PRIVACY-PRESERVING PROTOCOLS FOR AGGREGATE LOCATION QUERIES VIA HOMOMORPHIC ENCRYPTION AND MULTIPARTY COMPUTATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Cihan Eryonucu

July 2019

Privacy-Preserving Protocols for Aggregate Location Queries via Homomorphic Encryption and Multiparty Computation
By Cihan Eryonucu
July 2019

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Erman Ayday(Advisor)

Özgür Ulusoy

Ali Aydın Selçuk

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

PRIVACY-PRESERVING PROTOCOLS FOR AGGREGATE LOCATION QUERIES VIA HOMOMORPHIC ENCRYPTION AND MULTIPARTY COMPUTATION

Cihan Eryonucu
M.S. in Computer Engineering
Advisor: Erman Ayday
July 2019

Two main goals of the businesses are to serve their customers better and in the meantime, increase their profit. One of the ways that businesses can improve their services is using location information of their customers (e.g., positioning their facilities with an objective to minimize the average distance of their customers to their closest facilities). However, without the customer's location data, it is impossible for businesses to achieve such goals. Luckily, in today's world, large amounts of location data is collected by service providers such as telecommunication operators or mobile apps such as Swarm. Service providers are willing to share their data with businesses, doing this will violate the privacy of their customers. Here, we propose two new privacy-preserving schemes for businesses to utilize location data of their customers that is collected by location-based service providers (LBSPs). We utilize lattice based homomorphic encryption and multiparty computation for our new schemes and then we compare them with our existing scheme which is based on partial homomorphic encryption. In our protocols, we hide customer lists of businesses from LBSPs, locations of the customers from the businesses, and query result from LBSPs. In such a setting, we let the businesses send location-based queries to the LBSPs. In addition, we make the query result only available to the businesses and hide them from the LBSPs. We evaluate our proposed schemes to show that they are practical. We then compare our three protocols, discussing each one's advantages and disadvantages and give use cases for all protocols. Our proposed schemes allow data sharing in a private manner and create the foundation for the future complex queries.

Keywords: Data Privacy, Information Security, Homomorphic Encryption, Location Privacy, Multiparty Computation.

ÖZET

HOMOMORFİK ŞİFRELEME VE ÇOK PARTİLİ HESAPLAMA KULLANANARAK GİZLİLİĞİ KORUYAN TOPLU KONUM SORGULARI

Cihan Eryonucu
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Erman Ayday
Temmuz 2019

Müşterilere daha iyi hizmet sağlamak ve bunu yaparken de karlarını arttırmak şirketlerin iki ana hedefidir. Şirketlerin müşteriye servislerinin kalitesini arttırmanın bir yolu ise müşterilerin konum bilgisini kullanmaktır (örneğin, şirket müşterilerinin en yakın şubelerine olan ortalama mesafeyi azaltcak şekilde tesisleri konumlandırmak). Ancak, müşterilerin konum bilgisi olmadan şirketlerin hedeflerine ulaşmalrı mümkün değildir. Neyse ki, bugünün dünyasında, telekomünikasyon operatörleri ve Swarm uygulamasi gibi servis sağlayacıları yüksek miktarda veri toplamaktadır. Servis sağlayıcılar ellerinde bulunan verileri, bu tip şirketlerle paylaşmaya istekli ancak, bu paylaşımı müşterilerin gizliliğini ihlal etmeden yapmak sorunlara yol açabilir. Bu çalışmada biz, şirketler tarafından kullanıması için, servis sağlayıcıların topladığı konum verilerinin gizliliğini koruyan iki yeni protokol tasarladık. Yeni protokollerimiz için örgü tabanlı homomorfik sifreleme ve çok partili hesaplamayı kullanmaktayız. Aynı zamanda daha önce tasarladığımız kısmi homomorfik şifreleme tabanlı protokol ile iki yeni protokollerimizi karşılaştırıyoruz. Protokollerimizde, şirketlerin müşteri listesini servis sağlayıcılarından, müşterilerin konum bilgilerini şirketlerden ve sorgu sonucunu ise servis sağlayıcılarından saklıyoruz. Protokollerimizi deneysel ortamda değerlendirip onların pratik olduğunu gösteriyoruz. Sonrasında, bu üç protokolü kendi aralarında karşılaştırarak her birinin yararları ve zararları hakkında tartışıyoruz ve her protokol için birer kullanım örneği veriyoruz. Tasarladığımız protokeller veri paylaşımını gizliliği koruyan şekilde gerçekleştiriyor ve aynı zamanda gelecekteki karmaşık sorgular için de bir temel oluşturuyoruz.

Anahtar sözcükler: Veri gizliliği, bilgi güvenliği, homomorfik şifreleme, konum gizliliği, çok partili hesaplama.

Acknowledgement

First and foremost, I would like to thank my advisor Asst. Prof. Erman Ayday for his support and help. His expertise and knowledge in privacy, security and many other fields not only helped me to complete this thesis, but made me a better academician. It would have been impossible to complete this work without him. He accepted me as an undergraduate researcher while I was a junior undergraduate student and had little knowledge about the field. I will be eternally grateful for his four years of guidance.

I would like to thank my jury members Prof. Özgür Ulusoy and Prof. Ali Aydın Selçuk for accepting to be in my thesis committee. I owe my special thanks to Prof. Ali Aydın Selçuk who was my Introduction to Cryptography course instructor. I discovered how interesting the field of security and privacy is in his course.

I would like to thank my friends and collegues in Bilkent. Firstly, I would like to thank Alper and Gizem for their good friendships. Then, I would like to thank Çağlar, Onur, Miray and Ömer for all the good times and coffee breaks. Apart from our everyday discussions, our discussions about our different fields in computer science taught me much. Their valuable feedbacks and comments were always helpful. I will miss spending time with them in the offices and corridors of Bilkent.

I would like to express my gratitude to Bilal, Buğra, Furkan, Melik and Yakup. Their support during my master studies was valuable. I want to thank my friends Kadir, Ahmed, Furkan I., İnci and Güntülü for their morale and support through this journey. I consider myself a very lucky person to call these people as my friends.

I would like to thank my family especially my parents. Their consistent support through my education beginning from the elementary school made me who I am today. I cannot repay to them for their effort for my education.

Lastly, I would like to thank Şafak. Without her help and love, I do not know if I can come this far. Her presence made everything easier.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, people's location data are collected easier than ever. Together with extensive usage of smart phones and apps, these location data are mostly collected by Location-based Service Providerss (LBSPs) and mobile telecommunication operators. Businesses often want to use this location data of their customers (or potential customers) to provide better service. For example, businesses can use their customers' location data to determine where to open a new branch. LBSPs are eager to share these data with the businesses for their own interest, however, it is not possible for LBSPs to share these data directly because of privacy reasons. If LBSP share the location data of their customers directly to the businesses, businesses can see whereabouts of the customers without the consent of the customers. If businesses asks for the location data of some specific customers, then LBSP's may infer that some customers registered with that business which is a privacy issue. Therefore privacy-preserving solutions are needed to improve their services without violating the privacy of LBSPs' customers. To address these issues, we propose new types of privacy-preserving queries between businesses and LBSPs and design such queries with three different cryptosystem while comparing them each other. Businesses require locations of their customers to optimize their location-based decisions. Most of the businesses already have some information about the customers such as their home addresses, but what they want to know

the whereabouts of their customers during certain hours. Mobile telecommunication operators or LBSPs such as Swarm usually have this kind of up-to-date location data.

