k_i'}(x) | x \in V_{i,\bar{A}}\}$.

3.3. Encrypts each value x in $T_{i,B}$ with the corresponding key k_{ix} to obtain $Y_i^B = \{E_{k_{ix}}(z) | z \in V_{i,B}\}$, where E is a symmetric encryption function.

3.4. Computes set $I_i = \{(v, f_{k'_i}(v))|v \in \bar{\bar{V}}_R\}$ for the purpose of decrypting the values of attribute B at user site. The data owner O_i randomly reorders the tuples Y_i and Y_i^B and sends them along with I_i to the next owner $O_{(i \bmod m)+1}$.

Step 4. u selects the set of buckets $B = \{i|1 \leq i \leq s\}$ that contains values of $V_{\mathbf{r}}$, and sends $\bar{B} = \{\bar{\pi}_k|k \in B\}$ to O_2 as an initiator.

Step 5. At this step, each data owner O_i holds data Y_{i-1} of O_{i-1}. For every item j belonging to $\bar{B}$, O_2 sets $h_2 = W_{1j}$, selects $Y_1(h_2)$ i.e., the bucket in Y_1 where ID is h_2, and overencrypts with her own key. Then O_2 sends h_2 to the next owner. When data owner O_i receives h_{i-1} from O_{i-1}, she sets $h_i = W_{(i-1)h_{i-1}}$, selects the corresponding bucket $Y_{i-1}(h_i)$ and sends the position h_i to O_{i+1}. This step continues until each data owner selects the buckets corresponding to $V_{\mathbf{r}}$.

Step 6. Each data owner O_i, $1 \leq i \leq m$, forms a set of triples $\langle Y_i, Y_i^B, I_i\rangle$ of her selection from step 5 and sends to the next data owner $O_{(i \bmod m)+1}$.

Step 7. Data owner $O_{(i \bmod m)+1}$ encrypts only Y_i with the key $k_{(i \bmod m)+1}$ and sends the triples $\langle f_{k_{(i \bmod m)+1}}(Y_i), Y_i^B, I_i\rangle$ after reordering to the next participant in the ring. This process is repeated until Y_i is encrypted by all keys of m data owners, obtaining $Z_i = f_{k_1}(f_{k_2}(...(f_{k_m}(Y_i))))$, up to a permutation of the encryption keys[11].

Step 8. Each data owner O_i sends $\langle Z_i, Y_i^B, I_i\rangle$ to O_2, who in turn passes the set $\bar{\bar{V}}_{\mathbf{r}}$ through the ring in order to have it encrypted by all keys $k_1, \ldots, k_m$ for obtaining $\tilde{V}_{\mathbf{r}} = f_{k_1}(\ldots(f_{k_m}(f_{k_r}(\bar{\bar{V}}_{\mathbf{r}}))))$, and then sends back $\tilde{V}_r$ with the set of triples $\langle Z_i, Y_i^B, I_i\rangle$ to the user, for all $i \in \{1, \ldots, m\}$.

Step 9. User u decrypts each value $\tilde{v} \in \tilde{V}_{\mathbf{r}}$ with the own decryption key to obtain set $\hat{V}_{\mathbf{r}}$, and then for each i, $1 \leq i \leq m$:

- Finds tuples in Z_i whose entry related to the searchable attribute is equal to jth value of $\hat{V}_{\mathbf{r}}$;
- Considers the entry corresponding to attribute B of those tuples;
- Decrypts the jth value of I_i with k_r, obtaining $f_{k_i}(\tilde{v})$;
- Uses $f_{k_i}(\tilde{v})$ to decrypt the corresponding entry in Y_i^B.

[11]The keys $k_1, k_2, \ldots, k_m$ represent a commutative set of keys.

For instance, suppose that user u asks for tuples belonging to range $\mathbf{r} = [29, 42]$ w.r.t. attribute $T.A$. According to the proposed protocol, u sends an equality query for each integer value in

$$V_{\mathbf{r}} = \{29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42\}$$

Then, each equality query is computed by using B-SMEQ protocol. While straightforward, this approach has two primary disadvantages: first, it has high computation and communication costs, since it computes $|V_{\mathbf{r}}|$ equality queries; second, it has a limitation on domain type, since not all ordered domains can be handled in this way. For instance, floating point values do not lead themselves well to this type of representation.

3.5.2 Range query over real domain (RDR)

To handle the abovementioned problems, we split real values of searchable attribute into two values to provide a finite discrete domain in which the range query can be resolved through IDR protocol. In the following we describe the pre- and post-processing phases of the protocol.

Pre-processing phase

- *Data owners side*: Each data owner O_i splits the attribute $T.A$ into two sub attributes, $T.X$, $T.Y$, as follows:
  - $\forall v \in T.A$, $v = v_x + v_y$
  - Suppose that $v_1 v_2 \dots v_c$ is the integer part of v. Then we set $v_x = v_1.10^{c-1}$, $v_y = v - v_x$. Observe that $v_x \in N$ by definition and that T.Y is considered an extra attribute, only T.X is used as searchable attribute.

- *User side*: Suppose that q is a range query the user u wants to perform, with range $\mathbf{r} = (r_{min}, r_{max})$.
  - User u splits range boundaries as described above, obtaining $r_{min} = r_{min,x} + r_{min,y}$, $r_{max} = r_{max,x} + r_{max,y}$
  - Suppose that α is the most significant digit of $r_{min,x}$ and β is the most significant digit of $r_{max,x}$, and that c_1 and c_2 are the number of digits of $r_{min,x}$ and $r_{max,x}$ respectively.
  - The range query q is mapped onto 3 categories of equality queries:
    $q^{(1)} := \{i.10^{c_1-1}, i \in \{\alpha, \alpha+1, \dots, 9\}\}$

$q^{(2)} := \{\gamma.10^{\bar{c}}, \gamma \in \{1, 2, \ldots, 9\}, c_1 - 1 < \bar{c} < c_2 - 1, \bar{c} \in N\}$
$q^{(3)} := \{\eta.10^{c_2-1}, \eta \in \{1, 2, \ldots, \beta\}\}$

- The query sets $q^{(1)}$, $q^{(2)}$, $q^{(3)}$ are computed then by adopting IDR protocol.
- Observe that for each range $[10^i, 10^{i+1}], i \geq 1$, we compute at most 10 equality queries, whereas IDR computes $(10^{i+1} - 10^i)$ equality queries.

Post-processing phase

- *User side.* Suppose that $S^{(1)}, S^{(2)}, S^{(3)}$ are the results obtained by user u for the equality sets $q^{(1)}, q^{(2)}, q^{(3)}$, respectively. By definition of $q^{(2)}$, all the tuples in $S^{(2)}$ are in the range $\mathbf{r} = (r_{min}, r_{max})$. User u should only select from $S^{(1)}$ those tuples t for which $t.y \geq r_{min,y}$, and from $S^{(3)}$ those tuples t such that $t.y \leq r_{max,y}$.

3.5.3 A worked-out example

Consider the ring of three data owners shown in Figure 3.8 (a), where each owner has a searchable attribute A on real numbers along with corresponding bucket IDs. According to the protocol, each data owner first splits every $v \in A$: for instance, data owner O_1 follows the pre-processing phase and obtains $v_x = 10$, $v_y = 7.7$ for $v = 17.7$ (Figure 3.8 (b)).

Now, suppose that user u asks for tuples t whose searchable attribute value is within the range $\mathbf{r} = [8.62, 242]$. Then, u splits $r_{min} = 8.62$ and $r_{max} = 242$ to $r_{min,x} = 8$, $r_{min,y} = 0.62$ and $r_{max,x} = 200$, $r_{max,y} = 42$, respectively. By definition, user u sets $\alpha = 8$, $\beta = 2$, $c_1 = 1$ and $c_2 = 3$ to obtain $q^{(1)} = \{8, 9\}$, $q^{(2)} = \{10, 20, \ldots, 90\}$ and $q^{(3)} = \{100, 200\}$. Then u sends the equality queries $q^{(1)}$, $q^{(2)}$ and $q^{(3)}$ to data owners, who in turn execute IDR protocol and return the corresponding results $S^{(1)}$, $S^{(2)}$ and $S^{(3)}$ to the user.

Finally, a post-processing phase is required to weed out the false positives from $S^{(1)}$ and $S^{(3)}$. Thus, user u selects tuples t from $S^{(1)}$ for which $t.y \geq 0.62$ and from $S^{(3)}$ those tuples t such that $t.y \leq 42$.

3.5.4 False positive analysis

In this section, we analyze the impact that data and query distribution and the number of buckets have on the number of false positives that user gets in the query result (i.e. the tuples not belonging to the query range). More specifically, we tested RDR protocol for $m = 3$ data owners, respectively

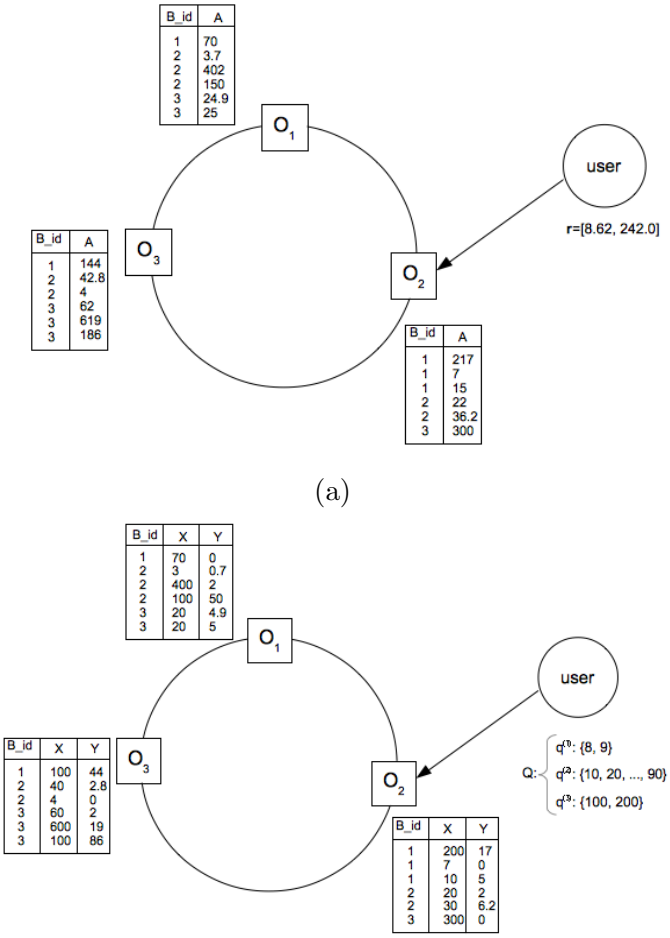


Figure 3.8: (a) Three data owners with real searchable attribute A, (b) Each data owner splits attribute A into two sub attributes X and Y

with $n1 = 50000$, $n2 = 60000$, $n3 = 70000$ tuples in their own tables T_1, T_2 and T_3. We repeated the experiment with two data distributions: uniform distribution (UD) and normal distribution (ND) (mean $\mu = 50.5$, standard deviation $sd = 10$). Moreover, we also tried out two categories of queries, whose range was chosen respectively with a uniform and a normal distribution, with mean μ and standard deviation sd. Moreover, we repeated our experiment with $s = (5, 10, 15, 20, 25)$ number of buckets, computing exactly the same queries for each value of s.

In Figure 3.9 we show the averaged number of false positives (FP) across all the performed queries. Firstly, we can observe that FP always decreases

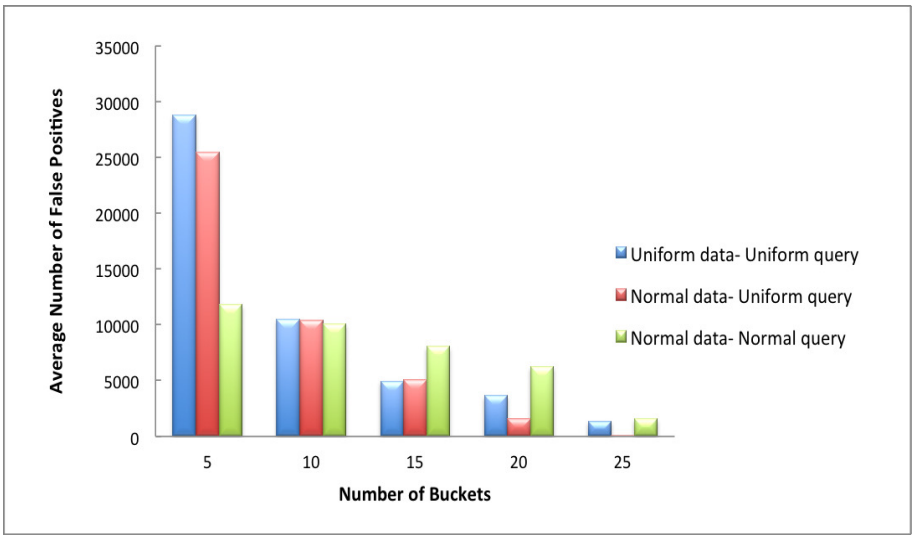


Figure 3.9: Number of False Positives vs Number of Buckets

when s increases. This is more evident for uniform queries, for both UD and ND data. Furthermore, we observe that, with uniform queries, UD data has higher FP than ND data for $s = 5, 20, 25$, while it is almost the same for the other choices of s. Moreover, normal queries on ND data (green bar) has FP number much lower than uniform queries on UD/ND data when $s = 5$, almost the same when $= 10$, and higher FP number for $s > 10$. This behavior suggests that for normally distributed queries on ND data bucketization is less effective, since both the data and the queries are concentrated around μ, thus leading to some buckets containing almost all the tuples.

3.5.5 Privacy Issues

In this section, we study the security of our protocol for single query processing in the presence of honest-but-curious data owners[12]. We omit to discuss the preservation of privacy for equality queries using B-SMEQ, which has already been discussed in Section 3.3.3. Hence, we only have to discuss whether splitting the searchable attribute A into sub-attributes X and Y may lead to data leakage. For this reason, we focus on the privacy of Steps 5 and 7 where the records of each data owner are selected according to the user bucket ID.

[12]For the case where a malicious participant can become aware of the distribution of query values by receiving multiple queries, we already discussed in Section 3.4.

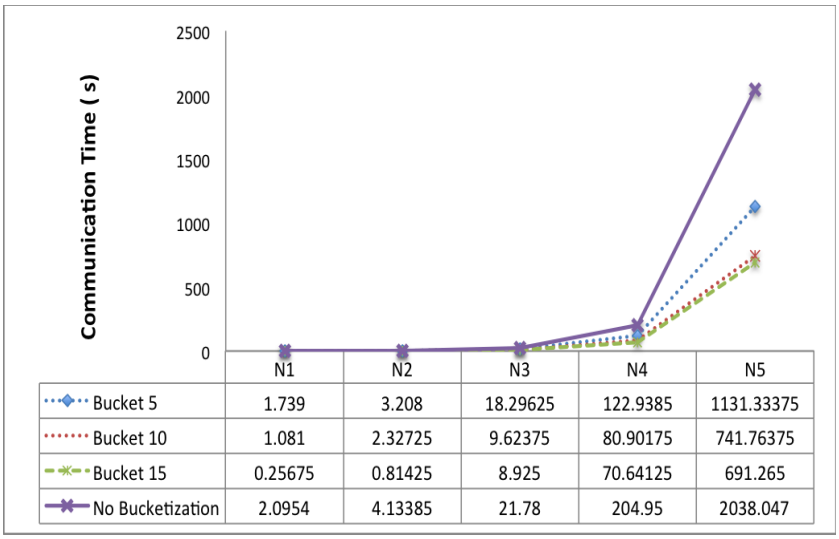
- *S*tep 5: Let us recall that at this step, each data owner holds the encrypted data of its predecessor in the ring. The only information that is visible to each data owner are bucket labels corresponding to tuples of its predecessor. Since the values of the predecessor are in encrypted form, there is no way in which the owner can over the relations between bucket ID and values of attributes X and Y. In this step, the owners select the bucket corresponding to user query, observing thereby the size of the bucket. However, this does not reveal the size of buckets, since the owner does not know which bucket ID corresponds to the received tuples from the predecessor.

- *S*tep 7: At each round of the protocol, each data owner receives new encrypted values of attribute X with different keys. While the values of attribute Y remained unchanged during the protocol execution, the data owner can not infer the relationship between received data sets.

3.5.6 Time Complexity Analysis

In this section, we perform some experiments to clearly illustrate the benefit of bucketization approach for processing range queries in SMC paradigm in terms of communication time. We use Castalia simulator on a Linux machine with dual Intel CPU running at 2.26 GHz, and 2GB Ram. In the experiments, we construct 5 nodes in Castalia simulator including 3 data owners, 1 user and the TTP. Each data owner holds a table with one searchable attribute for range query generated according to uniform distribution. To encrypt the searchable attribute of each data owner's table, we implemented a simple *commutative encryption* protocol based on *exponentiation modulo p*. For the range query, we prepared a set of 8 wide range queries Q, where the range predicates are defined on the selected attribute.

We measured the communication time of our protocol for different number of buckets and different number of records, which are partitioned horizontally among 3 nodes. The experimental results are shown in Figure 3.10. Each point in the figure corresponds to the average communication time obtained by running the query workload Q. The x axis shows the number of tuples N_i partitioned among data owners, where $N_i = 18.10^{i-1}$ for $1 \leq i \leq 5$. The y axis shows the average time of the query workload on the data owner side, which is dominated by the time of circulating buckets containing range values along the ring. The solid line of the plot displays the communication time of our protocol without using bucketization, whereas the dotted lines display the results of our protocol, which adopts bucketization on searchable attribute.