In this thesis, we denote data owners as servers and businesses as clients. We focus on a particular problem in which a client is interested in finding the most optimal location to open a new facility (based on the whereabouts of its customers). For instance, consider a bank that has existing branches and want to open new branch. The bank can use the Reverse Nearest Neighbor Cardinality Query (RNNQ) to see how many of its customers are close to each of its existing facility, and decide where to open a new facility accordingly. In a non-privacy preserving setting for RNNQ, the client shares its user list along with the locations of its existing facilities (and locations of facilities that it plans to open). The server checks the whereabouts of client's customers and determines the nearest facility for each customer. Then, server sends the number of closest users to each facility (the query result) to client. In this setting, client's customers are exposed to the server. In addition, query result is exposed to the server as well. This is not desired since for example, the server may use the query result to inform other rival businesses. Thus, we consider query result as a sensitive information and it should stay hidden from the server. Both customer lists are sensitive information, and hence they should stay hidden to each parties.

We propose two new privacy-preserving methods to execute such location-based queries. Then, we compare these two methods with our existing previous work [1] which addresses the same problem and proposes a solution using homomorphic encryption (Paillier cryptosystem [2]). Our first proposed scheme uses Lattice-based cryptography and the second proposed scheme uses secure Multiparty computation (MPC) methods. Lattice-based cryptography is resilient against quantum computers which most of the famous cryptographic protocols, such as RSA, are vulnerable to the such attacks. Paillier cryptosystem and lattice cryptosystem both support homomorphic operations, meaning that one can perform mathematical operations on the encrypted data without the need for decryption. We discuss advantages and disadvantages of each method and compare them with each other. In our protocols, we hide customer lists of businesses from

LBSPs, locations of the customers from the businesses, and query result from LBSPs. In addition, we make the query result only available to the businesses and hide them from the LBSPs.

Our contribution in this work is as follows:

1. We propose a method which uses lattice cryptography. Lattice cryptosystems are resistant to post-quantum based attacks. Thus, we show how such a privacy-preserving scheme can be developed that is resilient against quantum computers. While doing so, we introduce negligible amount of extra overhead of memory usage and computation time compared to existing work [1].
2. We introduce a scheme that utilizes MPC when executing queries. The proposed scheme uses very little memory and its computational complexity is comparable to a non-privacy preserving setting.
3. We compare the proposed methods with the existing work and investigate their advantages and disadvantages. Furthermore, we discuss which scheme is more suitable to certain settings.

Rest of this paper is organized as follows: In chapter 2, we discuss related works and provide background information about the cryptosystems we used in this work. In chapter 3, we describe the system model. We describe the Paillier cryptosystem-based protocol in chapter 4, lattice-based protocol in chapter 5, and MPC-protocol in chapter 6. chapter 7 presents experiments and results. Finally conclusion and future work is in chapter 8.

Chapter 2

Related Works and Background

2.1 Related Works

Optimal location query finds the best possible location using the given facilities and customers while maximizing a specific property [3]. Property can be anything such as distance or time. Optimal location queries are useful especially for businesses. A business can execute queries to find optimal locations for their new facilities. Optimization criteria depends on the business' needs. For example, a business may looking for a location that is close to the majority of their customers. In other words, that business is trying to maximize influence over its customers. These kind of queries are based reverse nearest neighbors (RNNs) [4] queries. We use bichromatic version of RNN query which focuses on the nearest facility. We assume that each customer prefers his/her nearest facility.

In the area of privacy-preserving location query processing, several approaches exist. One approach proposes to anonymize the customer locations and execute queries on the cloaked customer locations [5, 6] which are based on k-anonymity model [7, 8]. There are also cryptographic approaches exist [9, 10]. There are also combined approaches. For example, [11] combines differential privacy with k-anonymity model.

2.2 Homomorphic Encryption

Homomorphic encryption is special type of encryption that allows one to perform computations on ciphertexts. For instance, multiplication of two ciphertexts is equal to addition of the plain-texts of that ciphertexts. Homomorphic encryption schemes that supports only limited type of operations are called Partially Homomorphic Encryption (PHE). PHE supports either addition or multiplication. Paillier cryptosystem [2] and ElGamal encryption [12] are examples of partially homomorphic encryption schemes. Fully Homomorphic Encryption (FHE) supports both multiplication and addition at the same time. Gentry's scheme [13] opened the way for FHE schemes. However, drawback of the FHE is its practicability. FHE schemes are slower compared to PHE schemes. There is also another type of homomorphic encryption called Somewhat Homomorphic Encryption (SWHE). SWHE is similar to FHE that is one can use both addition and multiplication. However, there are two main differences of SWHE compared to FHE. First difference is SWHE schemes are more efficient compared to the FHE schemes. Second difference is FHE schemes allows one perform unlimited number of operations over ciphertexts. In SWHE schemes, one can perform limited operations. If limited number operations are exceeded in SWHE schemes, ciphertext becomes corrupted and can never be decrypted. So in SWHE we trade performance for the usability. The Fan-Vercauteren cryptosystem (FV) cryptosystem [14] is an example to somewhat homomorphic encryption. Below, we will give the homomorphic encryptions we used in our protocols.

2.2.1 Paillier Cryptosystem

Paillier cryptosystem is PHE scheme which also uses public key cryptography. We use additive property of Paillier cryptosystem. Keys are generated by as follows: Two large primes, namely p and q, are chosen and let $n = pq$. We pick a random integer g which is in range of 0 to n^2. n and g are the public keys. λ is defined as $\lambda = lcm(p-1, q-1)$ where lcm stands for least common multiple. In addition, μ is defined as $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ where $L(x) = (x-1)n^{-1}$.

μ and λ are the private keys. Encryption and decryption are defined as follows:

$$E(m) = g^m r^n \bmod n^2 \; ; \; r \in \{1, 2, ..., n^2 - 1\} \tag{2.1}$$

$$D(c) = L(c^\lambda \bmod n^2).\mu \bmod n \tag{2.2}$$

Random number r ensures that encryption of same message will result a same ciphertext with very low probability. In addition r and n must be co-prime. Paillier cryptosystem allows one to multiply the ciphertext and obtain its addition in plaintext domain. Additive property is defined in Eq. (2.3) below.

$$E(x).E(y) = E(x + y) \tag{2.3}$$

Formally, $E(x).E(y) = g^{x+y}(r_x + r_y)^n = E(x + y)$. We can use this property to multiply ciphertexts with constants.

$$E(x)^k = E(x.k) \tag{2.4}$$

Paillier cryptosystem can be used in many areas from e-voting [15] to genomic privacy [16].

2.2.2 Somewhat Homomorphic Encryption

Gentry's Ph.D. thesis [13] was first of its kind and it introduced first ever FHE scheme. Gentry's FHE scheme supports both addition and multiplication of the ciphertexts. However, its performance is not efficient in terms of both memory and computation time. SWHE is kind of a FHE scheme but SWHE is more practical and efficient compared to the FHE. One can use both additions and multiplications in SWHE but number of operations are limited. Each operation on a ciphertext adds a 'noise' to that ciphertext. If a noise threshold of the ciphertext is exceeded, then ciphertext become corrupted and cannot be decrypted again. Noise threshold of the ciphertext depends on the security parameters of the scheme such as ciphertext size. Generally, addition increases noise little compared to the multiplication of two ciphertexts. Most of the SWHE schemes [14, 17, 18] are based on Learning with Errors problem [19] for their hardness assumption. Therefore, using SWHE is much more preferable to FHE for real life applications.

2.3 Multiparty Computation

MPC is a powerful tool which allows parties to make desired computation jointly without need of a trusted third party. First two-party computation is proposed by Yao in 1982 [20] which is secure against semi-honest attacker. In our MPC based protocols, parties do not reveal their private data but reveal the parts of their private data. In other words, if there are n parties, each user first splits their secret data x as $x = x^{(1)} + x^{(2)} + ... + x^{(n)}$ randomly and sends the parts to the corresponding parties. Each party make their own computation using the parts they own. In this way, each party contributes to the end result without revealing their private data. Multiparty computation is more efficient compared to the homomorphic encryption schemes, due to there are no costly operations as encryptions/decryptions. Data are not stored in huge ciphertexts. In addition, instead of homomorphic operations over the ciphertexts, multiparty computations are simple floating number computations.

A simple MPC addition example is as follows:

There are 3 parties, namely P_1, P_2 and P_3, and they have their secret values x, y and z respectively. Each party randomly splits their secret data to three data parts such that $x = x^{(1)} + x^{(2)} + x^{(3)}$, $y = y^{(1)} + y^{(2)} + y^{(3)}$ and $z = z^{(1)} + z^{(2)} + z^{(3)}$. Then, each party sends the data shares to the corresponding party, i.e P_1 gets $x^{(1)}$, $y^{(1)}$ and $z^{(1)}$, P_2 gets $x^{(2)}$, $y^{(2)}$ and $z^{(2)}$ etc. After the data distribution, each party adds their data parts. P_1 computes $R^{(1)} = x^{(1)} + y^{(1)} + z^{(1)}$, P_2 computes $R^{(2)} = x^{(2)} + y^{(2)} + z^{(2)}$ and P_3 computes $R^{(3)} = x^{(3)} + y^{(3)} + z^{(3)}$. Parties later later share or broadcast their intermediate results, $R^{(1)}$, $R^{(2)}$, $R^{(3)}$. All parties add the intermediate results and obtain the final result since $R^{(1)} + R^{(2)} + R^{(3)} = x^{(1)} + y^{(1)} + z^{(1)} + x^{(2)} + y^{(2)} + z^{(2)} + x^{(3)} + y^{(3)} + z^{(3)} = x + y + z$. Parties jointly compute the desired sum without showing their secret values to the each party.

In our MPC based protocols, we use secure two party multiparty multiplication which is more complex then simple addition. For multiplication, we use Beaver's multiplication triplets [21].

Chapter 3

System Model

We define client C as businesses who wants to execute aggregate queries. LBSPs who have the location data of their customers are defined as server S. We call customers of the server and client as the users. We define the set U_c as the user ids of the client and U_s as the user ids of server. We need an identifier to evaluate our queries and identify the customers uniquely in the user sets. Client wants to identify its customers in server since not all users of U_c and U_s are overlap. We define their common users, i.e. the intersection of two user sets as U_I. Parties agree upon an identifier before the queries. Identifier may be telephone numbers or national identification numbers or social security numbers. Businesses collect this kind of information from their customers as well as LBSPs. Lastly, we define superset U which has the all possible ids for selected identifier. For example, if the identifier is selected as Social Security Number (SSN), we have 9 digits and 1 billion (10^9) possible ids. Thus U consist all the possible social security numbers. To clarify the user sets and role of identifier let's give an example: Assume that a person with a social security number of 35 is customer of client but not a customer of server. Then U_c will include 35 but U_s will not include 35 since customer with SSN is not the server's customer. In this example 35 will be a element of superset U since it is a valid SSN. With right identifier, we can uniquely identify the users in user sets. Choosing an identifier will play crucial part in our protocols. We will return why we define such a superset later. We

show the overall system model on the Fig 3.1.

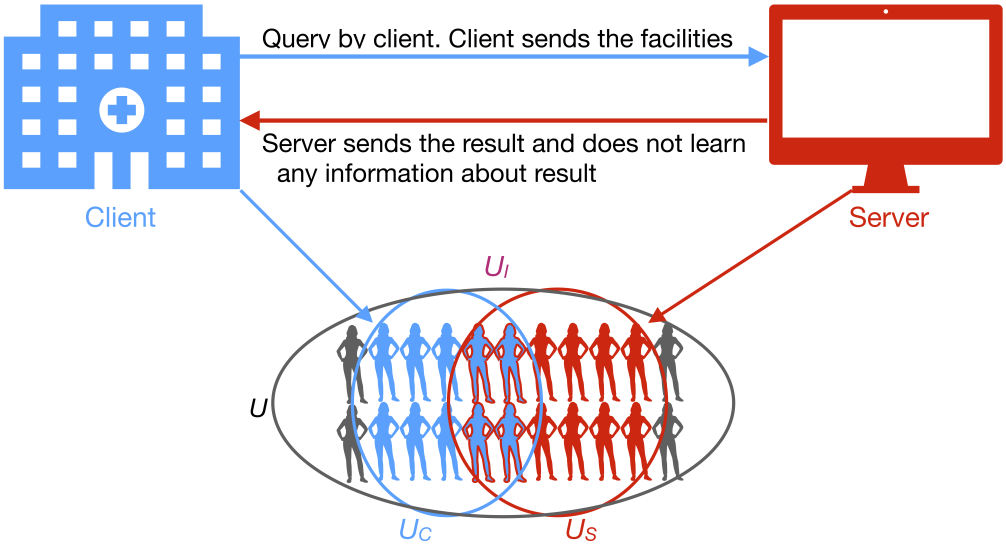

Figure 3.1: System Model

We denote the facilities by F. Existing facility locations are public since businesses' existing facility locations are known by all. Client wants to execute aggregate queries using its users location data. However, client does not have a such data. Server holds the location data of its own users. If some users of client is also users of server, then we have the location data. So, the client is interested in users in U_I. Challenge is to prevent server from learn anything about the U_c and client from learn anything about U_s while executing queries. Homomorphic encryption and MPC ensures that the private data is protected.

3.1 Threat Model

Our system is secure in semi-honest, or honest-but-curios, model. In other words, both server and client follow the protocol correctly, but they may try to analyze the data and learn some information about the users or query result. Both

parties correctly provide inputs, do the computations as stated and output the correct results. They may not, for example, do arbitrary computations during the query or insert some other input to the query. However, there might be some malicious people in both parties and they might to try learn some information. For instance, person in server might want to learn the query result and sell this result to businesses' rivals. Client wants to get the correct result in order to serve their customers better and server knows that correct result crucial because otherwise client may stop using the services of server. For these reasons, we believe semi-honest threat model is acceptable and realistic.