	N1	N2	N3	N4	N5
Bucket 5	1.739	3.208	18.29625	122.9385	1131.33375
Bucket 10	1.081	2.32725	9.62375	80.90175	741.76375
Bucket 15	0.25675	0.81425	8.925	70.64125	691.265
No Bucketization	2.0954	4.13385	21.78	204.95	2038.047

Figure 3.10: Communication Time vs Number of Buckets

The dotted lines show the communication time when the searchable domain attribute $[1.0, 100.0]$ is divided into 5, 10 and 15 buckets having the same size. The difference in communication time between solid and dotted lines increases fairly slowly when the number of tuples is relatively small, but it grows much faster as the number of tuples increases. The results confirm the fact that bucketization approach is effective in reducing the number of tuples circulating along the ring, resulting in lower communication time. Moreover, bucketization decreases communication time dramatically at first, for example when the number of buckets is 10, the protocol provides about 2 times improvement; however, then the marginal contribution of additional buckets to speed up tends to decrease. This behavior depends obviously on the domain of the searchable attribute, and does not depend on data and query distribution. Some examples are given in Appendix A.

3.6 A Multi-Party Protocol for Privacy-Preserving Equi-join Queries

In cloud environment, querying against a large repositories of data on distributed servers in a privacy way is still a challenge, especially for complex queries such as join operation. There are many practical applications where join queries need to be supported, including social networks like Facebook (e.g., aftermentioned example). Among the possible solutions for performing such queries while preserving privacy, a recently emerging approach is secure multi-party computation (SMC). Here we investigate the combination of bucketization and MapReduce techniques in SMC scenario to compute *equi-join* queries over large databases. A "mapping" phase reorders the data in order to assign each data owner to its own encrypted data, and to bring data under the same encryption key. After mapping, in the "reduce" phase the equi-join is computed via bucketization on the equality attribute, which allows owners to work only over a subset of tuples and thereby to reduce communication and computation complexity.

Example: Consider an activity of the social network *facebook* with three servers S_1, S_2, and S_3. Each server has a different set of users, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_4, u_5\}$, and $S_3 = \{u_6, u_7, u_8\}$ (see Figure 3.11). For each $i \in \{1,\ 2,\ 3\}$, server S_i holds a table $T_i = (user\_id, user\_job)$, which shows the job of users residing at server S_i. Users in different servers can connect to the other servers and do some friendship requests. For instance, user u_1 is connected to server S_2, then to the server S_3 and sends some friendship requests to users hosting in S_2 and S_3. For each request from server S_i, a record is inserted in table $\hat{T}_i = (sender\_id, receiver\_id, sender\_job)$ located at server S_i. Now suppose that we want to know the job of users who send friendship request to users who are *professors*. This problem is a typical situation of big partition tables where data encrypted with different keys are located at different servers. A MapReduce technique is applied so that a new virtual node is created for each user and table $\hat{T}_i$ in each server is parted based on *sender_id* attribute among the new nodes. For each new node a table $T^{'} = (receiver\_id, sender\_job)$ is created containing the rows of the corresponding user. Then nodes are arranged in a logical ring on top of the servers as shown in Figure 3.11. Afterwards, a SMC technique is invoked to compute the query $T \underset{(T_{user\_id}=T^{'}_{receiver\_id})\wedge(T_{user\_job}=professor)}{\bowtie} T^{'}$, whose result is a table with attributes $(receiver\_id, sender\_job, user\_job)$.

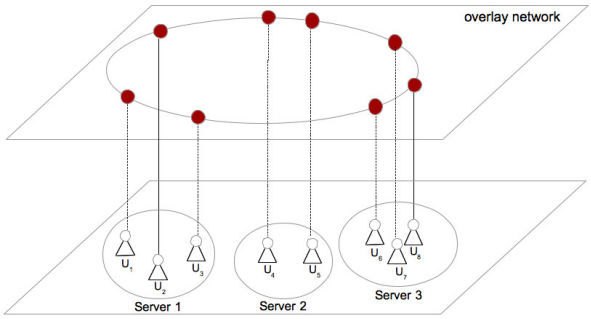


Figure 3.11: nodes as data owners are connected by logical ring as an overlay on top of the real network irrespective of their topologies

This table contains the information needed to answer the initial question.

3.6.1 MPC-based equi-join: Problem statement

Our multi-party system is described as follows:

- Two tables T and $T^{'}$ with n records, which have been horizontally partitioned among a set of m data owners $O = \{O_1, \ldots, O_m\}$ that are organized in a ring.
- Each owner O_i receives two shares T_i and $T_i^{'}$ such that $\bigcup_{i=1}^{m} T_i = T$, $\bigcup_{i=1}^{m} T_i^{'} = T^{'}$, and $T_i \cap T_j = \phi$, $T_i^{'} \cap T_j^{'} = \phi \ \forall i, j \in \{1, \ldots, m\}$.
- Each share couple $(T_i, T_i^{'})$ is encrypted with private key of O_i and contains a set of join attributes $A_1, \ldots, A_{k_1}$, a set of searchable attributes $B_1, \ldots, B_{k_2}$ and a set of extra attributes $C_1, \ldots, C_{k_3}$, which k_1, k_2 and k_3 are not necessarily equal. For the sake of simplicity, we assume that each share T_i includes one join attribute and one searchable attribute $T_{i,A}$ and $T_{i,B}$ with data set $V_{i,A}$ and $V_{i,B}$, respectively and each share $T_i^{'}$ includes one join attribute $T_{i,A'}^{'}$ and one extra attribute $T_{i,C'}^{'}$ with data set $V_{i,A'}$ and $V_{i,C'}$, respectively.

Given an equi-join query Q over T and $T^{'}$ for selection value v, the problem consists in finding all tuples $t \in T$ for which there exist $\bar{t} \in T^{'}$ such that $t_A = \bar{t}_{A'}$ and $t_B = v$. In database terminology, user wants to compute the join of T and $T^{'}$ on join condition $T_A = T_{A'}^{'}$ and selection condition $T_B = v$. That is, $q = \pi_{T_{C'}^{'}}(\sigma_{T_B=v}(T \underset{T_A=T_{A'}^{'}}{\bowtie} T^{'}))$.

3.6.2 The protocol

The protocol we propose for equi-join is composed of four main stages: preprocessing, selection, query computation, and postprocessing.

1. *P*reprocessing. Data owners and user hash and encrypt their data with their own keys. The data owners then send their data in encrypted form to the next owner along the ring.

2. *Selection.* In this step each data owner selects the bucket corresponding to the selection condition of query. This stage is carried out utilizing B-SMEQ protocol over shares T_i and $T_i^{'}$.

3. *Query computation.* There are two levels to the computation of join query: initial and final. During the initial level, every participant locally computes equi-join between encrypted shares received at stage one to get partial result. For instance, consider a simple model of two data owners O_1 and O_2 with shares $(T_1, T_1^{'})$ and $(T_2, T_2^{'})$, respectively associated to tables T and $T^{'}$ (Figure 3.12). Now suppose that user u wants to find the results R of $T \bowtie T^{'}$ over join attribute. Let R_i^j denotes equi-join result between shares associated to T and $T^{'}$ residing at owner i and j, respectively. After the initial level, owners O_1 and O_2 get partial results R_1^1 of $T_1 \bowtie T_1^{'}$ and R_2^2 of $T_2 \bowtie T_2^{'}$ respectively, which are not complete because the following results have not been computed yet:

   $$R_2^1 \leftarrow T_1 \bowtie T_2^{'} \text{ and } R_1^2 \leftarrow T_2 \bowtie T_1^{'}$$

   Therefore, at finial level each owner overencrypts shares and partial result of the first level with its own key and sends them to the next owners along the ring to be encrypted by all keys. Then, initiator collects data encrypted with different order of keys from owners and find the final result exploiting commutative encryption property.

4. *Post-processing.* The last stage allows user to filter tuples whose extra attribute of T are equivalent to the selection condition of user query.

Let's recall that the B-SMEQ protocol requires a TTP generates an interchange matrix W and sends the row vectors of W to the participants. For instance, suppose that owners $O = \{O_1, \ldots, O_m\}$ have received vectors $\{W_m, W_1, \ldots, W_{m-1}\}$ from TTP, respectively. Then, owner O_2 is considered as initiator and the aim is to compute the results of $q = \pi_{T_{i,A}, T_{i,B}, T^{'}_{i,A^{'}}} T \bowtie T^{'}$

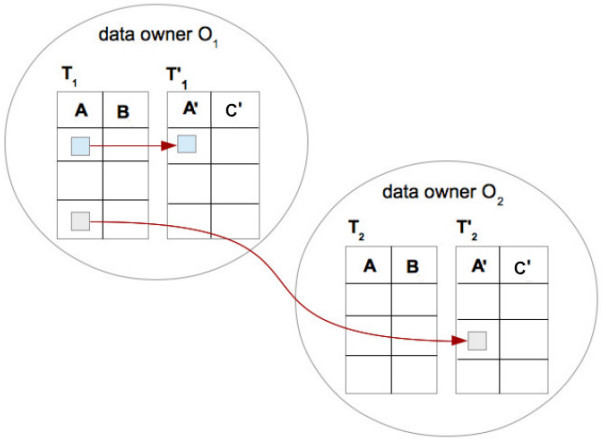


Figure 3.12: An example that shows local join computation of O_1 is not complete

with the query condition $c = (T_{i,A} = T'_{i,A'}) \wedge (T_{i,B} = v)$, which can be computed as the follows:

$$q = (\bigcup_{i=1}^{m} \pi_{T_{i,A},T_{i,B},T'_{i,A'}}(T_i \underset{c}{\bowtie} T'_i)) \cup (\bigcup_{i,j=1,i\neq j}^{m} \pi_{T_{i,A},T_{i,B},T'_{i,A'}}(T_i \underset{c}{\bowtie} T'_j))$$

Preprocessing

Step 1. Data owners apply hash function h to their join attribute A and searchable attribute related to the selection condition B.
Let $\bar{T}_{i,\bar{A},\bar{B}} = h(V_{i,A}, V_{i,B}), \hat{T}_{i,\hat{A}} = h(V_{i,A'})$, for each $i \in \{1,\ldots,m\}$, and let $v' = h(v)$ be the result of hashing value v corresponding to selection condition posing by user. The user and data owners randomly choose a secret key, k_r for u and $\langle k_i, k'_i \rangle$ for data owner O_i.

Step 2. u spans the encrypted hash value $v'' = f_{k_r}(v')$ to all data owners.

Step 3. Each data owner O_i, $1 \leq i \leq m$, does the following:

3.1 Computes $Y_{i,A,B} = f_{k_i}\left(V_{i,\bar{A}}, V_{i,\bar{B}}\right)$

3.2 Generates a set of new keys for encrypting every value w of attribute C' as ext(w), $K_i^{ext(w)} = \{k_{ix} = f_{k'_i}(x) | x \in V_{i,\bar{A}}\}$

3.3 Computes $Y'_{i,A'} = f_{k_i}(V_{i,\hat{A}})$ and $Y'_{i,C'} = E_{k_{ix}}(V_{i,C'})$ where E is an encryption function, which can be efficiently inverted given key k_{ix}

3.4 Computes $I_i = E_{k_i'}(v'')$ for the purpose of decrypting values of attribute C' by user

3.5 O_i randomly reorder the tuples related to partitions $Y_{i,A,B}$, $Y'_{i,A',C'}$ and sends tuples $\langle Y_{i,A,B}, Y'_{i,A',C'} \rangle$ with their corresponding I_i to the $O_{(i \ mod \ m)+1}$

Selection

Step 4. u sends $\pi(k) = \alpha$ to the initiator O_2, where k is bucket ID that contains value v of selection condition.

Step 5. Let's recall that at this step, each owner O_i holds partitions Y_i, Y_i' of O_{i-1} and a vector from matrix W corresponding to π_{i-1}. O_2 sets $h_2 = w_{1\alpha}$ i.e., the bucket in Y_1 where bucket ID corresponding to $Y_{1,B}$ is h_2 gets the result $\bar{Y}_1$, and sends the position to the next participant. This step continues until each owner selects the bucket corresponding to the equality predicate of query.

Query computation

Step 6. During the initial level, each data owner O_i, $1 \leq i \leq m$, does the following:

6.1 O_i executes a local equi-join on join attributes $\bar{A}$ and A'between shares[13] $\bar{Y}_{i-1}$ and Y'_{i-1} to get result $R_{i-1}^{i-1} = (\bar{Y}_{i-1,\bar{A}}, \bar{Y}_{i-1,\bar{B}}, Y'_{i-1,A'})$. Hence, R_{i-1}^{i-1} includes tuples $\bar{t} \in \bar{Y}_{i-1}$ for which exists $t' \in Y'_{i-1}$ such that $\bar{t}.\bar{A} = t'.A'$

6.2 Data owner O_i overencrypts join and searchable attributes of shares $\bar{Y}_{i-1}, Y'_{i-1}$ and R_{i-1}^{i-1} with its own key and sends the result set $Z_i = \langle \bar{Y}_{i-1}, Y'_{i-1}, R_{i-1}^{i-1}, I_i \rangle$ to the next owner $O_{(i \ mod \ m)} + 1$. This process is repeated until join and searchable attributes of Z_i be encrypted by all keys along the ring.

Step 7. At this step, final level, owners O (excluding O_2) send Z_1 and $\{Z_i\}_{i \in 3,\ldots,m}$ to O_2 for completing the query result.

Step 8. O_2 collects all shares $\bar{Y}_i$ and Y_i', $i \in \{1, \ldots, m\}$, executes local join between them to get $\overline{R} \leftarrow (\bigcup_{i=1}^{m})\bar{Y}_i \underset{\bar{Y}_{i,\bar{A}}=Y'_{i,A'}}{\bowtie} (\bigcup_{i=1}^{m})Y_i'$, and sends $R \leftarrow \bar{R} \cup (\bigcup_{i=1}^{m} R_i^i)$ to the user side.

[13]For brevity we assume $\bar{Y}_{((i+2) \ mod \ m)+1}$ and $Y'_{((i+2) \ mod \ m)+1}$ as $\bar{Y}_{i-1}$ and Y'_{i-1}, respectively.

Post-processing

Step 9. User passes $f_{k_r}(v^{'})$ through the ring to have it encrypted by all keys $k_1, \ldots, k_m$ for obtaining $y'_R = f_{k_1}(\ldots(f_{k_m}(f_{k_r}(v^{'}))))$.

Step 10. User 1) decrypts y'_R with k_r to obtain $\bar{y}_R = f_{k_1}(\ldots(f_{k_m}(v^{'})))$; 2) finds tuples in R whose entry related to searchable attribute is equal to $\bar{y}_R$; 3) considers the entry corresponding to extra attribute of those tuples; 4) decrypts value I corresponding to those tuples with k_r to hobtain $f_{k_i}(v^{'})$; 5) uses $f_{k_i}(v^{'})$ to decrypt the corresponding entry of extra attribute.

3.6.3 Privacy Issues

In this section, we will study the security of different stages of the protocol under the assumption of *honest-but-curious* behavior.

- *P*reprocessing (Steps 1-3). O_i receives shares from O_{i-1} in encrypted form, which are indistinguishable from uniformly random number because applying hash function to actual values provides random inputs for commutative encryption. Therefore, O_i is not able to learn the original values from encrypted shares of O_{i-1}. Moreover, O_i does learn the domain size $|Y_{i-1,A,B}|$ and $|Y^{'}_{i-1,A',C'}|$, which are not useful to infer additional information.

- *S*elect (Steps 4, 5). The only information that O_i gets in plaintext is the bucket indices corresponding to share T_{i-1} of owner O_{i-1}. But, since each owner chooses local permutation and data corresponding to those indices are in encrypted form, there is no way in which owner O_i finds the relation between bucket IDs and other attributes.

- *Q*uery computation (Step 6-8). At first level, O_i executes local join between encrypted shares of O_{i-1}, which does not help her to deduce the join information, unless she is able to break the encryption. Furthermore, learning the number of tuples of local join will not help her to learn anything. At final level, each data owner receives shares encrypted with different keys, which provides protection from relations inference.

- *P*ost-processing (Steps 9, 10): In this stage, user can not infer any additional information besides all values of extra attribute corresponding to the intersection of searchable attribute and user's selection condition v. This comes from the fact that the decryption keys are obtained if and only if the values corresponding to searchable attribute do intersect with v.

3.6.4 Time Complexity Analysis

In this section we provide a theoretical and experimental analysis of the protocol time complexity, taking into account both computation and communication costs.

Theoretical cost analysis

Computation time Here we analyze the computation costs with the assumption that each data owner has $\frac{n}{m}$ records for each share associated to T and $T^{'}$ (uniform distribution). Let C_h be the cost of evaluating the hash function h, C_f be the cost of encryption/decryption by function f, and C_E be the cost of encryption/decryption by function E. Next, we will follow our protocol step-by-step in order to quantify the complexity of each step.