In all of three protocols, we aim to protect the same private data of the parties. We can list the private data as follows:

1. User lists U_c and U_s. Both parties should not know whether any specific user i exist in the opposite party. In other words, C should not understand in any way whether $i \in U_s$ or S should not understand whether $i \in U_c$. All users should stay anonymous.
2. Locations of users in U_s should be hidden away from C
3. Query result should stay hidden from S.

We assume that, during the protocols, communication channel is secure and encrypted against the eavesdroppers. Table 3.1 includes the symbols that are used within our protocols.

3.2 Query Types

We have two different aggregate queries for our protocols which both of them uses user location data. We use the same queries in our previous work [1].

Table 3.1: Symbols and Notations

Symbol	Description
PK	Public key of C
SK	Private key of C
$E(m)$	Encryption of message m using PK
$[x]$	Encrypted message x
$D(c)$	Decryption of a ciphertext c
U_c	User list of C
U_s	Users list of S
U	User superset i.e all possible users
U_I	$U_c \cap U_s$
F	Facilities send by client during query
T	Existence Array of client
Y	Existence Array of Server
Λ	Randomized existence array of C
Γ	Randomized existence array of S
$q_{j,i}$	Intermediate result parts of C
$z_{j,i}$	Intermediate result parts of S
x_j	Result parts of C
v_j	Result parts of S
n_c	Number of users in U_c
n_s	Number of users in U_s
n_u	Number of users in U
n_I	Number of users in U_I
n_t	Number of triplets
$d(u, f)$, $d_{u,f}$	Distance between user u and facilitiy f
k	Total number of facilities in the query, $\|F\|$
a, b and c	Pre-computed triplets in the form of $c = ab$

3.2.1 RNN Cardinality Query

Aim of the RNNQ is to distribute the users to the facilities equally so that all facilities work evenly. Businesses, such as banks, want to open new branch of their facilities in places where their customers densely exist. This will make sure that the new branch is closer to customers and at the same time reduce the workload of the existing facilities. Formally, we can define RNNQ as: Given users, U_c and U_s, and facilities F, find the total number of users that is closest to each facility. In other words, for each facility, find the total number of users that is closest to that facility. Client can run RNNQ to find the distribution of their users in its existing facilities or it can run RNNQ to find the distribution of users in new planned facilities.

3.2.2 Average Distance Query

Another optimal location query is Average Distance Query (AVGQ). Businesses may want to learn and minimize the average distance of their customers to each ones closest facility. In other words, AVGQ finds the average distance between users in U_I and their closest facilities in F. Businesses can run AVGQ to reduce the distances of their customers to their facilities. In this way, businesses may want to make their facilities more attractive to their users.

Chapter 4

Paillier Cryptosystem Based Protocols

In this chapter, we present our previously proposed scheme [1]. We include this chapter for comparison with other schemes since this is our base scheme. In addition, Paillier based protocols are similar to the lattice and MPC based protocols. We have also demonstrated the practicability of this scheme in our demo paper [22]. We use Paillier cryptosystem in these schemes where we especially utilize additive homomorphic property of Paillier i.e $E(x).E(y) = E(x+y)$, (2.3). We use client based protocols from [1] because it is shown that client based protocols are more efficient in terms of communication and computation efficiency.

Before any query execution, there is one time setup phase which needs to be completed. In setup phase, client generates PK and SK, then sends PK to server. Client also decides the superset by choosing the identifier and then creates the encrypted existence array $[T] = \{[t_1], [t_2], ..., [t_{n_u}]\}$ where n_u is total number of users in U. For every user $i \in U$ client calculates the following: If $i \in U_c$ then $[t_i] = E(1)$ otherwise $[t_i] = E(0)$. In other words, client puts encrypted 1 to the existence array's i'th location if that user i is also clients' user. For example, let's consider the identifier as phone numbers. If the client has a customer with phone number x then $[t_x] = E(1)$. Since every encryption Paillier cryptosystem

has random number in it, existence array will contain the same value with very low possibility. Proper superset should be chosen to ensure that U hides the U_c. Client sends the encrypted existence array $[T]$ to the server.

Paillier based protocols can be summarized in 6 steps. In the first step, client sends the query request along with facilities, F, which is used by server for optimal location queries. In second step, server calculates distances between its users U_s and F. In third step, server calculates the query result. In fourth step, server masks the result. In fifth step, server sends the result to the client. In final step, client decrypts the result and obtains the query result. We describe the protocols in detail below. Figure 5.1 is the overview of the protocols.

4.1 RNN Cardinality Query

Objective of RNNQ is to find the total number of users closest to each facility. Below, we explain each step in detail. Step numbers are matched with protocol overview in fig. 5.1

1. Client, C, sends the query request along with the locations $F = \{f_1, f_2, ..., f_k\}$.
2. Server, S, calculates the distance between each user and the facilities. S decides the each user's closest facility.
3. S calculates the result ciphertexts $[X] = \{[x_1], [x_2], ..., [x_k]\}$. For every facility, S multiplies existence array values which their closest facility is the same. In other words, if i'th user's closest facility is f_j, then $[x_j] = [x_j].[t_i]$. Note that $[t_i] = E(1)$ if user $i \in U_c$ and $[t_i] = E(0)$ if user $i \notin U_c$. S only multiplies its own users existence array values, $[t_i]$'s. Further, if $[t_i]$ is also the user of C then it is $E(1)$ thus by Eq. (2.3), S adds 1 to the results $[x_j]$. If user i is not user of U_c then $[t_i] = E(0)$, basically S adds 0 to the result. By definition of the user sets we successfully count the number of elements in U_I.

4. S encrypts k zeros and them multiply them with results. For every $[x_i] \in [X]$, S does $[x_i] = [x_i].E(0)$. This will not alter the results X since multiplying with encrypted 0 means adding with 0.

5. Server sends the results $[X]$ to the client.

6. Client decrypts $D([X]) = \{D([x_1]), D([x_2]), ..., D([x_k])\}$ obtains the query result.

4.2 Average Distance Query

In AVGQ, client aims to find the average distance of their users to each ones nearest facility. In Figure 4.1, we give an example to this process.

1. Client, C, sends the query request along with the locations $F = \{f_1, f_2, ..., f_k\}$.

2. Server, S, calculates the the distance between each user $i \in U_s$ and the facilities and decides the user's closest facility, f_j. In addition, S holds the distances between users and their closest facility, $d_{i,j}$. Note that $d_{i,j}$ means that distance between user i and facility f_j.

3. S calculates the two result ciphertexts $[X] = \{[x_1], [x_2]\}$. Initially, both result ciphertexts are encrypted zeros. For every user $i \in U_s$, server first raises its existence array value with user i's distance to his/her closest facility and then it multiplies with $[x_1]$. In other words, if i'th user's closest facility is f_j and his/her distance to it is $d_{i,j}$, then $[x_1] = [x_1].[t_i]^{d_{i,j}}$. We can summarize the operation as $[x_1] = \sum_{i \in U_s} [t_i]^{d_{i,j}}$ where f_j is the i's nearest facility. Note that $[t_i] = E(1)$ if user $i \in U_c$ and $[t_i] = E(0)$ otherwise. Hence, if user exist in U_I we multiply 1 with $d_{i,j}$ then add it to the result by Eq. (2.3) and Eq. (2.4) otherwise we multiply 0 with the user's distance and add it. Thus users that are not in U_I does not included in result. $[x_1]$ becomes addition of the distances of the users to their closest facility. We can call $[x_1]$ be as

encrypted *sum*. In addition, server calculates total number of user in U_I, n_I, by $[x_2] = \prod[t_i]$ which $i \in U_s$ using the property of (2.3). We can call $[x_2]$ as encrypted n_I.

4. S encrypts zeros and then multiply them with result ciphertexts. $[x_1] = [x_1].E(0)$ and $[x_2] = [x_2].E(0)$

5. Server sends the result $[X]$ to the client.

6. Client decrypts $D([X]) = \{D([x_1], D([x_2])\}$ and obtains total sum sum and n_I. Average distance is $avg = sum/n_I$

User *i*	$[t_i]$
1	E(0)
2	E(1)
3	E(0)
4	E(1)
5	E(1)
6	E(0)
7	E(1)
8	E(0)
9	E(0)
10	E(1)

U	U_c	U_s	U_I
1,2,3,4,5,6,7,8,9,10	2,4,5,7,10	1,2,4,6,7,9,10	2,4,7,10

n_U	n_c	n_s	n_I
10	5	7	4

Step 2

User *i*	1	2	4	6	7	9	10
Closest Facility	f1	f2	f2	f1	f1	f1	f2
Distance	10	29	17	22	61	43	23

Step 3

$[X_1] = [t_1]^{10}. [t_2]^{29} . [t_4]^{17} . [t_6]^{22} . [t_7]^{61} . [t_9]^{43} . [t_{10}]^{23} = E(0)^{10}. E(1)^{29} . E(1)^{17} . E(0)^{22} . E(1)^{61} . E(0)^{43} . E(1)^{23} = E(128)$
$[X_2]=[t_1]. [t_2] . [t_4] . [t_6] . [t_7] . [t_9] . [t_{10}] = E(0) . E(1) . E(1) . E(0) . E(1) . E(0) . E(1) = E(4)$

Figure 4.1: An example run of AVGQ. From left top to right botton clockwise: Existence array created by C and sent to S in setup phase. User sets and their number of users. Step 2 of the protocol that is U_s's closest facility and their distance calculated. Step 3, S multiplies the $[t_i]^{d_{i,f}}$ values and $[t_i]$ values which $i \in U_s$ to obtain encrypted sum and n_I

Chapter 5

Lattice Based Protocols

In this chapter, we propose protocols for our aggregate optimal location queries using lattice based cryptography. As shown in our previous works, our current protocols can be implemented by using any simple additive homomorphic scheme. However, for complex queries, additive homomorphic property may not be enough. In addition, additive homomorphic property may not be practical to design such complex future queries. For example, one such query is filtering the users based on features such as age. For this query, client must prepare another existence array according to the ages. Another possible problem with the Paillier cryptosystem is, one needs to raise the ciphertext to constant d to multiply plaintext with d, $E(x)^d = E(x.d)$. This operation may become a costly operation since ciphertexts are large numbers and d may be a large number too in some cases. In addition, Paillier cryptosystem is not a post-quantum resistant scheme. For our future works, where we plan to extend queries to more complex structure and more suitable to real life applications, Paillier cryptosystem may not a suitable for these drawbacks. Instead of Paillier cryptosystem, one can use fully or somewhat homomorphic encryption schemes which are more flexible in terms of computation type. For these reasons, we design and implement the queries using SWHE scheme to see whether they are pratical and feasible or not. We use the FV cryptosystem for our queries [14].

We use basically two functions from FV scheme namely $add(c_1, c_2)$ and $plain\_mult(c_1, d)$. $add(c_1, c_2)$ function takes two ciphertexts, c_1 and c_2, adds them and put the resulted sum to c_1. $plain\_mult(c_1, d)$ takes ciphertext c_1 and constant rational number d, multiplies them and put the result to c_1.

Lattice based protocols are almost the same as Paillier based protocols. All the operations are and order of the steps are same. Before any query, setup phase must be completed in lattice based protocols as well. C creates the keys and sends PK to S. C also generates existence array $[T] = \{[t_1], [t_2], ..., [t_{n_u}]\}$ as follows: If $i \in U_c$ then $[t_i] = E(1)$ otherwise $[t_i] = E(0)$. In other words, client puts $E(1)$ to the existence array's i'th location if that user i is also clients' user. For example, let's consider the identifier as phone numbers. If the client has a customer with phone number x then $[t_x] = E(1)$. C sends the existence array to S. Proper superset should be chosen to ensure that U hides the U_c.

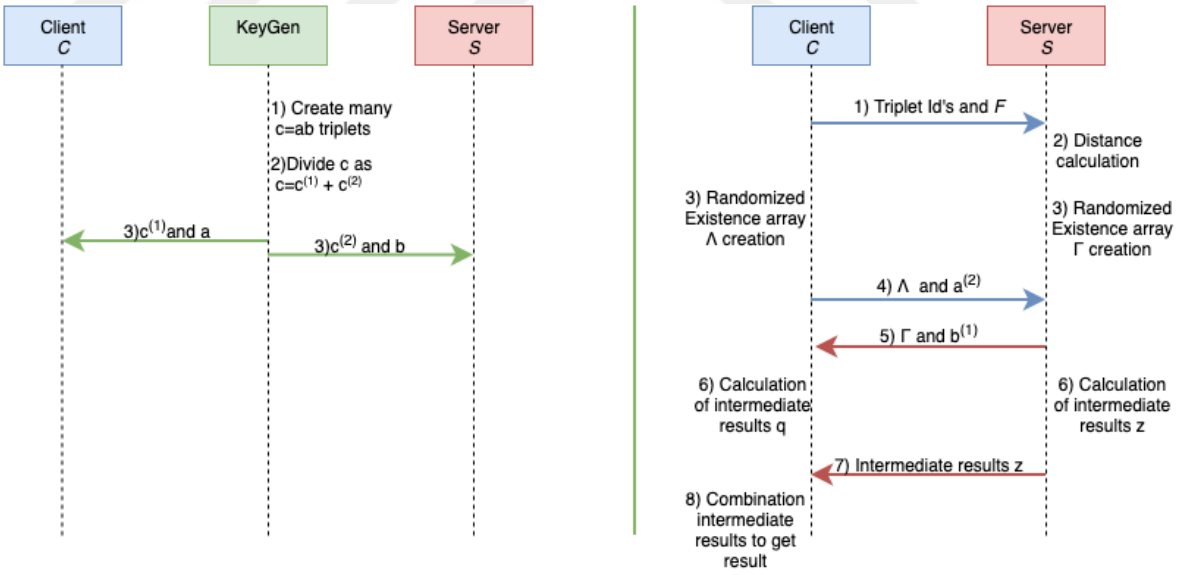

Figure 5.1: Overview of the setup phase on left. Overview of the lattice and Paillier based protocols are on right

We have 6 steps in lattice based protocols. Steps 1 and 5 are communication steps. In step 2, S determines closest facilities of users and their distance to facilities. In step 3, encrypted results are calculated. In step 4, masking occurs. In step 6, C decrypts the encrypted result and obtain query result. Figure 5.1 summarizes the protocols.

5.1 RNN Cardinality Query

Skeleton of lattice based RNNQ is same with Paillier based RNNQ. Instead of (2.3), lattice based protocols use $add(c_1, c_2)$ function here.