Step 1: $C_h(3n+1)$
$-C_h(|T_{1,A}|)+\ldots+C_h(|T_{m,A})|+C_h(|T_{1,B}|)+\ldots+C_h(|T_{m,B}|)=C_h(2n)$, the cost of hashing partition T
$-C_h(|T^{'}_{1,A'}|)+\ldots+C_h(|T^{'}_{m,A'})|=C_h(n)$, the cost of hashing partition $T^{'}$
$-C_h(v)=C_h(1)$, the cost of hashing user value v to obtain $v^{'}=h(v)$

Step 2: $C_f(1)$
$-C_f(v^{'})=C_f(1)$, the cost of encrypting user hash value to get $v^{''}=f_{k_r}(v^{'})$

Step 3: $C_f(4n)+C_E(n+m)$
for each data owner O_i, $1\leq i\leq m$:

3.1 $C_f(|\bar{T}_{i,\bar{A}}|)+C_f(|\bar{T}_{i,\bar{B}}|)=C_f(2\frac{n}{m})$ to obtain $Y_{i,A,B}$ with key k_i

3.2 $C_f(|\bar{T}_{i,\bar{A}}|)=C_f(\frac{n}{m})$ for creating a set of new keys for attribute $C^{'}$ of $\hat{T}_i$

3.3 $C_f(|\hat{T}_{i,\hat{A}}|)+C_E(|\hat{T}_{i,\hat{C}}|)=C_f(\frac{n}{m})+C_E(\frac{n}{m})$ to obtain $Y^{'}_{i,A,B'}$

3.4 $C_E(I)=C_E(1)$ for encrypting value $v^{''}=f_{k_r}(v^{'})$ with $k_i^{'}$ as a decryption key for values of $\hat{T}_{i,\hat{C}}$ at user side

3.5 No computation cost

The cost of Step 3 for each data owner is $C_f(\frac{4n}{m}) + C_E(\frac{n}{m} + 1)$, and accordingly $C_f(4n) + C_E(n + m)$ for m owners

Step 4, 5: No computation cost

Step 6: $mC_f(\frac{2n}{s}) + mC_f(n) + mC_f(\frac{n}{s})$
for each data owner O_i, $1 \leq i \leq m$:

6.1 No computation cost

6.2 Having $|R'_{i-1}| = \frac{n}{2sm}$ where s is the size of bucket, the cost of this step is $mC_f(\frac{2n}{ms}) + mC_f(\frac{n}{m}) + mC_f(\frac{2n}{2ms})$ for encryption searchable and join attributes of shares $\bar{Y}_{i-1}, Y'_{i-1}, R'_{i-1}$. Consequently, $m^2C_f(\frac{2n}{ms}) + m^2C_f(\frac{n}{m}) + m^2C_f(\frac{2n}{2ms})$ for m data owners.

Step 7, 8: No computation cost

Step 9: $mC_f(1)$
$-mC_f(1)$ for encrypting user value with all keys of m data owners

Step 10: $C_f(1) + C_E(m + |R|)$
- $C_f(1)$ is the cost of decrypting y'_R to find records $R = \{r_i = x_i \cap v | x_i \in \bar{R}\}$ using commutative encryption property. With the assumption that $\bar{R}$ has obtained from m data owners, it needs $mC_E(I_i)$, $i \leq 1 \leq m$ to derive keys for decrypting join information of R records with the cost equal to $|R|C_E(1)$.

Assuming unitary cost per record for C_h, C_f and C_E, the total computation cost of the protocol is given by:

$$\begin{aligned} C_p = \quad & (3n + 1) + (2 + 4n + mn + m + \tfrac{3mn}{s}) \\ & + (n + 2m + |R|) \in O(mn) \end{aligned}$$

Communication time This cost can be computed as the total number of bits transmitted during the protocol execution. We suppose k is the length of each encrypted codeword in the domain of encryption function f and k' is the length of the encryption function E.

Step 1: No communication time

Step 2: $m \cdot k$

$m \cdot k$ for sending $f_{k_r}(v^{'})$ to m data owners

Step 3: $3n \cdot k + (n+m) \cdot k^{'}$

for each data owner O_i, $1 \leq i \leq m$:

3.1 - 3.4 : No communication time

3.5 : $3n \cdot k + (n+m) \cdot k^{'}$

$(|Y_{i,A}| + |Y_{i,B}| + |Y^{'}_{i,A'}|) \cdot k + (|Y^{'}_{i,C'}| + 1) \cdot k^{'} = (\frac{3n}{m}) \cdot k + (\frac{n}{m} + 1) \cdot k^{'}$ and accordingly $3n \cdot k + (n+m) \cdot k^{'}$ for m data owners

Step 4, 5: $m \cdot k$

$m \cdot k$ for sending α from user to the initiator

Step 6: $(m-1)[(\frac{3n+ns}{s}) \cdot k + (\frac{ns+n}{2s} + 1) \cdot k^{'}]$

for each data owner O_i, $1 \leq i \leq m$:

6.1 No communication cost

6.2 $(|\bar{Y}_{i-1,\bar{A},\bar{B}}| + |Y^{'}_{i-1,A'}| + |R^{i-1}_{i-1,\bar{A},\bar{B}}|) \cdot k + (|Y^{'}_{i-1,C'}| + |R^{i-1}_{i-1,C'}| + 1) \cdot k^{'} = m \sum_{i=1}^{m-1} [(\frac{3n+ns}{ms}) \cdot k + (\frac{n}{m} + \frac{n}{2ms} + 1) \cdot k^{'}]$ for m data owners.

Step 7: $(m-1)[(\frac{3n+2s}{ms}) \cdot k + (\frac{2ns+n+2ms}{2ms}) \cdot k^{'}]$

$(|\bar{Y}_{i-1,\bar{A},\bar{B}}| + |Y^{'}_{i-1,A'}| + |R^{i-1}_{i-1,\bar{A},\bar{B}}|) \cdot k + (|Y^{'}_{i-1,C'}| + |R^{i-1}_{i-1,C'}| + 1) \cdot k^{'} = (m-1)[(\frac{2n}{ms} + \frac{n}{m} + \frac{n}{ms}) \cdot k + (\frac{n}{m} + \frac{n}{2ms} + 1) \cdot k^{'}]$ for sending shares that have been encrypted with all keys from $m-1$ data owners to O_2.

Step 8: $(\frac{n}{s} + n) \cdot k + (\frac{n}{2s} + \frac{n}{2}) \cdot k^{'}$

$(\bigcup_{i=1}^{m} |R^{i}_{i,\bar{A},\bar{B}}| \cdot k + \bigcup_{i=1}^{m} |R^{i}_{i,C'}| \cdot k^{'}) + (|\bar{R}_{\bar{A},\bar{B}}| \cdot k + |\bar{R}_{C'}| \cdot k^{'})$ for sending partial results and $\bar{R}$ to the user. With the assumption that $\frac{n}{2}$ tuples belong to $\bar{R}$, the total cost is $(\frac{n}{s} + n) \cdot k + (\frac{n}{2s} + \frac{n}{2}) \cdot k^{'}$

Step 9: $(m+1) \cdot k$

$(m+1) \cdot k$ for encrypting user query with all keys along the ring.

Step 10: No communication cost

Again, the total communication cost is $O(mn)$

3.6.5 Experimental evaluations

We performed experimental procedure to clearly illustrate the benefit of bucketization approach for processing equi-join with selection condition in SMC paradigm in terms of communication time. We use Castalia simulator on a Linux machine with dual Intel CPU running at 2.26 GHz, and 2GB Ram. In the experiments, we construct 5 nodes in Castalia simulator including 3 data owners, 1 user and the TTP. Each data owner holds two tables with join and extra attributes. To encrypt the values of each owner's table, we implemented a simple *commutative encryption* protocol based on *exponentiation modulo p*.

For running the protocol, we prepared a set of values for selection condition uniformly. We measured the communication time of our protocol for different number of buckets and different number of records, which are partitioned horizontally among 3 nodes. The experimental results are shown in Figure 5.5.

Each point in the figure corresponds to the average communication time obtained by running the query workload Q consisting 8 queries on searchable attribute. The x axis shows the number of tuples N_i partitioned among data owners, where $N_i = 18.10^{i-1}$ for $1 \leq i \leq 4$. The y axis shows the average time of the query workload on the data owner side, which is dominated by the time of circulating buckets containing range values along the ring.

Even in this experiment, the effect of bucketization in reducing communication time is significant, almost halving the communication time when passing from 1 bucket to 5 buckets. Moreover, as we can see in the figure, the number of buckets cannot be increased arbitrarily, because further increasing may not lead to significant better results, e.g. from $s = 10$ to $s = 15$.

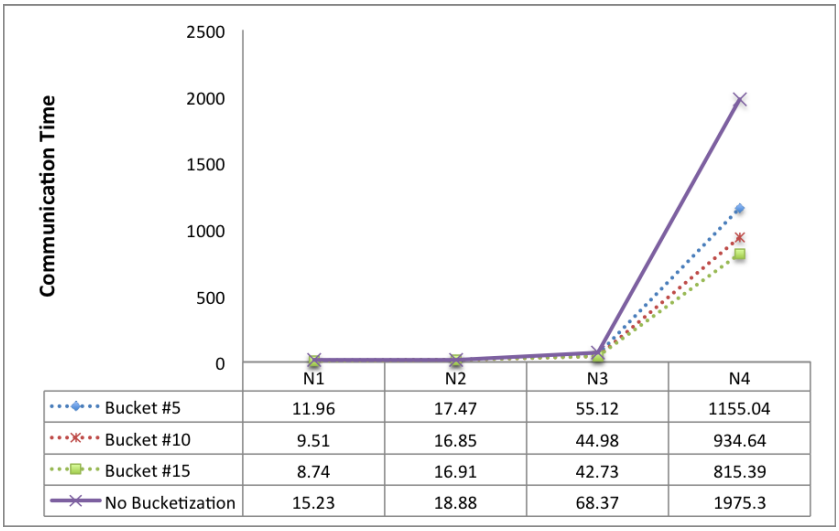


Figure 3.13: Communication Time vs Number of Buckets

3.7 Chapter summary

We studied the problem of securely SQL^{--} queries over data portioned among multiple data owners in SMC Paradigm. Our proposed protocol is based on bucketization technique to reduce the computation and communication cost. Unlike existing approaches, the protocol is designed to work with more than two parties, and the experimental tests on randomly generated data showed that the protocol can scale well to large size data.

Chapter 4

Encrypted Data Outsourcing

This chapter includes an overview of some known techniques related to privacy-preserving query processing (P^3Q) in the area of encrypted data outsourcing (EDO). We start with discussing the importance of data outsourcing (DO) paradigm in research community, and then present five categories of privacy-preserving searchable techniques over EDO paradigm. The literature of this problem is vast and we just describe one or two techniques for each category to perform the query language we defined, $SQL^{--} = \{$equality test, range, equi-join$\}$; finally we compare them with respect to the security and efficiency requirements.

4.1 Introduction

Today, data outsourcing has received more attention from organizations needing to manage large amount of data due to the growing cost of in-house storage and data management. This possibility allows them to save the cost of buying expensive hardware and software, and more importantly provides higher availability. However, despite of all benefits that DO inevitable provides, many organizations are reluctant to outsource their data due to the fear of loosing control and/or disclosing sensitive information to a service provider who is not fully trusted. This challenge can be addressed by encrypting data before the outsourcing in order to keep them hidden from the service provider [20]. Clearly, this solution poses a problem related to the query processing on encrypted data. A naive approach would be to download the whole data at organization side and perform decryption and querying. However, this solution brings more communication cost without taking full advantage of data outsourcing paradigm. To this purpose, a large number of research activities have been conduct about the problem of

querying over outsourced encrypted data considering security and efficiency requirements [30, 1, 2, 11, 12, 18, 13, 62, 27, 34, 60, 27, 62, 17, 6, 47, 56, 4]. Most of the proposed techniques associate with encrypted data indexing information [30, 18, 34, 1, 27, 6] enabling the execution of queries on encrypted data. However since indexes carry some information about the original plaintext values it may open the door to possible violations to the data. Also, index-based solutions introduce overhead in query execution due to query transformation through indexes, data decryption and filtering the result set. Hence, there has been considerable interest to perform privacy-preserving queries over encrypted data while maintaining an acceptable query processing performance. Next, we give a collection of techniques that are used to perform SQL^{--} queries on service provider with encrypted data.

The remainder of this chapter is organized as follows. Section 4.2 states background on data outsourcing paradigm. Section 4.2.1 describes the main techniques proposed in the literature for performing queries over encrypted data. Section 4.2.2 focuses on comparing some of the proposed techniques based on some security requirements including data and query privacy. Section 4.2.3 presents some factors that affect the performance. Finally, Section 4.3 concludes the chapter.

4.2 Background

In this section, we describe the main issues that need to be addressed for the problem of query processing on outsourced encrypted data in a privacy way. First, we present the data outsourcing model, then we describe how data are organized in the outsourced database context, and what privacy issues are raised by outsourcing. We then outline the proposed solutions to support SQL^{--}, processing queries on encrypted data.

System model

Techniques that have been investigated by a large number of research activities comply with **D**atabase **A**s a **S**ervice (DAS) model [30] to process various types of queries. This model is composed of four entities (Figure 4.1):

- data owner: A person or organization who produces and outsources encrypted data to a server with more storage capabilities as providing some additional information called metadata to client side for the purpose of query transformation;

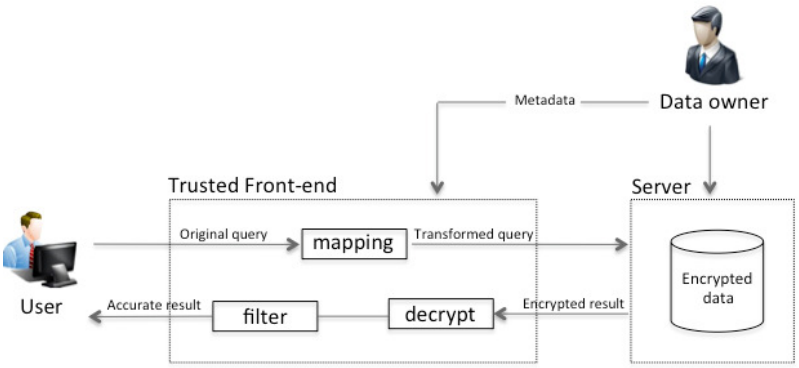


Figure 4.1: DAS scenario

- client: A trusted front-end entity that translates the query posed by user into equivalent one for processing desired operations on encrypted data at server side;
- user: An entity that requires to access encrypted data stored at server by querying through a client front-end entity;
- server: An external service provider that returns the response of query to the client who in turn decrypts and returns the result to the user.

In this scenario, the server is assumed to be *honest-but-curious*, which roughly means that even though the server manages the outsourced encrypted data honestly, he can try to learn extra information by analyzing the transcripts of messages received during the query execution. Moreover, for the sake of simplicity, it is assumed that outsourced encrypted data are stored within a relational database management (DBMS), where data are organized in tables. Note that the DBMS running at the server is not allowed to decrypt the data for the purpose of query execution, therefore a post-processing phase is needed by the trusted front-end to decrypt the query evaluated by the server and to filter out the results that do not satisfy the user query condition (false positives).

Data organization

As a matter of fact, each data tuple (record or row) is stored at the server in encrypted form according to the selected encryption algorithms. In principle, there are two well known classes of encryption algorithms: symmetric(e.g., AES) [53] and asymmetric(e.g., RSA) [58]. Due to the superior performance of symmetric encryption over asymmetric encryption, most tech-

niques are based on symmetric key encryption for encrypting data before outsourcing them to the server. The granularity of data encryption can be applied at various levels according to when and how the data need to be accessed [38]:

1. attribute value: the smallest granularity in which each attribute value of a plaintext tuple is encrypted separately;

2. column: a more selective approach where certain columns (attributes) in the plaintext relation are encrypted;

3. tuple: each tuple in the plaintext relation is encrypted as a single value in the encrypted relation;

4. relation: each relation is represented as an encrypted value so that tuples and attributes are indistinguishable.

Note that different levels of data encryption can affect performance in terms of communication cost and client workload. For instance, encryption at attribute and relational level imply the communication to the requesting client as it is not possible to extract any subset of the tuples in the encrypted relation, and attribute value level encryption requires an excessive workload for the client in decrypting data. Therefore, most proposals adopt tuple encryption level to keep the communication and workload low.

Problem statement

Given a query $q \in SQL^{--}$ over outsourced encrypted data by an authorized user, the aim is to obtain the answer while satisfying the following properties:

1. data privacy: a security requirement that protects data owner from misuse of data by the client and server i.e., client is not allowed to get more information than what h/she is querying on the server, and encrypted data may not be decrypted at server side.

2. query privacy: a security notation on the amount of information leakage to the server according to the submitted queries by user.

3. query anonymous result: the user does not know whom the result belongs to. This property can be considered only in the multi-owner setting.

Overall, the research community aims to protect the user and data privacy as performing queries over encrypted data while still maintaining an acceptable query processing performance. Accordingly, next section discusses

various techniques and their weaknesses due to the security and efficiency requirements in addressing SQL^{--} queries over outsourced encrypted data.

4.2.1 Querying over outsourced encrypted data

The problem of querying over encrypted data has been deeply studied in the literature [30, 1, 2, 11, 12, 18, 13, 62, 27, 34, 60]. There are essentially five categories of solutions that have been developed recently for this problem: (*i*) bucketization-based techniques [30, 34], which present an effective approach based on indexes to make basic operations on encrypted data while trying to keep the information disclosure as minimum as possible; (*ii*) order-preserving encryption based techniques [2, 9], which guarantee that the order amongst plaintext data is presented in the ciphertext domain. This approach is suitable for evaluating range queries since it allows direct translation of range predicates from the original domain to the domain of the ciphertext; (*iii*) techniques that adopt some specialized data structure while trying to preserve notions of semantic security of the encrypted representation [18, 11, 60]; (*iv*) (fully) homomorphic encryption based techniques [26, 31] that allow server to compute specific operations on encrypted data in an efficient way; and (*v*) keyword search based techniques [27, 62, 17, 6, 47, 56, 4] that allow server to identify encrypted files containing specific keywords. To illustrate the known methods for each category, let us consider a scenario where a data owner O outsources sensitive piece of employee information $EMP = (eid, age)$ and $DEPT = (did, eid)$ to the server. Specifically, our aim is to discuss the queries in SQL^{--} that are feasible under these methods and security implications that arise when some of queries are performed.