1. Client, C, sends the query request with the locations $F = \{f_1, f_2, ..., f_k\}$.
2. Server, S, calculates the the distance between each user and the facilities and decides the user's closest facility.
3. S calculates the result ciphertexts $[X] = \{[x_1], [x_2], ..., [x_k]\}$. Initially all values of $[X]$ is encrypted zeros. For every facility, S adds existence array values which their closest facility is the same. Formally, for all $i \in U_s$ that i's closest facility is f_j, S calls $add([x_j], [t_i])$. Basically S counts the the total users who are closest to f_j.
4. S encrypts k zeros using PK of C then calls $add([x_j], E(0))$ for all facilities.
5. Server sends the result $[X]$ to the client.
6. Client decrypts $D([X]) = \{D([x_1]), D([x_2]), ..., D([x_k])\}$ obtains the query result.

5.2 Average Distance Query

We use $plain\_mult(c_1, d)$ function here instead of (2.4) compared to the Paillier based AVGQ.

1. Client, C, sends the query request with the locations $F = \{f_1, f_2, ..., f_k\}$.
2. Server, S, calculates the the distance between each user $i \in U_s$ and the facilities and decides the user's closest facility, f_j also holds the distances of users, d_{ij}.

3. S calculates the 2 result ciphertexts $[X] = \{[x_1], [x_2]\}$. Initially both result ciphertexts are equal to $E(0)$. For every user $i \in U_s$, server first multiply $d_{i,j}$ with its existence array value with then adds it to the $[x_1]$. In other words, $[x_1] = add([x_1], plain\_mult([t_i]d_{i,j}))$ where $i \in U_s$ and f_j is the i'th users closest facility. For all $i \in U_s$, S executes first $plain\_mult([t_i], d)$ and then $add([x_1], [t_i])$. Let's call $[x_1]$ as encrypted sum. Further, we need to calculate n_I since sum is only the total distance of users in U_I. S calculates $[x_2]$ as adding up all its users' existence array values. $[x_2] = add([x_2], [t_i]$ where $i \in U_s$.

4. S encrypts zeros with C's PK and then adds them to the results. In other words, S does $add([x_1], E(0))$ and $add([x_2], E(0))$ operations.

5. Server sends the result $[X]$ to the C.

6. Client decrypts $D([X]) = \{D([x_1], D([x_2])\}$ and obtains total sum sum and n_I. Average distance is $avg = sum/n_I$

Chapter 6

Multiparty Computation based Protocols

In this chapter, we propose protocols for the aggregate optimal location queries without any encryption/decryption and at the same time protecting the privacy of the parties and their customers. There are no encryptions/decryptions, hence the absence of ciphertexts saves time and memory. Our MPC scheme hides the users by one-time pad style masking which is provably secure. Since we are only interested in users in U_I, we can use similar idea to the other protocols. Each party creates its own existence array(vector). Inner product of the these two vectors gives us the U_I. In other words, we conduct set intersection operation. Base of our protocols in the MPC based protocols is this inner product idea. Our multiparty multiplication scheme is based on multiplication scheme in [23]. We will multiply these two vector in a private manner. For RNNQ, both C and S have existence arrays $T = \{t_1, t_2, ..., t_{n_c}\}$ and $Y = \{y_{1,1}, y_{1,2}, ..., y_{k,n_s}\}$ respectively which is constructed by the same rules as in previous protocols that is $t_i = 1$ if $i \in U_c$, $t_i = 0$ if $i \notin U_c$. In other words, T specifies users of C in the superset, whether they are the customers of C or not. S creates its own existence array Y in MPC based protocols. However, S create existence array for all k facilities meaning that $y_{j,i} = 1$ if $i \in U_c$ and also i's closest facility is f_j. Otherwise, $y_{j,i} = 0$. We can think Y as a k parallel existence arrays, each for indicating

closest users to different facility. In other words, server creates existence array for each facility. For AVGQ, Y is constructed as T constructed since we do not need to count for each facility. Thus for AVGQ, Y is defined as, $y_i = 1$ if $i \in U_s$, otherwise $y_i = 0$

In our protocols, there is no third party in computation part. However, for once, triplet generation is needed before any query can begin. In setup phase, pre-computation phase, a semi-trusted third party, as we call generator, generates many three variables which are a, b and c. These triplets are generated in the form of $c = a.b$ [21]. Generator generates sufficiently large amount, n_t, of triplets that is $n_t >> n_u$. Finally, generator distributes these triplets to client and server as follows: Each c is divided into 2 parts randomly such that $c = c^{(1)} + c^{(2)}$. C receives $c^{(1)}$ and a. S receives $c^{(2)}$ and b. Neither client or server cannot recover c since client lacks either needs b or other part of c. Similarly, server needs a or $c^{(1)}$ to recover c. Thus b is secret to C and a is secret to S. c is secret to all parties. After this setup phase, query execution is ready.

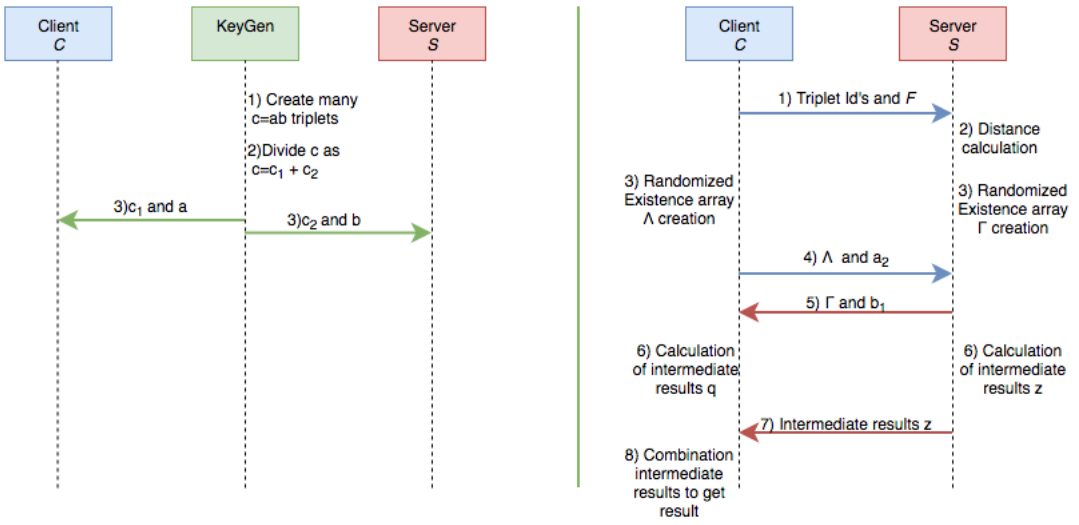

Figure 6.1: Overview of the MPC based protocols. Setup phase is shown in left. Overview and steps of the MPC based protocols are on right.