Techniques based on bucketization

The bucketization-based technique was first introduced by Hacigümüs et al. [30] and it has been investigated as well as improved by many researches thereafter [34]. Their proposed approach involves partitioning the domain of attributes considered sensitive into a number of non-overlapping subsets of values called buckets. For each bucket, a unique random ID is defined that represents the partition to which the unencrypted value belongs. Then, the bucket IDs along with the encrypted tuples are outsourced to the server. Referring to the example above, owner O maps each tuple $t_i = \langle eid : x_i, age : y_i \rangle$ of relation EMP to a new tuple $t_i^{'} = \langle eid^{'} : B_{x_i}, age^{'} : B_{y_i}, etuple_i \rangle$ that belongs to encrypted relation $EMP^{'}$, where x and y refer to plaintext values of attributes eid and age respectively, B stands for the bucket ID,

and *etuple* represents an encrypted string corresponding to tuple t_i. Now if an authorized user u specifies query $q : \pi_{EMP_{eid}}(\sigma_{age.y=35}(EMP))$ and the bucketization scheme contains a bucket ID 2 representing the range 20 to 40, then the client transforms the query q to $q^{'} : \pi_{EMP^{'}_{eid'}}(\sigma_{age'.B_y=2}(EMP))$. However, this method does not support range queries efficiently because the domain of bucket IDs does not necessarily preserve the plaintext domain ordering. For example, if user asks for a query $q : \pi_{EMP_{eid}}(\sigma_{age.y>62}(EMP))$ and the bucketization scheme contains bucket IDs 2, 7, 5, 1, and 4 representing the ranges $[0-20], (20-40], (40-60], (60-80]$ and $(80-100]$ respectively, then the client maps the query to $q^{'} : \pi_{EMP^{'}_{eid'}}(\sigma_{age'.B_y>1}(EMP))$, which returns the whole relation $EMP^{'}$. Processing equi-join query is feasible in this method since equal values are indexed with equal bucket IDs, but it provides a great exposure to possible privacy violations at server side. For better privacy, the server should not be able to join columns for which the client did not request a join. Clearly, the final results provided by this method are not accurate and needs filtering to weed out any tuples that do not satisfy the the original query conditions (false positives).

Following Hacigümüs's technique, Hore et al. [34] addressed the problem of computing range queries to minimize the number of false positive results. They studied some analysis to compute the optimal buckets for numeric domain attributes by looking at data distribution.

Techniques based on specialized data structures

Damiani et al. [18] proposed a B^+-tree based indexing method, which allows the evaluation of both equality and range queries. In principle, their scheme can be adopted for each attribute to efficiently support range queries on encrypted data. First, a B^+-tree is built over plaintext values of searchable attributes at client site, and then corresponding to each node a pair $(id, enode)$ is outsourced to the server with id the node identifier[1] and $enode$ the content of node composed of $\langle l\_id, key, r\_id\rangle$ in encrypted form using a deterministic encryption that generates the same ciphertext for the same plaintext where l_id and r_id refer to the left and right identifiers of the key, respectively.

To process a range query $q : \pi_{EMP_{eid}}(\sigma_{40\leq age.y\leq 50}(EMP))$ following this approach, a $B^+ - tree$ is built on searchable attribute age at client side as shown in Figure 4.2, and then: (a) the client starts with root node and sends $id = 0$ to the server; (b) the server returns back the $enode$ corresponding to Id 0; and (c) the client decrypts the $enode$ and compares 40 with the

[1]This id is assigned automatically by system each time the node is inserted.

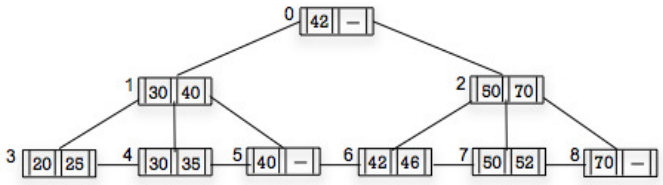


Figure 4.2: $B^+ - tree$ indexing over searchable attribute *age*

key, if $40 < key$, it takes *l_id* and traverses the left branch; otherwise it takes the right node. This procedure repeats until the leaf with *age* 40 is found, then the client navigates the leaves to reach the *age* 50. The final result of this type of indexing search is accurate, but since for each search the client has to traverse the tree by accessing a sequence of some nodes, the performance decreases due to to the high communication between the client and the server.

Shi et al. [60] presented a privacy-preserving searchable scheme to support multidimensional range queries over encrypted data (MRQED). In this method, each attribute is encoded using discrete integer values 1 through T, where T is the number of possible values each attribute may assume. Then a binary interval tree $tr(T)$ is built over integers 1 from T where each node i with a pre-defined identity x_i represents a range $cv(x_i)$ that corresponds to its descendants. Generally speaking, a range $[s,t] \in [1,T]$ is represented by a collection of nodes $\Lambda(s,t) \in tr(T)$, where $\Lambda(s,t)$ is the smallest of all subsets $V \in tr(T)$ such that $\bigcup_{x_i \in V;\, 1 \leq x_i \leq 2T-1} cv(x_i) = [s,t]$. To encrypt a message Msg under point p, Msg encrypts under identities $X_p = \{x_i$:node i is on the path from the root to the leaf node representing p$\}$; and to decrypt the message Msg, the keys for all x in $\Lambda(s,t)$ are released. If $p \in [s,t]$ then $X_p \cap \Lambda(s,t) = 0$. As an example, consider a 1-dimensional range query ($MRQED^1$) with domain $D = [1,8]$ for attribute *age* in relation EMP. Figure 4.3(a) shows a path from the leaf node representing $p \in [T]$ to the root. $X_p = \{x_1, x_2, x_4, x_9\}$. Figure 4.3(b) shows a ciphertext C encrypted under the point p, which composed of $\{C_1, \ldots, C_l\}$ where $l = O(log\ T)$. On the right, the decryption keys for $[2,5]$ consists of three sub-keys K_A, K_B, K_C corresponding to nodes $\Lambda(2,5) = \{x_9, x_5, x_{12}\}$, respectively have been shown. Though this scheme provides provably strong security, it is not efficient as for a single search request; a full scan over the whole encrypted relation is required by the server.

Boneh et al. [12] proposed a public key encryption primitive called Hidden Vector Encryption (HVE) [55], which supports equality and range queries.

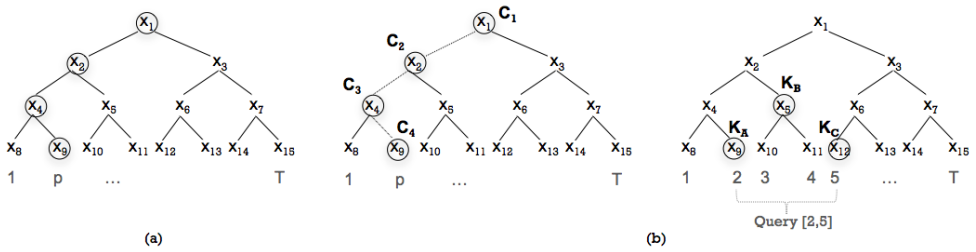


Figure 4.3: An $MRQED^1$ Scheme (a) The path from a leaf node representing $p \in [T]$ to the root (b) Encryption under the point $p = 2$ and the keys released for the range [2,5]

However, the complexity of range search per data record is linear with the range size T, which is far from efficient when the range is large. Further, their technique does not admit any form of indexing for faster access that could be prohibitively expensive for large tables.

Techniques based on order-preserving

Order-preserving encryption schemes (OPES) are presented to support queries with comparison conditions since the current encryption techniques do not preserve order and therefore an entire table scan would be needed to evaluate such kind of queries. In [2], Agrawal et al. presented an order-preserving technique to support equality and range queries over encrypted data. Their approach is applicable to numeric data and the idea is to take an input user-provided target distribution and transform the plaintext values by using an order-preserving transformation in such a way that the transformed values follow the target distribution i.e., the ith plaintext value in sorted list of plaintext values is encrypted into the ith value in sorted list of values obtained from target distribution. Although the complexity of order-preserving technique is logarithmic time by locating ciphertexts in a tree-based structure, but the security risk associated with revealing data order prevent many data owners from deploying it.

Boldyreva et al. [9] showed that order-preserving technique proposed by Agrawal et al. [2] does not satisfy all the standard notion of security because the encryption method in OPES needs to be deterministic and therefore it reveals the frequency of each distinct value in the dataset. Hence, they proposed a random order preserving function to prevent adversary from trivial distinguishability. However in both of these techniques [2, 9], an adversary (server) can improve its estimation of true value of ciphertext as the number of queries increases.

Following [2], Wang et al. [65] considered a model of exact occurrence frequencies of domain values as the background knowledge possessed by the adversary. Their technique known as Order Preserving Encryption with Splitting and Scaling (OPESS) is an evaluation of OPES, which adopts $B-tree$ data structure on index values to support equality and range queries. Without loss of generality, OPESS maps each value in the searchable attribute domain to multiple ciphertext values such that they have similar number of occurrences exploiting techniques called splitting and scaling. Let's consider attribute age in relation EMP with k distinct values $\{v_1, \ldots, v_k\}$ whose number of occurrences are $\{f_1, \ldots, f_k\}$ and $v_1 < v_2 < \ldots < v_k$. During the splitting technique, each value v_i is mapped to a compound data value using three consecutive positive integers $m-1$, m, $m+1$ such that the corresponding f_i can be expressed in a form $f_i = c_1(m-1) + c_2(m) + c_3(m+1)$, where c_1, c_2, and c_3 are non negative integer values (see Figure 4.4). Moreover, an order preserving function such as was proposed by [2] is used to preserve the order of index values with respect to the original domain of searchable attribute i.e., for any two values $v_k < v_l$ and for any index values i_k and i_l associated with v_k and v_l respectively, it is necessary that $i_k < i_l$. However, since after splitting the sum of frequencies of indexes representing value v_i is equal to the original frequency of v_i, an attacker with an exact knowledge of occurrence frequencies of values can exploit this property to break the indexing method. Hence, a scaling technique is used to defend against such an attack. When a value v_i is split into n index values $i_1, \ldots, i_n$, each index entry in $B-tree$ corresponding to i_k is replicated s_k times, where s_k is a randomly chosen scale factor. Consequently, the index frequency distribution is not uniform anymore. While the final results of this approach is accurate, the splitting and scaling techniques is expensive due to the fact that the same plaintext value is mapped into different index values and both splitting and scaling techniques need to be inverted for query evaluation [24].

Techniques based on fully homomorphic encryption (FHE)

Recently, Wang et al. [66] presented a technique based on fully homomorphic encryption that can support general database queries such as equality, range and join over encrypted data. Fully homomorphic encryption is a kind of homomorphic encryption scheme that can handle all functions, represented by addition and multiplication[2]. They showed that query predicates can

[2]Note that not all homomorphic encryption scheme can support both additions and multiplications on ciphertexts, e.g. the Paillier cryptosystem [54] cannot perform multi-

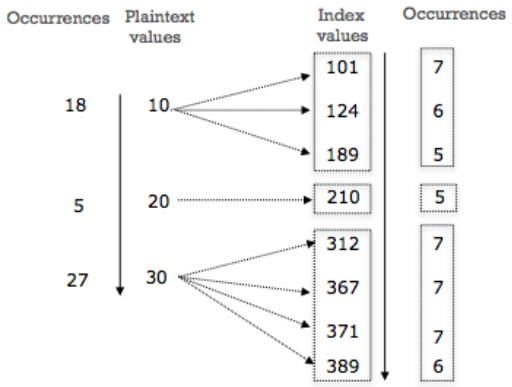


Figure 4.4: An example of splitting with $m = 6$ for searchable attribute *age* with values $\{10, 20, 30\}$ and frequencies $\{18, 5, 27\}$, respectively

be represented as a circuit constructing from XOR, AND, NOT, and OR exploiting some properties described as follows, resulting computations on ciphertexts directly without the need for decryption. Let bx and by be the binary representation of values x and y in a finite integer field, we define: (P_1) *Integer Addition.* $Enc(x) + Enc(y) = Enc(x + y)$; (P_2) *Integer Multiplication.* $Enc(x) * Enc(y) = Enc(x * y)$; $(P_3.A)$ *Integer Subtraction.* $Enc(x) - Enc(y) = Enc(x - y)$; $(P_3.B)$ *Binary Subtraction.* $Enc(bx) - Enc(by) = Enc(bx - by)$; (P_4) *Bitwise AND.* $Enc(bx) \wedge Enc(by) = Enc(bx \wedge by)$; (P_5) *Bitwise OR.* $Enc(bx) \vee Enc(by) = Enc(bx \vee by)$; and (P_6) *Bitwise NOT.* $Enc(1) \oplus [Enc(bx)]_i = Enc(\overline{[bx]_i})$.

Now consider relation EMP with searchable attribute *age* in which an attribute $age.v$ has its binary representation as an $(n+1)$-bit string $age.v_1$ with the first leftmost bit as the sign bit. Corresponding to each value of attribute *age* a pair $(Enc(v), Enc(v_1))$ is outsourced to the server in which $Enc(v_1)$ is a bit-wise encryption of v_1 and consists of $(n+1)$ ciphertexts $[Enc(v_1)]_j$ corresponding to bit $[v_1]_j$ $(1 \leq j \leq n+1)$. Consequently, for evaluating an equality query condition $(\sigma_{age.v=35}(EMP))$, the binary circuit $\bigwedge_{i=2}^{n+1}(Enc(1) \oplus (Enc(v_{1_j})]_i - [Enc(z)]_i))$ where z is the binary representation of 35 is constructed. For each value v of attribute *age*, the circuit outputs $Enc(1)$ if $age.v = 35$; otherwise $Enc(0)$. Similarly, for evaluating range query condition $(\sigma_{age.v<35}(EMP))$, the circuit is constructed as $[Enc(v_{1_j})]_1 - [Enc(z)]_1$. Then, for each value v of attribute *age*, if $age.v < 35$ holds, the value (age.v-35) should be negative and the sign bit (first bit) of its binary representation should be 1, thus $[Enc(v_{1_j})]_1 - [Enc(z)]_1$

plications in ciphertexts.

should be $Enc(1)$. On the other hand, for evaluating query condition $(\sigma_{age.v>35}(EMP))$, we have $Enc(1) \oplus [Enc(v_{1_j})]_1 - [Enc(z)]_1$, which returns $Enc(0)$ if query condition holds. Clearly, for processing equi-join condition $\sigma_{EMP_{eid.v}=DEPT_{did.v'}}(EMP \bowtie DEPT))$ for each value v and $v^{'}$ of attributes *eid* and *did* respectively, the server needs to perform $N \times N$ evaluations to check if $EMP_{eid.v} - DEPT_{did.v'} == 0$, where N is the number of tuples in relations EMP and $DEPT$. This can be achieved by constructing an equality condition circuit as mentioned above. Although this technique achieves strong privacy compared to previous methods [30, 2, 34], the performance according to the expensive homomorphic encryption scheme and binary circuits construction for presenting predicates for each type of queries is high.

Techniques based on keyword search

Many searchable techniques have been proposed on the basis of keyword search. The challenge is to support applications that allow users to search for documents that contain a given word. We categorize the state-of-the-art as follows:

- **single-user searchable encryption.** Song et al. [62] study the problem of keyword search on encrypted data using symmetric encryption (SE) where document is encrypted word by word and outsourced to the server. Their scheme reveals the frequency of each word. Following Song et al., Goh [27] proposes an index based scheme for keyword search over encrypted data using Bloom filters [8], which is an efficient way to store information about the existence of a record in a database. To prevent statistical analysis attacks, each Bloom filter is randomized using a unique identifier, which in turn introduce false positives.

- **multi-user searchable encryption.** In multi-user setting, each user may have access to a different set of documents stored at server. Curtmola et a.l [17] propose a multi-user searchable encryption approach by extending a single-user scheme; sharing secret keys among all users. Their approach is based on symmetric key encryption, and the server is enabled to search for the documents that contain a particular keyword without revealing information about the contents of documents and queries. While their approach provides faster retrieval time, it has two main limitations in practical applications: first, multiple users are not able to write database, and second user revocation affects all non-revoked users. To alleviate these drawbacks, Bao et al. [6] and Dong et

al. [21] propose a keyword search on encrypted data allowing a group of users to share data in a way that all users are able to write and search without sharing their secrets. Their method is scalable for large systems where user revocation may occur frequently. The solution given by Dong et al. [21] is more efficient to support keyword search, enabling the server to return only the encrypted data that satisfies an encryption query without decrypting it. The proposed method by Dong et al. [21] is based on proxy re-encryption scheme, which allows the proxy server to transform a ciphertext for Alice to another ciphertext of the same plaintext for Bob.

The proposed schemes in [17, 6, 21] only support equality queries. Asghar et al. [4] extend the work proposed by Dong et al. [21] to support complex queries such as numerical inequalities ($<, \leq, >, \geq$). Their scheme is based on two variant of encryption scheme including keyword encryption for supporting equality test and proxy re-encryption for protecting data in outsourced environment.

- **multi-key searchable encryption.** Multi-key searchable encryption enables search over data encrypted under different keys by multiple data owners. The seminal work of Gentry [26] designed a multi-key searchable scheme based on fully homomorphic encryption (FHE), which allows users to perform computations on encrypted data without having the secret decryption key. In this scheme, multiple data owners jointly run a multi-party computation (MPC) protocol to generate a joint public key and then outsource their data under the joint public key to the server who then uses FHE for query computation. Finally, the owners run another MPC protocol to recover the result. Nevertheless, this scheme is not efficient because its communication and computational costs increase due to the MPC protocol adopted to find common public key. Moreover, this scheme is not practical in our scenario where data owners do not trust each other to encrypt their data using a common key. As further related work, López-Alt et al. [47] proposed a multi-key FHE scheme, which allows owners to outsource their data using their own keys. Then, the server computes query processing adopting FHE scheme. Note that this scheme requires the owners to jointly decrypt the query result through a MPC protocol.

  Recently, Popa et al. [56] proposed a cryptographic scheme that allows server to search for a keyword posing by a user in documents encrypted with different keys while hiding the content of documents

Table 4.1: Data outsourcing techniques for supporting SQL^{--}

Method	Query			Techniques
	Equality	*Range*	Join	
Hacigümüs et al. [30]	•	○	•	Bucketization
Hore et al. [34]	•	•	–	Optimal bucketization algorithm
Damiani et al. [18]	•	•	–	$B^+ - tree$
Shi et al. [60]	•	•	–	Binary interval tree
Boneh et al. [12]	•	•	–	HVE
Boldyreva et al. [9], Agrawal et al. [2]	•	•	–	OPES
Wang et al. [65]	•	•	–	OPESS (B-tree)
Popa et al. [57]	•	•	•	FHE, OPE,JOIN-ADJ
Wang et al. [66]	•	•	•	FHE(Circuit-based)
Song et al. [62]	•	○	–	Symmetric encryption
Goh [27]	•	○	–	Bloom filters
Curtmola et al. [17]	•	○	–	Symmetric encryption
Bao et al. [6]	•	○	–	Index-based keyword search
Dong et al. [21]	•	○	–	Proxy re-encryption
Asghar et al. [4]	•	•	–	Proxy re-encryption
Popa et al. [56]	•	○	–	Multi-key search using access graph
López-Alt et al. [47]	•	○	–	FHE, MPC

• supported; ○ not supported; – not solved yet

and the search word. In this scheme, each user outsources a document encrypted with its own key and gives access to other users by giving the decryption key through an access graph. Consider a scenario where user u outsources n documents encrypted under k_j for $1 \leq j \leq n$. In order to search for a word w over all documents, u computes a token for w using its own key k_u and sends it along with a piece of information to the server. Then the server use the information to convert a search token under user's key to a search token under $k_1, \ldots, k_n$, matching token against words encrypted with $k_1, \ldots, k_n$. The advantage of this scheme is that there is no trusted party for choosing keys or providing access to documents.

Other techniques such as CryptDB proposed by Popa et al. [57] performs different types of database queries using layers of different encryption schemes. For instance, deterministic encryption for equality query, order-preserving for range queries, and a new cryptographic primitive called JOIN-ADJ (adjustable join) for equi-join queries. The system relies on a trusted proxy server that encrypts each query condition with the best suited encryption scheme as mentioned above and sends them to the DBMS server who executes them using standard SQL and returns the encrypted query result to the proxy for decryption. The granularity in which CryptDB allows the DBMS to perform queries is an entire column.

Table 4.1 summarizes the discussion by showing for each technique previously presented, which indexing method and which queries in SQL^{--} are supported.

Table 4.2: Comparison data outsourcing techniques due to the key and user management

Method	Query			
	Single-user	*Single-key*	*Multi-user*	*Multi-key*
Hacigümüs et al. [30]	•	•	○	○
Hore et al. [34]	•	•	○	○
Damiani et al. [18]	•	•	○	○
Shi et al. [60]	•	•	○	○
Boneh et al. [12]	•	•	○	○
Boldyreva et al. [9], Agrawal et al. [2]	•	•	○	○
Wang et al. [65]	•	•	○	○
Popa et al. [57]	•	•	○	○
Wang et al. [66]	•	•	○	○
Song et al. [62]	•	•	○	○
Goh [27]	•	•	○	○
Curtmola et al. [17]	•	•	•	○
Bao et al. [6]	○	•	•	○
Dong et al. [21]	○	•	•	○
Asghar et al. [4]	○	•	•	○
Popa et al. [56]	○	○	•	•
López-Alt et al. [47]	○	○	•	•

Table 4.2 categorizes the proposed techniques based on the key and user management they support. Most solutions focused on single-key search scheme when the data is encrypted with the same key. Following, we analyze the security and performance of single-key approaches in single-user setting.

4.2.2 Security analysis

In this section, we discuss the privacy of the mentioned techniques in supporting SQL^{--} queries.

The scheme presented by Hacigümüs et al. [30] does not preserve data and query privacy. By the fact that equal values in the plaintext domain are also equal in the encrypted domain, the server can estimate the number of different values. Moreover, when the server observes two equal query conditions $q_1^{'}$ and $q_2^{'}$ on outsourced data, h/she infers that the corresponding user query conditions q_1 and q_2 respectively are the same or lie in the same bucket ID. This privacy concern is more crucial when a fine partitioning is used for bucketization.

The bucket-based indexing method proposed by Hore et al. [34] preserves data privacy partially adopting a controlled diffusion algorithm that lets data owner fine-tune bucketization by sacrificing the accuracy. However it still does not satisfy the query privacy requirement.

The method proposed by Damiani et al. [18] has better security level than [30], however it does not meet data and query privacy since during the query processing, both the client and the server will get some information

which is not related to the user query. For instance, consider the example mentioned in Section 4.2.1 with a range query $q : \pi_{EMP_{eid}}(\sigma_{40 \leq age.y \leq 50}(EMP))$. In order to process q, the client will access nodes 0, 1, and 5, which reveals that there are at least two other ages 42 and 40 in the database (see Figure 4.2). Accordingly, the server will realize that the user was accessing nodes 0,1, and 5, and node 0 is the root, node 1 is an internal node, and node 5 is a leaf node of the tree. Using such information collected gradually together with some mining techniques allows the server to rebuild the whole tree and therefore infer sensitive information from the encrypted data. Moreover, the server is able to learn and guess the distribution of a domain data if the domain is highly skewed (e.g., day of month). To handle some limitations in such a case, more discussion can be found in [44, 19].

In the scheme proposed by Shi [60], the server can learn the range a client is querying. This is because encryption is public key based and therefore the server encrypts messages under various attribute values by launching dictionary attack. Opposed to Shi et al. the encryption scheme proposed by Boneh et al. [12] hides the resulting data items in matching query (access pattern), thus it is better for query privacy. In both techniques, the server can improve his estimation of the true value of ciphertexts by observing more queries. Hence, both techniques reveal at least partially query to the server.

In Agawal's order-preserving method [2], if the server observes the encrypted data, h/she will realize the order of plaintexts. Moreover, since the encryption method is deterministic, duplicates can be used to guess the distribution of attribute domain, particularly if the distribution is highly skewed (e.g., day of month). This concern may be improved by appropriately choosing target distribution, the adversary can be forced to make large estimation errors. Both methods proposed by [2, 9] are vulnerable to "value-localization" problem i.e., the server can improve its estimation of the original value of a ciphertext as the number of queries increases [33]. For instance, if the same result set is returned for queries $q_1 = [20, 50]$ and $q_2 = [30, 80]$, the server infers that the result set contains values between 30 to 50. According to Huiqi et al. [67], order-preserving scheme does not address query privacy.

In splitting and scaling method of [65], data values with multiple and unique occurrences are both translated in the same fashion and accordingly the server is not able to reconstruct the correspondence between plaintext and index values.

The scheme presented by Wang et al. [66] is semantically secure because it is based on homomorphic encryption and therefore the server is not able

Table 4.3: Privacy requirements achieved by the proposed methods

Method	Privacy requirements	
	Data privacy	Query privacy
Hacigümüs et al. [30]	No	No
Hore et al. [34]	Partially	No
Damiani et al. [18]	No	No
Shi et al. [60]	Partially	No
Boneh et al. [12]	Partially	Partially
Boldyreva et al. [9], Agrawal et al. [2]	Partially	No
Wang et al. [65]	Yes	Partially
Popa et al. [57]	Partially	Partially
Wang et al. [66]	Yes	Partially

to see the contents of outsourced data. The only information revealed are the attributes with the query condition and the type of query, however this information does not give any detail about the original contents of the query or about the underlying data encryption.

CryptDB proposed by [57] reveals duplicate values in the column in equality query and order of elements in the column in the case of range query. With respect to the join queries, the server can not infer join relations among groups of columns that were not requested, and the scheme does not reveal pliantext.

Table 4.3 summarizes which privacy requirements are satisfied by each of the previously proposed schemes. As Table 4.3 shows, none of the solutions provide all the properties at the same time.

4.2.3 Efficiency analysis

In this section we discuss issues related to the performance of the surveyed schemes. The efficiency in the context here can be interpreted in terms of communication and computation costs.

Computation cost can be measured as the number of encryption and decryption operations required at owner and client sides. At owner side, the computation cost depends on the size of sensitive data needed to be outsourced and accordingly the type of encryption scheme. For instance, fully homomorphic encryption is prohibitively slow. At client side, the computation cost is the number of encryption and decryption operations needed for encryption query conditions posted by user and decrypting the server result. Clearly, this cost depends on the indexing method. More precise indexes provide more accurate results in terms of false positives. The total number of false positives (TFP), where all queries are equiprobable can be

computed as

$$TFP = \sum_{\forall q \in Q} (|R^{'}| - |R|)$$

where q is a random query drawn from set Q of all queries over a relation, R and $R^{'}$ are the set of tuples, which satisfy the query conditions q and $q^{'}$, respectively. It is clear that in bucketization-based technique [30, 34], this factor is inversely proportional to the number of partitions. If the partitions are smaller, the more information is leaked but the server needs to send less false positives.

Communication cost can be computed as the total number of bits transmitted between DAS entities: owner-server, server-client and, client-user. Obviously, false positives affects the communication cost, large number of false positives causing additional communication cost. Specifically, the protocol given by Damiani et al. [18] has more communication cost during protocol execution between the client and the server. Therefore, one possible modification is to assign a new id based on "in-order" $B^+ - tree$ to each node. Then, the id along with the encrypted tuples are outsourced to the server who builds a local $B^+ - tree$ for each query processing. While this solution reduces communication time during the query processing, but leaks the order of plaintext for relations with one searchable attribute.

4.3 Chapter summary

Data outsourcing is an interesting area in data management that introduce many research challenges. In this chapter, we first presented some preliminaries used in data outsourcing paradigm and after that we reviewed some techniques for privacy-preserving query processing over encrypted data. Finally we listed the the surveyed techniques and compared them by considering some security requirements and efficiency. As Table 4.3 shows and also according to Curtmola et al. [17], most of searchable encryption schemes inevitably reveals certain query traces to the server, unless using private information retrieval techniques (PIR) [15]. We refer interested readers to [15, 46] for various discussions on PIR. Moreover, all the solutions are considered in single-owner setting and therefore "*query anonymous result*" property is not addressable here. Our goal in next chapter is to propose searchable encryption techniques for supporting SQL^{--} queries in multi-key setting where a set of data owners with different keys outsource their data in encrypted form to a cloud, providing access to encrypted data for a group of authorized users.

Chapter 5

Querying by Encrypted Data Outsourcing

In today's collaborative environment there is an increasing need for privacy-preserving query processing over data held by multiple data owners. For this reason, data owners may delegate their private data in encrypted form to an external service provider, executing queries on the entire of outsourced data encrypted with different keys (multi-key setting). This scenario has received more attention in cloud computing concept. However, a problem that should be addressed is how to handle queries over data encrypted with different keys. Motivated by this concern, in this chapter we present a novel solution based on proxy re-encryption (PRE) [7, 10, 21] scheme, limiting the adoption of expensive re-encryption by the cloud service provider. Our proposed approach efficiently executes queries over encrypted data while satisfying privacy issues such as data privacy, query privacy and query anonymous result.

5.1 Introduction

Cloud computing is one of the most popular notion in IT today. This paradigm offers scalable resources as a service over the Internet, which provides much greater flexibility than previous computing models. The benefits of cloud computing are many as for instance, data owners can outsource and store their data in the cloud exploiting data storage service [63]. On the other hand when cloud can not be trusted enough to have data in plaintext, the data privacy becomes a primary concern [36, 42]. To solve this problem, the data is encrypted by data owners before being uploaded to the cloud [39]. However, this problem poses another concern, querying over

data encrypted with different keys. Thus, an alternative solution is to allow the cloud service provider to superencrypt data with all different keys, resulting data encrypted under the same key. This solution needs data owners to follow public encryption scheme, which allows cloud to re-encrypt data with different public keys. Additionally, the front-end side in DAS model (see 4.2) transforms user query by re-encrypting it with all public keys of data owners, making it possible to search at server side. While this simplest solution is feasible in multi-key setting, it introduces heavy computation overhead of re-encryption operation in both cloud and user sides. Moreover, this solution is susceptible to brute force key search attack when the distinct number of values in domain is small. The better solution is to apply proxy re-encryption technique (PRE) [7, 10], which relies on proxy server to transform information that was encrypted under the key of owner so that it can be decrypted by the authorized user. This is done without the need for the owner to release its secret key, and more importantly the proxy server does not learn the content of the received information. PRE can be built on top of different cryptosystems, such as El Gamal [22] and RSA [58], and it has many applications such as digital rights management [61], distributed file storage system [5], law enforcement [37], encrypted email forwarding [7], and outsourced filtering of encrypted system [5].

The surveyed schemes previously proposed in Chapter 4 are not applicable here as long as they have been tailored to single-owner setting and keyword search. Hence, in the following we propose a protocol for performing SQL^{--} queries over data encrypted with different keys based on proxy re-encryption scheme. We consider the application scenario where a set of data owners outsource their data to a proxy server. Then, the proxy re-encrypts the data with its own key and sends the transformed data to the cloud service provider, who in turn partially decrypts the transformed data and returns back the results. Finally, users can decrypt the encrypted results with their own keys without the help of front end side in DAS model (see Section 4.2), resulting less communication and computation costs.

The rest of this chapter is organized into 5 sections. In Section 5.2, we introduce some definitions that we will employ in the construction of our protocol. In Section 5.3, we present protocols for SQL^{--} adopting proxy re-encryption. In Section 5.4, we evaluate the security of the equality test scheme. In section 5.5, we summarize the time complexity analysis and the the obtained result from experiment.

5.2 Preliminaries

In this section, we briefly go over some definitions that will be used to describe our approach: we describe first the proxy re-encryption scheme on which our solution is based, and then we present the system model for performing SQL^{--} queries.

5.2.1 Proxy re-encryption (PRE) scheme

There are situations in practice that an application needs to re-encrypt an encrypted data with a new key [49]. In such cases, the translation is simple if the owner of the encrypted data is online. Thus, the owner can decrypt data with its own key to obtain the original data, enabling the application to encrypt data with a new key. However, permanent online will increase the possibility of adversary attacks [48]. In 1988, Blaze et al. [7] proposed a solution to this key management problem using proxy re-encryption (PRE) scheme. They introduced a semi-trusted proxy holding a re-encryption key, which translates a message encrypted under a public key into an encryption of the same message under a different public key while proxy is not able to learn anything about the encrypted message.

El Gamal-based Proxy Encryption Scheme

In cryptography, The El Gamal encryption system [22] is an asymmetric key encryption algorithm for public key cryptography, which is based on the Diffie-Hellman key exchange. The scheme consists of three components: the key generator, the encryption algorithm, and the decryption algorithm.

- Key generation: On input 1^k, get a cyclic group $\mathbb{G}$ with a generator g such that $\mathbb{G}$ is the unique order, and two prime numbers p and q in which q divides $p-1$. Then choose a random $x \in \mathbb{Z}$ ($x \xleftarrow{R} \mathbb{Z}_q$) and computes $h = g^x$. The public key is $\langle \mathbb{G}, q, g, h \rangle$ and the private key is x.

- Encryption: On input public key $pk = \langle \mathbb{G}, q, g, h \rangle$, and a message $m \in \mathbb{G}$, choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and output the ciphertext $c(m) = \langle g^r, h^r m \rangle$.

- Decryption: On input the private key $sk = x$ and a ciphertext $c(m)$, output $h^r m.(g^r)^{-x} = g^{rx-rx}m = m$.

The proxy encryption scheme is represented by six algorithm: initialization, key generator, participant-side (owner/user) encryption, proxy-side

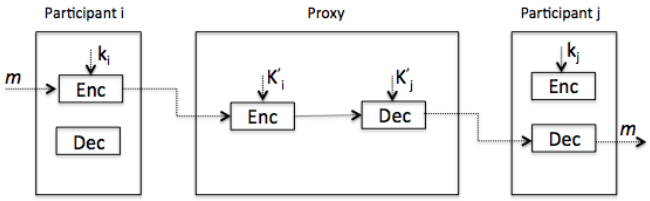


Figure 5.1: Encryption/ Decryption in the Proxy Encryption Scheme based on El Gamal ($k_i^{'}$ and $k_j^{'}$ are the proxy-side keys corresponding to the participants i and j, respectively)

re-encryption, proxy predecryption, and participant-side decryption.

- The initialization algorithm is run by a Key Administration (KA) to generate the public and master secret keys. The master secret key is stored securely by the KA while the public key parameters are shared to the participants.
- The key generator algorithm is run by the KA who receives the master keys and a participant's identity in order to reproduce two key sets for both participant and proxy sides.
- Participant-side encryption algorithm is run by the participant who encrypts a message and sends the ciphertext to the proxy.
- Proxy-side re-encryption algorithm is run by the proxy to re-encrypt the ciphertext received from the participant with proxy-side key.
- Proxy-side predecryption algorithm is run by the proxy to partially decrypt the message such that it can be encrypted only by the participant.
- Participant-side decryption algorithm is run by the participant who fully decrypts the message with its own secret key.

The proxy re-encryption based on proxy has proven to be indistinguishable under chosen plaintext attack (IND- CPA) [21].