In our protocols, we use secure multiparty multiplication and addition. We define equations for our computations.

$$c = a.b \tag{6.1}$$

$$c = c^{(1)} + c^{(2)} \tag{6.2}$$

$$a = a^{(1)} + a^{(2)} \tag{6.3}$$

$$b = b^{(1)} + b^{(2)} \tag{6.4}$$

$$\lambda = t - a \tag{6.5}$$

$$\gamma = y - b \tag{6.6}$$

$$t.y = c + \lambda.b + \gamma.a + \lambda.\gamma \tag{6.7}$$

We have 8 steps in multiparty computation based protocols. In first step, C picks the n_u triplets from pre-computed n_t triplets and send their id's to the S along with facilities and query request. In step 2, distance calculation and determining the closest facility takes place. In step 3, C and S calculates Λ and Γ respectively also splits their a and b's randomly. In step 4 and 5, C and S sends the corresponding data which they calculated in step 3. In step 6, result shares are calculated. In step 7, S sends its share of query result to C. In the last step, C combines the result shares and obtains the result. Fig. 6.1 shows the overview of setup phase and protocols. Fig. 6.2 shows an example of the RNNQ run.

6.1 RNN Cardinality Query

1. Client C picks $k.n_u$ triplets from already received triplets. For every member of the existence array, we need different triplets in order to the achieve privacy. Therefore, we need triplets as large as superset size. C has only part of the triplets, a and $c^{(1)}$, hence S must choose the corresponding part of the triplets. C sends the id's of triplets as well as the facility locations F and query request.

2. S determines the each user's closest facility. Remember that S can only calculate the users in U_s.

3. C calculates Λ as for all $i \in U$, $\lambda_{j,i} = t_i - a_{j,i}$. $a_{j,i}$ comes from pre-computed triplets. Note that, $t_i = 1$ if $i \in U_c$ and 0 otherwise. In addition, C splits every a randomly such that $a = a^{(1)} + a^{(2)}$. Similarly S calculates Γ as for all $i \in U$ and $j \in F$, $\gamma_{j,i} = y_{j,i} - b_{j,i}$. Remember that, $y_{j,i} = 1$ if $i \in U_s$ and j is the closest facility of user i. Also S randomly splits every b as $b = b^{(1)} + b^{(2)}$. Note that $a_{j,i}$ pairs with $b_{j,i}$ to complete the triplet $c = ab$.

4. C sends $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_{n_u}\}$ and all $a^{(2)}$'s to S.

5. S send $\Gamma = \{\gamma_{1,1}, \gamma_{1,2}, ..., \gamma_{k,n_u}\}$ and all $b^{(1)}$'s to C. After this step, S has $c^{(2)}$, Λ, Γ, $a^{(2)}$ and b. Similarly, C has $c^{(1)}$, Λ, Γ, $b^{(1)}$ and a.

6. Both parties calculate the part of the result. C calculates the intermediate result parts for all $i \in U$ and $j \in F$, $q_{ji} = c^{(1)}_{j,i} + \lambda_{j,i}.b^{(1)}_{j,i} + \gamma_{j,i}.a^{(1)}_{j,i}$. Note that, $q_{j,i}$ is the half of the Eq. (6.7), not all of the equation. Other part of Eq. (6.7) will be calculated by S. After all $q_{k,i}$'s are calculated C adds them all according to their facilities. $x_j = \sum_{i \in U} q_{j,i}$ where $j \in F$. By summing the intermediate result parts, C obtains k different result parts. In addition, summing the intermediate result parts, $q_{j,i}$, ensures that we anonymize the users results and obtain only aggregate information about them. Similarly S does the exact same thing but with its parts of the data. For all $i \in U$ and $j \in F$, $z_{j,i} = c^{(2)}_{j,i} + \lambda_i.b^{(2)}_{j,i} + \gamma_{j,i}.a^{(2)}_{j,i} + \lambda_{j,i}.\gamma_{j,i}$. After all intermediate result shares are calculated, S calculates $v_j = \sum_{i \in U} z_{j,i}$ where $j \in F$. Let $X = \{x_1, x_2, ..., x_k\}$ and $V = \{z_1, z_2, ..., z_k\}$.

7. S sends the V to C. C does not sends it result parts since we do not want S to learn query results.

8. C adds x_j with v_j to obtain total number of closest user to F_j.

6.1.1 Proof of RNNQ

We will prove the security and correctness of RNNQ. We start with the correctness then proceed with the security.

Correctness of RNNQ

We first show that intermediate result parts are actually multiplication of $y_{j,i}$ and t_i. Remember, $y_{j,i}.t_i$ determines that whether user i's closest facility is j. If $y_{j,i}.t_i = 1$, then user i is in U_I and its closes facility is j. Thus objective is obtaining $y_{j,i}.t_i$. It can be seen that $q_{j,i} + z_{j,i} = c_{j,i}^{(1)} + \lambda_{j,i}.b_{j,i}^{(1)} + \gamma_{j,i}.a_{j,i}^{(1)} + c_{j,i}^{(2)} + \lambda_{j,i}.b_{j,i}^{(2)} + \gamma_{j,i}.a_{j,i}^{(2)} + \lambda_{j,i}.\gamma_{j,i} = c_{j,i}^{(1)} + c_{j,i}^{(2)} + \lambda_{j,i}.b_{j,i}^{(1)} + \lambda_{j,i}.b_{j,i}^{(2)} + \gamma_{j,i}.a_{j,i}^{(2)} + \gamma_{j,i}.a_{j,i}^{(2)} + \lambda_{j,i}.\gamma_{j,i}$. Clearly, $\lambda_{j,i}.b_{j,i}^{(1)} + \lambda_{j,i}.b_{j,i}^{(2)} = \lambda_{j,i}(b_{j,i}^{(1)} + b_{j,i}^{(2)}) = \lambda_{j,i}.b_{j,i}$ using (6.4), $\gamma_{j,i}.a_{j,i}^{(1)} + \gamma_{j,i}.a_{j,i}^{(2)} = \gamma_{j,i}(a_{j,i}^{(1)} + a_{j,i}^{(2)}) = \gamma_{j,i}.a_{j,i}$ using (6.3) and $c_{i,j}^{(1)} + c_{i,j}^{(2)} = c_{i,j}$ using (6.1) to simplify the equation. We get $q_{j,i} + z_{j,i} = c_{j,i} + \lambda_{j,i}.b_{j,i} + \gamma_{j,i}.a_{j,i} + \lambda_{j,i}.\gamma_{j,i}$ which is our multiplication equation (6.7). If we decompose λ, γ and c by (6.5), (6.6) and (6.1), we get $q_{j,i} + z_{j,i} = a_{j,i}.b_{j,i} + (t_i.b_{j,i} - a_{j,i}.b_{j,i}) + (y_{j,i}.a_{j,i} - a_{j,i}.b_{j,i}) + (t_i.y_{j,i} - t_i.b_{j,i} - y_{j,i}.a_{j,i} + a_{j,i}.b_{j,i})$. We simplify the equation and get $q_{j,i} + z_{j,i} = t_i.y_{j,i}$ which is indeed the multiplication of two entries of existence array.

In step 6, C adds all of its parts, $q_{j,i}$, according to their facility and S does the same thing with its parts, $z_{j,i}$. At step 8, C adds this parts to combine the shares to obtain the result. For example, for facility j, result is $x_j + v_j = q_{j,1} + z_{j,1} + q_{j,2} + z_{j,2} + ... + q_{j,n_u} + z_{j,n_u}$. We previously show that $q_{j,i} + z_{j,i} = t_i.y_{j,i}$. Thus $x_j + v_j = t_1.y_{j,1} + t_2.y_{j,2} + ... + t_{n_u}.y_{j,n_u}$ where $t_i.y_{j,i}$ is 1 if i's closest facility is j and $i \in U_I$. Hence we count the total number of users of both client and server whose closest facility is j.