5.2.2 System model

The model we consider for our problem is different from the DAS model as we described in Section 4.2 in the case that it supports multiple data

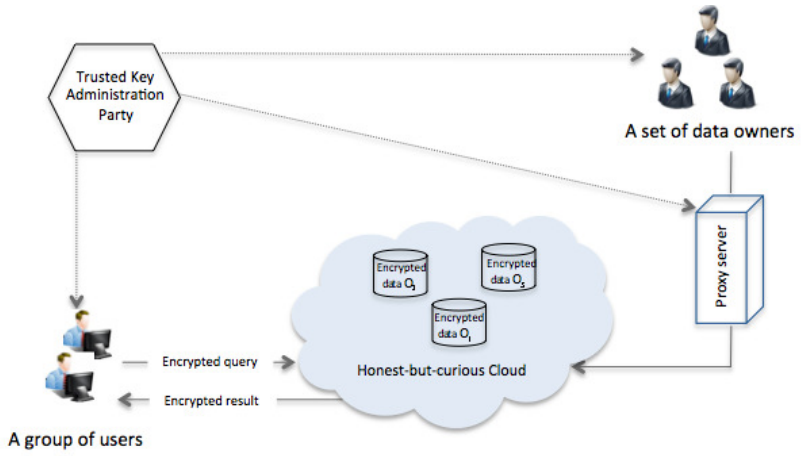


Figure 5.2: Multi-key data outsourcing scenario

owners and does not need client entity for processing user query. There are five types of entities in our system model (Figure 5.2):

- Data owners: A set of data owners $O = \{1, 2, \ldots, m\}$ in which any data owner O_i owns a share T_i of a common database D with n records, which has been horizontally partitioned among m data owners.
- Users: A group of authorized users $U = \{1, 2, \ldots, n\}$ that are able to search on the union of encrypted data outsourced by m data owners.
- Proxy server: A server with the responsibility of partially transforming (re-encryption) data encrypted with different keys under the same key.
- Cloud service provider: A server that complete the transformation (decryption) of data received from the proxy and retrieves the encrypted data according to users' requests. The encrypted data can be indexed in cloud service provider to improve query performance according to the type of query (see methods described in Chapter 4).
- key administration party: A fully trusted party that generates keys for participants and securely distributes them.

In this model called multi-key data outsourcing system, most of time data owners and the KA are offline and we assume that the authorization between the data owners and the users are appropriately done[1]. Moreover, we consider an *honest-but-curious* cloud server such that the server tries to get extra information from user queries and database access pattern.

[1]User authentication and access control are not the main focus in this thesis.

5.2.3 Definitions

Definition 1 (Negligible Function). A function $f(n)$ is called negligible if for each polynomial $p()$ there exists N such that for all integers $n > N$, $f(n) < |\frac{1}{p(n)}|$

Definition 2 (Decisional Diffie-Hellman (DDH) Assumption). The DDH problem is hard regarding a group $\mathbb{G}$ if for all probabilistic polynomial time adversaries A, there exists a negligible function $negl$ such that

$$|\mathsf{Pr}[(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta} = 1] - \mathsf{Pr}[(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1]| < negl(k)$$

where $\mathbb{G}$ is a cyclic group of order q ($|q| = k$) and g is a generator of $\mathbb{G}, \alpha, \beta, \gamma \in \mathbb{Z}_{\text{l}}$are uniformly randomly chosen.

5.3 Proxy-based SQL^{--} Query Processing

In this section, we summarize our proxy-based solution for supporting SQL^{--} queries over multi-key encrypted data outsourced by multiple data owners.

5.3.1 Proxy-based Equality test query processing

Consider a multi-key data outsourcing scenario, as illustrated in Figure 5.2, in which each share T_i, $1 \leq i \leq m$ contains a set of searchable attributes $T_{i,A_1,\ldots,A_{p_1}}$ and extra attributes $T_{i,B_1,\ldots,B_{p_2}}$ (p_1 and p_2 are not necessarily equal). For the sake of simplicity, let us assume that each share T_i includes one searchable attribute $T_{i,A}$ and one extra attribute $T_{i,B}$ with set of values V_A and V_B, respectively.

Given an equality test query q over outsourced encrypted data holding by cloud service provider, the wish is to obtain answer to query q while satisfying data privacy, query privacy, and user anonymous result.

Here, we describe a protocol for equality test, which adopts a proxy re-encryption scheme on searchable attribute of different data owners to have data encrypted under the same key. The protocol makes use of a fully trusted party KA to generate keys for data owners, users, proxy, and the cloud service provider. Data owners locally encrypt searchable attribute with their own keys and send the encrypted data to the proxy server that, in turn, overencrypts data with proxy-side key corresponding to each data owner. After data re-encryption, the proxy sends data to the cloud. When the user submits an equality query encrypted with her own key, the cloud service provider decrypts the received data from the proxy with the server-side key corresponding to the user and sends back the search result. The final result decrypts at user side and does not need filtering to weed out the false positives. A detailed description of the protocol is given below.

Input. A multi-key data outsourcing scenario (see Section 5.2.2).

Output. The result of user query $q : \sigma_{A.v}(T_1 \cup \ldots \cup T_m)$, where v is the equality condition posing by an authorized user.

Step 1: Initialization. The KA takes as input the security parameter 1^k and outputs public parameters Param and the master secret key K.

1.1 Generate two prime numbers p and q such that $p - 1 = 2q$

1.2 Generate a cyclic group $\mathbb{G}$ with generator g such that $\mathbb{G}$ is the unique order q subgroup of $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$. The generator g can be computed as the follows:

1. Randomly generate an integer $2 \leq g \leq p-1$
2. If $g^2 = 1 \ mod \ p$, go to 1
3. If $g^q \ != 1 \ mod \ p$, go to 1
4. Return g

1.3 Select K as a master key and $\mathsf{Param} = \langle \mathbb{G}, g, q \rangle$

Step 2: Key generation. The key Generator takes as input the master key K, $\mathsf{Param} = \langle \mathbb{G}, g, q \rangle$, and the identity of participant (data owner/user) to output keys for participants, proxy server and the cloud service provider.

2.1 For each data owner i, $1 \leq i \leq m$, the KA does the following:

1. Choose $K_{o_i} \xleftarrow{R} \mathbb{Z}_q$
2. Compute $K_{o_i}^{'} = \mathsf{K} - K_{o_i}$
3. Return keys K_{o_i} and $(i, K_{o_i}^{'})$ to the owner and proxy sides, respectively

2.2 For each user j, $1 \leq j \leq n$, the KA does the following:

1. Choose $K_{u_j} \xleftarrow{R} \mathbb{Z}_q$
2. Compute $K_{u_j}^{'} = \mathsf{K} - K_{u_j}$
3. Return keys K_{u_j} and $(j, K_{u_j}^{'})$ to the user and cloud service provider sides, respectively

Step 3: Encryption (owner-side). Data owners jointly generate a random $r \in \mathbb{Z}$ ($r \xleftarrow{R} \mathbb{Z}_q$), and broadcast r to their authorized users.

3.1 Each data owner O_i does the following (parallel operation):

1. Encrypt values of the attribute A with the key K_{o_i} and, if $A^{'}$ is the encrypted form of A, we set $T_{i,A'}^{'} = \{(g^r, g^{rK_{o_i}}.x) | x \in V_{i,A}\}$. Recall that $V_{i,A}$ is a set values corresponding to attribute A of share T_i.
2. Generate a new key $\bar{K}_{o_i}$ for encrypting the values of extra attribute

3. Encrypt values of extra attribute using $\bar{K}_{o_i}$ to obtain $T^{'}_{i,B'} = \{f_{\bar{K}_{o_i}}(x)|x \in V_{i,B}\}$, where f is an encryption function, which can be efficiently decrypted by key $\bar{K}_{o_i}$
4. Encrypt the key $\bar{K}_{o_i}$ with the key K_{o_i} to get $I_{o_i} = (g^r, g^{rK_{o_i}}.\bar{K}_{o_i})$
5. Send encrypted share $T^{'}_{i,A',B'}$ along with I_{o_i} to the proxy server

Step 4: re-encryption (proxy-side). Upon receiving ciphertexts from owner O_i, the proxy re-encrypt them with the proxy-side key related to O_i as the follows (parallel operation):

1. Find the proxy-side key $K^{'}_{o_i}$ corresponding to O_i
2. Compute $T^{''}_{i,A''} = \{(g^r)^{K^{'}_{o_i}}.g^{rK_{o_i}}.x = g^{r(K^{'}_{o_i}+K_{o_i})}.x = g^{r\mathsf{K}}.x \,|x \in V^{'}_{i,A'}\}$, where $V^{'}_{i,A'}$ is the set values of attribute $A^{'}$ from share $T^{'}_{i}$
3. Compute $I^{'}_{o_i} = (g^r)^{K^{'}_{0_i}}.g^{rK_{o_i}}.\bar{K}_{o_i} = g^{r(K^{'}_{o_i}+K_{o_i})}.\bar{K}_{o_i} = g^{r\mathsf{K}}.\bar{K}_{o_i}$
4. Set $T^{''}_{i,B''} = T^{'}_{i,B'}$
5. Outsource tuples $\langle T^{''}_{i,A'',B''}, I^{'}_{o_i}\rangle$ to the cloud service provider

Step 5: Search query (cloud-side). Upon receiving user j'query in an encrypted form $v^{'} = g^{rK_{u_j}} \cdot v$, the cloud does the following for each owner O_i:

1. Find the proxy-side key $K^{'}_{u_j}$ corresponding to the authorized user j
2. Decrypt each value $x \in V^{''}_{i,A''}$ using $K^{'}_{u_j}$ to obtain $\bar{T}_{i,\bar{A}} = \{g^{r\mathsf{K}}x.(g^r)^{-K_u} = g^{r(\mathsf{K}-K_u)} = g^{rK_u}.x\}$
3. Decrypt $I^{'}_{o_i}$ using key $K^{'}_{u_j}$ as $\bar{I}_{o_i} = g^{r\mathsf{K}}.(g^r)^{-K_{u_j}} \cdot \bar{K}_{o_i} = g^{r^{(\mathsf{K}-K_{u_j})}}.\bar{K}_{o_i} = g^{rK_{u_j}}.\bar{K}_{o_i}$
4. Set $\bar{T}_{i,\bar{B}} = T^{''}_{i,B''}$
5. Find tuples in $\bar{T}_i$ whose entry related to attribute $\bar{A}$ is equal to $v^{'}$to obtain result set $R_{i,C}$
6. Combine all $R_{i,C}, 1 \leq i \leq m$, and reorder tuples for the purpose of satisfying the "query anonymous result" property
7. Send $R^{'}_{C'} = \bigcup_{i=1}^{i=m} R_{i,C}$ together with $\bar{I}_i$ corresponding to each tuple to the user side

Note that in this step, cloud service provider can apply indexing methods such as $B^+ - tree$ on encrypted data to improve the query processing performance [18].

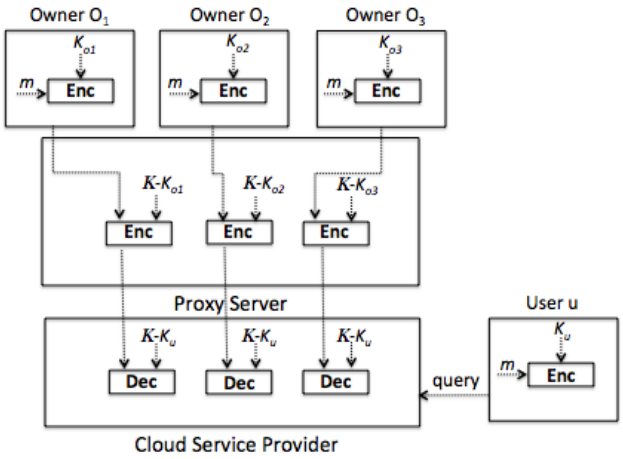


Figure 5.3: Encryption/decryption in proxy re-encryption scheme with 3 data owners and 1 user

Step 6: Post-processing (user side). For each tuple t in $R'_{C'}$, the user j fully decrypts the corresponding $\bar{I}_t$ with its own key as $g^{rK_{u_j}} \cdot \bar{K}_{u_j} \cdot (g^r)^{-K_{u_j}} = \bar{K}_{u_j}$ for the purpose of decrypting the tth extra value

Figure 5.3 shows how data encrypted with different keys can be transformed under the same key adopting El Gamal proxy encryption scheme.

A worked-out example

As an example, consider a scenario of 4 data owners and 1 user where each owner O_i has 1 value x_i in searchable attribute and user u is interested in values equal to its own value m. For simplicity, we assume two small prime numbers $p = 11$ and $q = 5$ with a generator $g = 3$ in range $[2, p-1]$ such that $(g^2! = 1 \bmod p)$ and $(g^p == 1 \bmod p)$. Now, lets suppose that the key administrator (KA) chooses $\mathsf{K} = 4$ as a master key and assigns random keys $K_1 = 7$, $K_2 = 2$, $K_3 = 5$, $K_4 = 9$ and $K_u = 4$ to the owners O_1, O_2, O_3, O_4 and user u, respectively. According to the protocol description, KA computes proxy-side keys for data owners and cloud-side keys for users to get $K'_1 = -3$, $K'_2 = 2$, $K'_3 = -1$, $K'_4 = -5$ and $K_u = 0$. Then, each owner O_i encrypts its own value x_i with its own key K_{o_i} (see Figure 5.4 (owner-side)), and send the results to the proxy server. The proxy re-encrypts each encrypted value x_i under proxy-side key K'_{o_i} (see Figure 5.4 (proxy-side)),

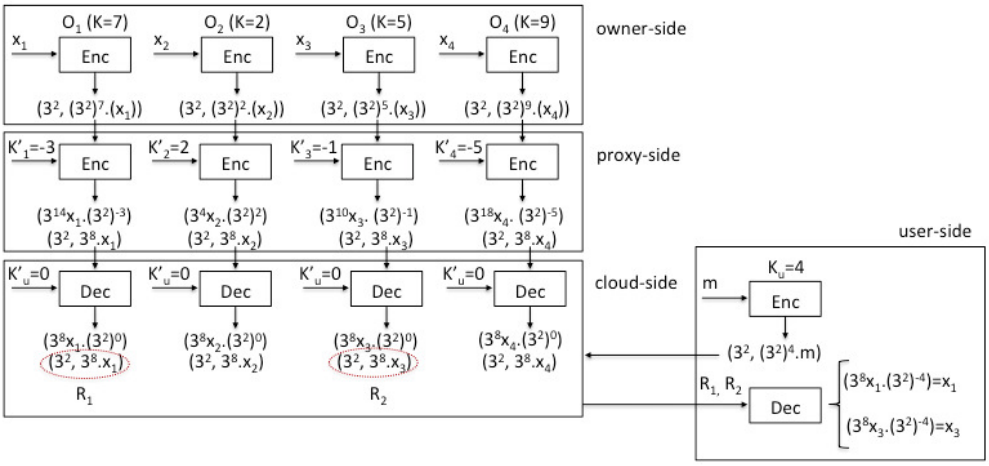


Figure 5.4: Example of multi-key outsourcing scenario based on proxy re-encryption with 4 data owners and 1 user where $g = 3$ and master key $\mathsf{K} = 4$

and sends the result to the cloud service provider. The cloud service provider partially decrypts each received value with its corresponding cloud-side key and finds the equal values related to the user search value (see Figure 5.4 (cloud-side)). With the assumption that values x_1 and x_3 are equal to the user value m, the cloud sends the results R_1 and R_3 corresponding to x_1 and x_3 to the user who in turn fully decrypts them to find the actual values without learning the keys of O_1 and O_3 (see Figure 5.4 (user-side)).

5.3.2 Proxy-based range query processing

In this section, we propose protocols to execute privacy-preserving range queries over partition data encrypted using different keys. As mentioned in Chapter 4, five category of solutions have been proposed for privacy-preserving range queries: based on bucketization [30, 34], order-preserving encryption schemes [2, 9, 65], specialized data structures [18, 60], fully homomorphic encryption method [66], and keyword-based search [27, 62, 17, 6, 47, 56, 4]. Regardless of the category, almost all of these solutions focus on single-key data outsourcing when data is encrypted under the same key at server side, and do not handle scenario in which data is encrypted using different keys (multi-key setting) by multiple data owners. The solution we present transforms a range query over real number data in a sequence of equality test queries, each resolved via proxy re-encryption scheme.

A naive approach.

Input. A multi-key data outsourcing scenario (see Section 5.2.2) in which each share T_i, $1 \leq i \leq m$, contains a searchable attribute $T_{i,A}$, and an extra attribute $T_{i,B}$; range query $\mathbf{r} = (r_{min}, r_{max})$; user query values $V_{\mathbf{r}} = \{x \in N | r_{min} \leq x \leq r_{max}\}$

Output. The result of user query $q : \sigma_{A.V_{\mathbf{r}}}(T_1 \cup \ldots \cup T_m)$.

Steps 1 and 2: Initialization and Key generation. These two steps correspond to the Steps 1 and 2 of equality test protocol (see Section 5.3.1)

Steps 3 and 4: Encryption (owner-side and proxy side). These two steps correspond to the Steps 3 and 4 of equality test protocol (see Section 5.3.1.

Step 5: Search query (server side). Upon receiving user set values in an encrypted form using user key K_u as $V_{\mathbf{r}}^{'} = \{g^{rK_u} \cdot v | v \in V_{\mathbf{r}}\}$, the server does the following for each owner O_i:

1-4 correspond to the Items 1-4 of equality test protocol (see Section 5.3.1),

5. Find tuples in $\bar{T}_i$ whose entry related to the attribute $\bar{A}$ belongs to the set $V_{\mathbf{r}}^{'}$ to obtain result set $R_{i,B}$

6-7 correspond to the Items 6-7 of equality test protocol (see Section 5.3.1).

Step 6: Post-processing (user side). This step corresponds to Step 6 of equality test protocol (see Section 5.3.1).

While this protocol is simple, it has a main drawback that does not support the domain of real numbers. Moreover, sending a sequence of all values in user range increases the computation and communication costs. To overcome these limitations, next we develop a new range protocol by splitting the domain of searchable attributes.

An improved approach. Here, we introduce some modifications in which the previous range protocol can execute on real number domain.

Input. A multi-key data outsourcing scenario (see Section 5.2.2). For the sake of simplicity, we assume that each share T_i, $1 \leq i \leq m$, contains only one searchable attribute $T_{i,A}$; range query $\mathbf{r} = (r_{min}, r_{max})$; user query values $V_{\mathbf{r}} = \{x \in R | r_{min} \leq x \leq r_{max}\}$

Output. The result of user query $q : \sigma_{A.V_{\mathbf{r}}}(T_1 \cup \ldots \cup T_m)$.

Step 1: Pre-processing.

- **Data owners side**: Each data owner O_i splits the searchable attribute $T.A$ into two sub attributes $T.X$, $T.Y$ as follows:
  1. $\forall v \in T.A$, $v = v_x + v_y$
  2. Suppose that $v_1 v_2 \ldots v_c$ is the integer part of v. Then we set $v_x = v_1.10^{c-1}$, $v_y = v - v_x$. Observe that $v_x \in N$ by definition and that T.Y is considered an extra attribute, only T.X is used as searchable attribute.

- **User side**: Suppose that q is a range query, with range $\mathbf{r} = (r_{min}, r_{max})$, the user u wants to perform.
  1. User u splits range boundaries as described above, obtaining $r_{min} = r_{min,x} + r_{min,y}$, $r_{max} = r_{max,x} + r_{max,y}$
  2. Suppose that α is the most significant digit of $r_{min,x}$ and β is the most significant digit of $r_{max,x}$, and that c_1 and c_2 are the number of digits of $r_{min,x}$ and $r_{max,x}$ respectively.
  3. The range query q is mapped onto 3 categories of equality queries:
     $q^{(1)} := \{i.10^{c_1-1}, i \in \{\alpha, \alpha+1, \ldots, 9\}\}$
     $q^{(2)} := \{\gamma.10^{\bar{c}}, \gamma \in \{1, 2, \ldots, 9\}, c_1 - 1 < \bar{c} < c_2 - 1, \bar{c} \in N\}$
     $q^{(3)} := \{\eta.10^{c_2-1}, \eta \in \{1, 2, \ldots, \beta\}\}$

Steps 1 and 2: Initialization and Key generation. These two steps correspond to the Steps 1 and 2 of equality test protocol (see Section 5.3.1)

Steps 3 and 4: Encryption (owner-side and proxy side). These two steps correspond to the Steps 3 and 4 of equality test protocol (see Section 5.3.1) considering X and Y as A and B, respectively.

Step 5: Search query (server side). Upon receiving user set values in an encrypted form using user key K_u as $V'_{\mathbf{r}} = \{(g^{rK_u} \cdot v_1), (g^{rK_u} \cdot v_2), (g^{rK_u} \cdot v_3) | v_1 \in q^{(1)}, v_2 \in q^{(2)}, \text{ and } v_3 \in q^{(3)}\}$, the server does the following for each owner O_i:

1-4 correspond to the Items 1-4 of equality test protocol (see Section 5.3.1) considering X and Y as A and B, respectively.

5. Find tuples in $\overline{T}_i$ whose entry related to the attribute $\bar{A}$ belongs to the set $V_{\mathbf{r}}^{'}$ to obtain result set $R_{i,Y}^{(1)}, R_{i,Y}^{(2)}, R_{i,Y}^{(3)}$ corresponding to $q^{(1)}, q^{(2)}, q^{(3)}$, respectively .
6. Combine all $R_{i,Y}^{(1)}, R_{i,Y}^{(2)}, R_{i,Y}^{(3)}$ from m data owners, $1 \leq i \leq m$,
7. Send $R_Y = \{R^{(1)} = \cup_{i=1}^{i=m} R_{i,Y}^{(1)}, R^{(2)} = \cup_{i=1}^{i=m} R_{i,Y}^{(2)}, R^{(3)} = \cup_{i=1}^{i=m} R_{i,Y}^{(3)}\}$ together with $\bar{I}$ corresponding to each tuple of R_Y to the user side.

Step 6: Post-processing (user side). For each tuple j in R_Y, the user fully decrypts the corresponding $\bar{I}_j$ with its own key as $g^{rK_u} \cdot \bar{K}_j \cdot (g^r)^{-K_u} = \bar{K}_j$ for the purpose of decrypting the jth extra value. By definition of $q^{(2)}$, all the tuples in $R^{(2)}$ are in the range $\mathbf{r} = (r_{min}, r_{max})$, thus the user u should only select from $R^{(1)}$ those tuples t for which $t.y \geq r_{min,y}$, and from $R^{(3)}$ those tuples t such that $t.y \leq r_{max,y}$.

5.3.3 Proxy-based equi-join query processing

In order to perform equi-join query with equality selection (Section 3.6.1) between partitions encrypted with different keys, the join attributes T_A, $T_A^{'}$ need to be encrypted under the same key. This can be achieved utilizing proxy re-encryption scheme. The protocol proposed for equality test query (see Section 5.3.1) can be adopted for equi-join query with the difference that after proxy-side encryption, the cloud finds, across all the data owners, the tuples $t \in T$, $\bar{t} \in T^{'}$ which respect the join condition, that is $t_A = \bar{t}_A$. Then, the cloud partially decrypts the obtained tuples with cloud-side key and selects the joined tuples that satisfy the selection criteria, i.e. $t_B = v$. The selected tuples are then sent back to user for complete decryption.

5.4 Security analysis

In this section we prove that the proxy is not able to infer information about the encrypted hash values in a chosen plaintext attack according to the proposed definitions (see Section 5.2.3).

Theorem 1. If the DDH problem is hard relative to $\mathbb{G}$, then the equality test encryption scheme ($\mathcal{ET}$) is $IND-CPA$ (indistinguishability under chosen plaintext attack) secure against the proxy. That is, for all PPT adversaries $\mathcal{A}$ there exists a negligible function $negl$ such that

$$Succ^{\mathcal{A}}_{\mathcal{ET},P^{(k)}} = \mathsf{Pr}\left[b^{'} = b \;\middle|\; \begin{array}{l} (\mathsf{Params},\ \mathsf{K}) \leftarrow \mathcal{ET}\text{-}\mathsf{Init}(1^k) \\ (K, K^{'}) \leftarrow \mathcal{ET}\text{-}\mathsf{KeyGen}(\mathsf{K},\mathsf{O}) \\ x_0, x_1 \leftarrow \mathcal{A}^{\mathcal{ET}-Enc(K,\cdot)}(K^{'}) \\ b \xleftarrow{R} \{0,1\} \\ T_i^{'}(x_b) = \mathcal{ET} - Enc(K_i, x_b) \\ b^{'} \leftarrow \mathcal{A}^{\mathcal{ET}-Enc(K,\cdot)}(K^{'}, T_i^{'}(x_b)) \end{array} \right] \left\langle \frac{1}{2} + negl(k) \right.$$

where O is a set of data owners and K, $K^{'}$ are the set of owner side key sets and the set of proxy side keys, respectively.

Proof. Let's consider a PPT adversary $\mathcal{A}^{'}$ who attempts to challenge the DDH problem using $\mathcal{A}$ as a sub-routine. Recall that $A^{'}$ is given $\mathbb{G}, q, g, g_1, g_2, g_3$ as input, where $g_1 = g^{\alpha}$, $g_2 = g^{\beta}$ and g^3 could be $g^{\alpha\beta}$ or g^{γ} for some random α, β, γ.

- $\mathcal{A}^{'}$ sends $\mathbb{G}, q, g$ to $\mathcal{A}$ as the public parameter.
- $\mathcal{A}^{'}$ selects randomly $K_i^{'} \xleftarrow{R} Z_q$ for each data owner i, $1 \leq i \leq m$, and computes $g^{K_i} = g_1 \cdot g^{-K_i^{'}} = g^{\alpha - K_i^{'}}$. It then keeps $(i, g^{K_i}, K_i^{'})$ and sends all $(i, K_i^{'})$ to $\mathcal{A}$,
- Whenever $\mathcal{A}$ requires oracle access to the data owner encryption algorithm, it passes x, which is a value of a column to $\mathcal{A}^{'}$,
- $\mathcal{A}^{'}$ chooses randomly $r \xleftarrow{R} Z_q$ and replies with $(g^r, g^{rK_i} \cdot x)$,
- $\mathcal{A}$ chooses a pair (x_0, x_1) and sends x_0, x_1 to $\mathcal{A}^{'}$,
- $\mathcal{A}^{'}$ picks a random bit $b \in_R \{0,1\}$ and executes $g_2, {g_2}^{-K_i^{'}} g_3(x_b)$,
- $\mathcal{A}$ outputs a bit $b^{'} \in \{0,1\}$, if $b = b^{'}$, $\mathcal{A}^{'}$ outputs 1, otherwise output 0.

– Case 1: $g_3 = g^{\gamma}$, in this case g^{γ} is a random group element of $\mathbb{G}$ because γ is chosen at random and therefore ${g_2}^{-K_i^{'}} g_3 \cdot x_b$ is a random group element of $\mathbb{G}$ and gives no information about x_b. Moreover, g_2 does not contain any information about x_b. Hence, the probability that the adversary can successfully output $b = b^{'}$ is exactly $\frac{1}{2}$ when b is chosen uniformly random.

$$\mathsf{Pr}[\mathcal{A}^{'}(G, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1] = \frac{1}{2}$$

– Case 2: if $g_3 = g^{\alpha\beta}$, then ${g_2}^{-K_i^{'}} g^3 \cdot x_b = g^{\beta - K_i^{'}} g^{\alpha\beta} \cdot x_b = g^{\beta(\alpha - K_i^{'})}$, which is a valid ciphertect under $\mathcal{ET}$ encryption scheme. So we have:

$$\mathsf{Pr}[\mathcal{A}^{'}(G, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1] = Succ^{A}_{\mathcal{ET}, P^{(k)}}$$

If DDH problem is hard regarding $\mathbb{G}$, then $Succ^{A}_{\mathcal{ET}, P^{(k)}} < \frac{1}{2}$ is true. □

Theorem 1 shows that with knowing owner side key, the proxy learns nothing about the encrypted value in a chosen plaintext attack. The same theory can be applied for proving that cloud service provider can not distinguish the ciphertexts without the help of user side.

5.5 Cost analysis

5.5.1 Theoretical cost analysis

In this section, we analyze the equality test protocol exploiting proxy re-encryption scheme in terms of computation and communication costs. With the assumption that each data owner has $\frac{n}{m}$ expected records (uniform distribution), we quantify the computation and communication costs of the protocol as below.

-Computation cost analysis. Computation cost is analyzed due to the number of encryption/decryption performed by the protocol. Let C_E denote the cost of encryption/decryption using El Gamal proxy encryption, and C_f denote the cost of encryption/decryption by function f using for extra attribute.

Step 1. No computation cost.

Step 2. No computation cost.

Step 3. For each data owner O_i we have:

1. $C_E(|T_{i,A}|) = \frac{n}{m}$, for encrypting the values of attribute A using key K_{o_i},
2. No computation cost,
3. $C_f(|T_{i,B}|) = \frac{n}{m}$, for encrypting the values of extra attribute B with $\bar{K}_{o_i}$,
4. $C_E(\bar{K}_{0_i}) = 1$, for encrypting the new key using owner's key K_{o_i},
5. No computation cost.

The total cost of this step for each owner O_i is $2\frac{n}{m}+1$, and accordingly $2n+m$ for m data owners.

Step 4. For each data owner O_i we have:

1. No computation cost,
2. $C_E(|T^{'}_{i,A'}|) = \frac{n}{m}$, for re-encrypting values of attribute $A^{'}$ using proxy side key $K^{'}_{o_i}$,
3. $C_E(\bar{K}_{o_i}) = 1$, for re-encrypting $\bar{K}_{o_i}$ with the proxy side key $K^{'}_{o_i}$,

4-5. No computation cost,

The total cost of this step for each data owner O_i is $\frac{n}{m}+1$, and accordingly $n+m$ for m data owners.

Step 5. For each data owner O_i we have:

1. No computation cost,
2. $C_E(|T^{''}_{i,A''}|) = \frac{n}{m}$, for partially decrypting values of attribute $A^{''}$ using proxy side key corresponding to the user K_{u_j},
3. $C_E(I^{'}_{o_i}) = 1$, for partially decrypting $I^{'}_{o_i}$ using the proxy side key related to the user K_{u_j},

4-5. No computation cost,

The total cost for this step for each data owner is $\frac{n}{m}+1$, and accordingly $n+m$ for m data owners.

Step 6. Let t denote the number of tuples for which the server finds the equality match with the user request. The cost of this step is $tC_E(1)$ for fully decrypting $\bar{I}_{o_i}$ with the user key K_u to get the actual key $\bar{K}_{o_i}$ plus $tC_E(1)$ for decrypting the extra value with the key $\bar{k}$.

The total computation cost of the protocol by assuming a unitary cost for C_f and C_E is given by:

$$\begin{aligned} C &= (2n+m)+(n+m)+(n+m)+2t \\ &= 4n+3m+2t \in O(n) \end{aligned} \tag{5.1}$$

- Communication cost analysis. Communication cost is analyzed as the number of bits transmitted during the protocol execution.

Let l denote the length of each encrypted codeword of the domain of El Gamal proxy encryption and $l^{'}$ the length of encryption function f on extra attribute.

Step 1-2. No communication cost.

Step 3. For each data owner O_i we have:

1-4. No communication cost,

5. $(|T^{'}_{i,A'}|) \cdot l + (|T^{'}_{i,B'}|) \cdot l^{'} + (1) \cdot l$ for sending $T^{'}_{i,A',B'}$ along with I_{o_i} to the proxy server.

The total cost of this step for each owner is $(\frac{n}{m} + 1) \cdot l + \frac{n}{m} \cdot l^{'}$ and accordingly $(n + m) \cdot l + n \cdot l^{'}$ for m data owners.

Step 4. For each data owner O_i we have:

1-4. No communication cost,

5. $(|T^{''}_{i,A''}|) \cdot l + (|T^{''}_{i,B''}|) \cdot l^{'} + (1) \cdot l$ for outsourcing $T^{''}_{i,A'',B''}$ along with $I^{'}_{o_i}$ to the cloud service provider.

The total cost of this step for each owner is $(\frac{n}{m} + 1) \cdot l + \frac{n}{m} \cdot l^{'}$ and accordingly $(n + m) \cdot l + n \cdot l^{'}$ for m data owners.

Step 5.

1-6. No communication cost,

7. Let t denote the number of tules in result set $R^{'}_{C'}$ whose values corresponding to searchable attribute $\bar{A}$ are equal to the user requested value $v^{'}$, the total cost is $t \cdot l + t \cdot l^{'}$.

Step 7. No communication cost.

Overall the communication cost is given by:

$$C^{'} = (2n + 2m + t) \cdot l + (2n + t) \cdot l^{'} \tag{5.2}$$

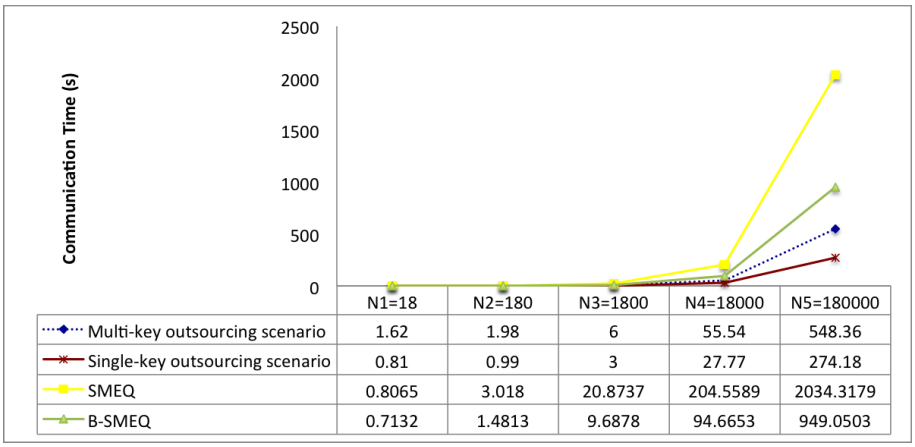
Hence, the communication cost is synoptically observed by $O(n)$.

5.5.2 Experimental cost analysis

In this section, we perform experiment to compute the communication time of executing the equality test queries adopting proxy re-encryption in multi-key data outsourcing scenario and compare it with three other solutions in multi-party computation and data outsourcing scenarios. We test the implementation using Castalia simulator on a Linux machine with dual Intel CPU running at 2.26 GHz, and 2GB Ram. In this experiment, we also

construct 7 nodes in Castalia including 3 data owners, 1 user, 1 proxy server, 1 for cloud service provider and 1 for key administrator (KA). For the sake of simplicity, each owner holds a share from a common database with one searchable attribute, which are uniformly distributed. To encrypt the values in searchable attribute, we implement the El Gamal encryption scheme with two small prime numbers $p = 11,\ q = 5$ and generator $g = 3$. Note that choosing small prime numbers does not affect the communication time. By running El Gamal algorithm, data owners receive owner side keys from the KA as $\{K_1 = 7, K_2 = 2, K_3 = 5, K_4 = 9\}$, and user gets $K_u = 4$ and accordingly, the proxy-side keys for data owners and cloud-side key for the user are computed as $\{K_1^{'} = -3, K_2^{'} = 2, K_3^{'} = -1, K_4^{'} = 4\}$, and $K_u = 0$ respectively.

The experimental result is shown in the Figure 5.5 (dotted line). Each point in the chart corresponds to the average communication time obtained by running 10 equality test queries on searchable attribute with range $[1, 100]$. The x axis shows the number of records of the common database, which is horizontally partitioned among 3 owners and the y axis shows the average time of outsourcing encrypted data from owners to the cloud service provider via proxy server. As figure shows, the average communication time is slightly twice more than the communication time of single-key data outsourcing technique (red line) where data owners encrypt their data with their own public key and outsource them to the cloud service provider. Then cloud service provider superencrypts data with all different keys, resulting data under the same key. On the other hand, it has less communication time than B-SMEQ (green line) and SMEQ (yellow line), which have been already described in Chapter 3. Based on our theoretical results, this difference decreases when the number of data owners increase.



	N1=18	N2=180	N3=1800	N4=18000	N5=180000
Multi-key outsourcing scenario	1.62	1.98	6	55.54	548.36
Single-key outsourcing scenario	0.81	0.99	3	27.77	274.18
SMEQ	0.8065	3.018	20.8737	204.5589	2034.3179
B-SMEQ	0.7132	1.4813	9.6878	94.6653	949.0503

Figure 5.5: Communication time vs Number of records

5.6 Chapter summary

In this chapter, we proposed protocols for performing SQL^{--} operations in multi-key data outsourcing scenario. Unlike existing approaches, our approach support queries over data encrypted with different keys (multi-key setting) for a group of authorized users. The proposed approach adopts proxy re-encryption scheme to reduce the workload of data re-encryption at server side while the proxy is not able to infer information about the encrypted value in a chosen plaintext attack.

Chapter 6

Conclusion and Future Work

In this chapter, we briefly summarizes the contributions of this thesis and we outline some future work.

6.1 Conclusion

Multi-party computation (MPC) and data outsourcing (DO) are two paradigms for performing queries over large amount of data shared among different data owners. In MPC, a collection of data owners wish to share data and perform jointly query processing on their private data, whereas in DO, data owners intend to outsource their private data to a remote service provider.

An important research challenge in MPC and DO paradigms is how to support queries in a privacy way when no data owner or remote service provider can be trusted enough to know all the inputs. This challenge can be addressed by encrypting data before sharing/outsourcing, which poses a new concern related to the query processing over encrypted data. For this reason, in this thesis we made the following contributions to the area of privacy-preserving query processing (P^3Q) by secure multi-party computation (SMC) and encrypted data outsourcing (EDO) paradigms:

- After a brief introduction and discussion of some known P^3Q methods in MPC (Chapter 2), we proposed a novel solution B-SMEQ to address the P^3Q problem by privately computing selection, range and equi-join queries so called SQL^{--}. Unlike existing approaches, our protocol scales well to large size data and is designed to work with more than two parties. B-SMEQś bucketization technique reduces the querier's computational burden and keeps query execution reasonably fast even when the number of records in the data owners' private databases increases. Experimental tests on randomly generated data

bases with around $2 \cdot 10^5$ records confirm the efficiency of our protocol in term of computational and, more importantly, communication complexity.
A major feature of our approach is using SMC to compute queries on partitioned data, leaving it to the querier to perform any additional processing on the query results. We are fully aware that SMC can be used more ambitiously, e.g. to privately compute metrics over data partitions, in the line of privacy-preserving data mining. However, we claim that decoupling (i) private querying and (ii) metrics computation on plaintext (the latter step was not covered in this thesis) makes our approach much more viable for practical applications that involve collaboration among mutually distrustful parties. Encryption systems that are homomorphic with respect to multiple arithmetic operators (typically, addition and multiplication) may one day provide us with efficient group arithmetics on encrypted data[1]; however, group arithmetics covers only a tiny fraction of the metrics that one may want to compute. Even computing simple ratios requires at least floating-point division, which is not supported by group arithmetics. Instead, our private query technique "sanitizes" the query results with respect to privacy, unlinking them from the original data held by the protocol participants. This way query results can be safely committed to (even untrusted) third parties for further processing.

- We showed how the idea underlying B-SMEQ protocol can be extended to tackle privacy-preserving range queries over real domain, transforming a range query in a sequence of equality test queries. Moreover, by appropriately splitting the searchable attribute, we reduced the number of equality test queries.

- We proposed a SMC-based technique for securely performing equi-join queries over big partition tables, which is still a challenge in cloud environment. The proposed protocol is based on bucketization and MapReduce techniques. In Map phase, the encrypted data under the same key is located at the same owner, and then a ring is organized to connect each owner to the other for sharing data. In Reduce phase, a SMC-based equality query with selection criteria is performed, which

[1]To the best of our knowledge, even devising a practical doubly homomorphic scheme, where one can both add and multiply ciphertexts, is still an open problem. Boneh et al. [12] solved a special case of the problem, but under their scheme ciphertexts can be multiplied only once.

adopts B-SMEQ protocol to reduce the communication and computation costs.

- We studied P^3Q techniques over encrypted data in DO paradigm and compare them due to some security requirements and efficiency (Chapter 4).

- We designed a set of searchable encryption techniques for supporting SQL^{--} queries in multi-key data outsourcing scenario where a set of data owners with different keys outsource their data in encrypted form to a cloud service provider. Our proposed protocol adopts proxy re-encryption scheme on searchable attributes of different data owners to bring data encrypted under the same key (single-key setting) while proxy is not able to learn anything about the encrypted message.

- We implemented the proposed SMC and EDO protocols for performing SQL^{--} operations in Castalia simulator in terms of communication cost. Moreover, we analyzed them in terms of computation cost.

6.2 Future work

The research described in this thesis can be extended along several directions as follows:

- In some circumstances in SMC paradigm, the data owners may not be able to agree on trusted third party (TTP). Hence, an important open question is how to develop B-SMEQ protocol without the help of TTP for the realization of bucketization scheme.

- One aspect of our future work in EDO paradigm may be adopting some indexing methods (Chapter 4) such as bucketization on searchable attributes to improve the efficiency by reducing the number of encryption and decryption in proxy and cloud sides.

- Another possible extension in EDO paradigm is to achieve access pattern privacy. A weakness of most data outsourcing techniques that allow the server to infer some information about the queries by knowing the access pattern of users.

Bibliography

[1] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 86–97, New York, NY, USA, 2003. ACM.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, 2009.

[4] M. R. Asghar, Giovanni Russello, Bruno Crispo, and Mihaela Ion. Supporting complex queries and access policies for multi-user encrypted databases. *The 5th ACM Workshop on Cloud Computing Security Workshop (CCSW)*, 2013.

[5] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, February 2006.

[6] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In Liqun Chen, Yi Mu, and Willy Susilo, editors, *ISPEC*, volume 4991 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2008.

[7] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *In EUROCRYPT*, pages 127–144. Springer-Verlag, 1998.

[8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[9] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 224–241, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 417–426. ACM, 2008.

[11] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[12] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the Third international conference on Applied Cryptography and Network Security*, ACNS'05, pages 442–455, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] Yan cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data, 2004.

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[16] Christopher Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Science*, 13(4):593–622, 11 2005.

[17] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[18] Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 93–102, New York, NY, USA, 2003. ACM.

[19] Tran Khanh Dang. Privacy-preserving basic operations on outsourced search trees. In *ICDE Workshops*, page 1194, 2005.

[20] George I. Davida, David L. Wells, and John B. Kam. A database encryption system with subkeys. *ACM Trans. Database Syst.*, 6(2):312–328, 1981.

[21] Changyu Dong, Giovanni Russello, and Naranker Dulay. Shared and searchable encrypted data for untrusted servers. *Journal of Computer Security*, 19(3):367–397, 2011.

[22] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[23] Fatih Emekci, Divyakant Agrawal, Amr El Abbadi, and Aziz Gulbeden. Privacy preserving query processing using third parties. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 27–, Washington, DC, USA, 2006. IEEE Computer Society.

[24] Sara Foresti. *Preserving Privacy in Data Outsourcing*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[25] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, volume 3027 of *LNCS*, pages 1–19, 2004.

[26] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[27] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[29] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[30] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 216–227, New York, NY, USA, 2002. ACM.

[31] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Efficient Execution of Aggregation Queries over Encrypted Relational Databases. In Yoon-Joon Lee, Jianzhong Li, Kyu-Young Whang, and Doheon Lee, editors, *Database Systems for Advanced Applications*, volume 2973 of *Lecture Notes in Computer Science*, chapter 10, pages 633–650. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2004.

[32] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 155–175, Berlin, Heidelberg, 2008. Springer-Verlag.

[33] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal*, 21(3):333–358, June 2012.

[34] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 720–731. VLDB Endowment, 2004.

[35] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *In CRYPTO 2003, Springer-Verlag (LNCS 2729*, pages 145–161. SpringerVerlag, 2003.

[36] Wassim Itani, Ayman I. Kayssi, and Ali Chehab. Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. In *DASC*, pages 711–716. IEEE, 2009.

[37] Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *in Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2003.

[38] Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik, and Yonghua Wu. A framework for efficient storage security in rdbms. In *In EDBT*, 2004.

[39] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Proceedings of the 14th international conference on Financial cryptograpy and data security*, FC'10, pages 136–149, Berlin, Heidelberg, 2010. Springer-Verlag.

[40] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 85–86, New York, NY, USA, 2012. ACM.

[41] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *in Advances in Cryptology - CRYPTO 2005, LNCS*, pages 241–257. Springer, 2005.

[42] C C Lee, P. S. Chung, and M. S. Hwang. A survey on attribute-based encryption schemes of access control in cloud environments. *International Journal of Network Security*, 15(4):231–240, 2013.

[43] Ronghua Li and Chuankun Wu. Co-operative private equality test. *I. J. Network Security*, 1(3):149–153, 2005.

[44] Ping Lin and K. Seluk Candan. Hiding traversal of tree structured data from untrusted data stores. In Hsinchun Chen, Richard Miranda, Daniel Dajun Zeng, Chris C. Demchak, Jennifer Schroeder, and Therani Madhusudan, editors, *ISI*, volume 2665 of *Lecture Notes in Computer Science*, page 385. Springer, 2003.

[45] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[46] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *In Proceedings of the 8th International Conference on Information Security*, pages 314–328. Springer-Verlag, 2005.

[47] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[48] Qiwei Lu, Yan Xiong, Xudong Gong, and Wenchao Huang. Secure collaborative outsourced data mining with multi-owner in cloud computing. In Geyong Min, Yulei Wu, Lei (Chris) Liu, Xiaolong Jin, Stephen A. Jarvis, and Ahmed Yassin Al-Dubai, editors, *TrustCom*, pages 100–108. IEEE Computer Society, 2012.

[49] Masahiro MAMBO and Eiji OKAMOTO. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts (special section on cryptography and information security). *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(1):54–63, jan 1997.

[50] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 245–254, New York, NY, USA, 1999. ACM.

[51] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[52] Arjun Narayan and Andreas Haeberlen. Djoin: differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 149–162, Berkeley, CA, USA, 2012. USENIX Association.

[53] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, 2001.

[54] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[55] Jong Hwan Park. Efficient hidden vector encryption for conjunctive queries on encrypted data. *IEEE Trans. on Knowl. and Data Eng.*, 23(10):1483–1497, October 2011.

[56] Raluca A. Popa and Nickolai Zeldovich. Multi-key searchable encryption. *IACR Cryptology ePrint Archive*, 2013:508, 2013.

[57] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium*

on Operating Systems Principles, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[58] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[59] Yingpeng Sang, Hong Shen, Yasuo Tan, and Naixue Xiong. Efficient protocols for privacy preserving matching against distributed datasets. In *Proceedings of the 8th international conference on Information and Communications Security*, ICICS'06, pages 210–227, Berlin, Heidelberg, 2006. Springer-Verlag.

[60] Elaine Shi, John Bethencourt, T h. Hubert, Chan Dawn, and Song Adrian Perrig. Multi-dimension range query over encrypted data. In *In IEEE Symposium on Security and Privacy*, pages 350–364. Smith, 2007.

[61] Tony Smith. Dvd jon: buy drm-less tracks from apple itunes. Cryptology ePrint Archive, Report 2003/216, 2005.

[62] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[63] XiuXia Tian, Xiaoling Wang, and Aoying Zhou. Dsp re-encryption based access control enforcement management mechanism in daas. *I. J. Network Security*, 15(1):28–41, 2013.

[64] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.

[65] Hui Wang and Laks V. S. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 127–138. VLDB Endowment, 2006.

[66] Shiyuan Wang, Divyakant Agrawal, and AE Abbadi. Is homomorphic encryption the holy grail for database queries on encrypted data. Technical report, Technical report, Department of Computer Science, UCSB, 2012.

[67] Huiqi Xu, Shumin Guo, and Keke Chen. Building confidential and efficient query services in the cloud with rasp data perturbation. *CoRR*, abs/1212.0610, 2012.

[68] Andrew C Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[69] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

Appendix

A. Experimental analysis of data and query distribution

Equality query

We performed experiments with two different data distribution: uniform and normal (see Figure 6.1). For each experiment, we generate 4 equality queries where the value is extracted according to a uniform distribution and 4 queries whose values are extracted according to a normal distribution on searchable attribute A with range $[1, 100]$. For normal distribution, the standard deviation and mean are equal to 12 and 49.5 respectively in order to ensure adequate coverage of buckets.

Range query

For finding the optimal bucket number, which is independent from the data and query distribution, we generate two different set of queries Q_{uni}, Q_{nor}, in which each has 8 queries with wide ranges corresponding to the searchable attribute with range $[1, 100]$. Range of each query in Q_{uni}, Q_{nor} is selected randomly from uniform and normal distribution, respectively. For each experiment, we generated 4 range queries whose value is extracted according to a uniform distribution and 4 queries whose values are extracted according to a normal distribution. The searchable attribute A has range $[1, \ldots, 100]$. For normal distribution, the standard deviation and mean are equal to $sd = 10$ and $\mu = 50.5$ respectively in order to ensure adequate coverage of buckets.

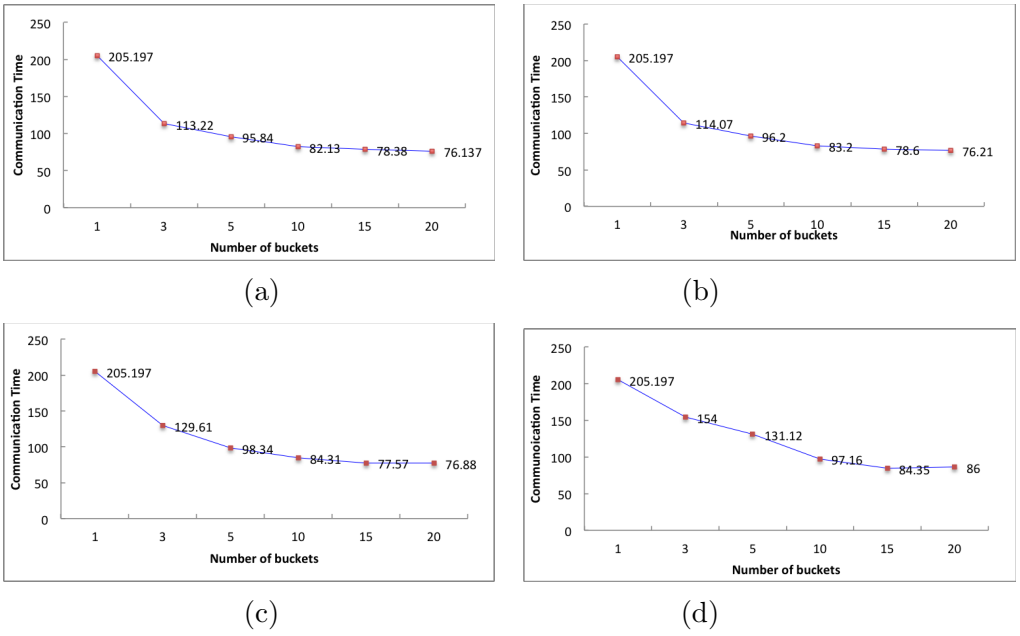


Figure 6.1: (a) Uniform Data- Uniform Query (b) Uniform Data- Normal Query (c) Normal Data- Uniform Query (d) Normal Data- Normal Query

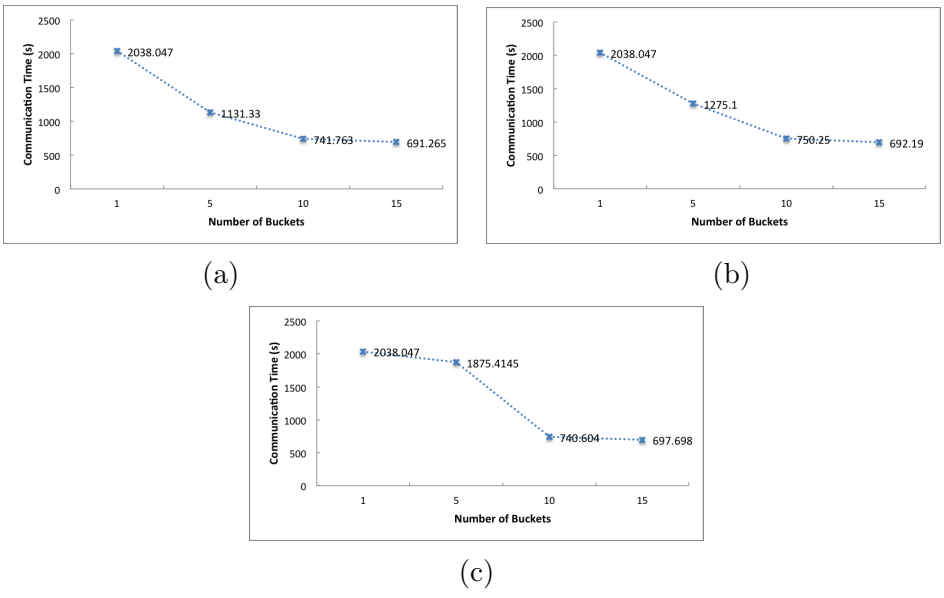


Figure 6.2: (a) Uniform data-Uniform query (b) Normal data-Uniform query (c) Normal data-Normal query