Security of RNNQ

We show the security of RNNQ by investigating each data part. First we investigate $\lambda_{j,i}$ and $\gamma_{j,i}$ since they are directly shared with other party. S cannot recover t_i and $a_{j,i}$ from $\lambda_{j,i}$ since $\lambda_{j,i} = t_i - a_{j,i}$ which S does not know any of the variables. C cannot recover any data form $\gamma_{j,i}$ too since it has the same characteristic. For

every t_i and $y_{j,i}$, there is a distinct $a_{j,i}$ and $b_{j,i}$ respectively. Thus (6.5) and (6.6) works like a one time pad. C shares $a_{j,i}^{(2)}$ with S at step 4. S now has the $a_{j,i}^{(1)}$, $b_{j,i}$ and $c_{j,i}^{(1)}$. Clearly, S cannot recover $a_{j,i}$ since other part is not known to it. Same applies for $c_{j,i}$ as well. Thus, S cannot recover $c_{j,i} = a_{j,i}.b_{j,i}$. Same thing applies to C too it has the opposite parts. Thus, C cannot recover $c_{j,i} = a_{j,i}.b_{j,i}$ too. Since $q_{j,i}$ and $z_{j,i}$ calculated with these data parts, $q_{j,i}$ and $z_{j,i}$ are perfectly private. Directly adding $q_{j,i}$ and $z_{j,i}$ may be problematic since it gives the the direct multiplication of t_i and $y_{j,i}$. Parties knows their value and from the multiplication, they can infer the other value. For example, let $t_i.y_{j,i} = 1$ which means both variables are 1. C knows its own value which is $t_i = 1$ from the result C can infer that $y_{j,i} = 1$. However, we only need aggregate information about the users thus adding intermediate result parts, $q_{j,i}$ and $z_{j,i}$, hides this sensitive information. Therefore x_j and v_j is remains private. Finally, without x_j, S cannot recover the query results.

6.2 Average Distance Query

1. Client C picks $2n_u$ different triplets. C sends the id's of triplets with the facility locations F and query request.

2. S determines the each user's closest facility with their distance. Remember that S can only calculate the users in U_s.

3. C calculates Λ as for all $i \in U$, $\lambda_{1,i} = t_i - a_{1,i}$ and $\lambda_{2,i} = t_i - a_{2,i}$. In addition, C splits every a randomly as $a = a^{(1)} + a^{(2)}$. Similarly S calculates Γ as for all $i \in U$ and $j \in F$, $\gamma_{1,i} = y_i.d_{i,j} - b_{1,i}$ and $\gamma_{2,i} = y_i - b_{2,i}$. Also S randomly splits every b as $b = b^{(1)} + b^{(2)}$. Note that $a_{1,i}$ pairs with $b_{1,i}$ to complete the triplet $c = ab$.

4. C sends Λ and all $a^{(2)}$'s to S.

5. S send Γ and all $b^{(1)}$'s to C.

6. Both parties calculate the parts of the result. C calculates the intermediate result parts for all $i \in U$, $q_{1,i} = c_{1,i}^{(1)} + \lambda_{1,i}.b_{1,i}^{(1)} + \gamma_{1,i}.a_{1,i}^{(1)}$ and $q_{2,i} = c_{2,i}^{(1)} +$

$\lambda_{2,i}.b_{2,i}^{(2)} + \gamma_{2,i}.a_{2,i}^{(1)}$. Note that client computes two intermediate result parts namely $q_{1,i}$'s and $q_{2,i}$'s which are sum and n_I respectively. Client lastly computes the aggregate result by $x_1 = \sum_{i \in U} q_{1,i}$ and $x_2 = \sum_{i \in U} q_{2,i}$. Server S calculates, for all $i \in U$, $z_{1,i} = c_{1,i}^{(2)} + \lambda_{1,i}.b_{1,i}^{(2)} + \gamma_{1,i}.a_{1,i}^{(2)} + \lambda_{1,i}.\gamma_{1,i}$ and $z_{2,i} = c_{2,i}^{(2)} + \lambda_{2,i}.b_{2,i}^{(2)} + \gamma_{2,i}.a_{2,i}^{(2)} + \lambda_{2,i}.\gamma_{2,i}$. After all intermediate result parts are calculated, S calculates the aggregate results $v_l = \sum_{i \in U} z_{l,i}$ and $v_2 = \sum_{i \in U} z_{2,i}$ which z_1 is the sum of all distances of users to their closest facility sum and z_2 is n_I.

7. S sends the v_1 and v_2 to C. C does not sends its result parts since C does not want S to learn query the results.

8. C adds x_1 with v_1 to obtain total distance of all users to their closest facility, sum. It then adds x_2 with v_2 to obtain n_I. It gets the average by $avg = sum/n_I$

6.2.1 Proof of AVGQ

We will prove the security and correctness of AVGQ. We start with the correctness then continue with the security.

Correctness of AVGQ

We first show that intermediate result parts are actually multiplication of $y_i.d_{i,j}$ and t_i. Remember, $y_i.t_i$ determines that whether user i is the users of both client and server. Further, $d_{i,j}$ is the distance of user i to the F_j. Thus objective is obtaining $y_i.t_i.d_{i,j}$ and also $y_i.t_i$. It can be seen that $q_{1,i} + z_{1,i} = c_{1,i}^{(1)} + \lambda_{1,i}.b_{1,i}^{(1)} + \gamma_{1,i}.a_{1,i}^{(1)} + c_{1,i}^{(2)} + \lambda_{1,i}.b_{1,i}^{(2)} + \gamma_{l,i}.a_{1,i}^{(2)} + \lambda_{1,i}.\gamma_{1,i}$. We previously showed how to simplify a statement similar to this when we proved the RNNQ. We get $q_{1,i} + z_{1,i} = c_{1,i} + \lambda_{1,i}.b_{1,i} + \gamma_{1,i}.a_{1,i} + \lambda_{1,i}.\gamma_{1,i}$. If we decompose λ, γ and $c = a.b$ we get $q_{1,i} + z_{1,i} = a_{1,i}.b_{1,i} + (t_i.b_{1,i} - a_{1,i}.b_{1,i}) + (y_i.d_{i,j}.a_{1,i} - a_{1,i}.b_{1,i}) + (t_i.y_i.d_{i,j} - t_i.b_{1,i} - y_i.d_{i,j}.a_{1,i} + a_{1,i}.b_{1,i})$. If we cancel out the other variable at the end we get $q_{1,i} + z_{1,i} = t_i.y_i.d_{i,j}$ which is indeed the multiplication of two entries of existence array multiplied

with i's distance. In addition, we get the same equation for the other parts which is $q_{2,i} + z_{2,i} = t_i.y_i$.

In step 6, C adds all of its parts, $q_{1,i}$ and $q_{2,i}$. S does the same thing with its shares, $z_{1,i}$ and $z_{2,i}$. At step 8, C adds this shares to combine them and obtain result sum and n_I. For sum, $q_1 + z_1 = q_{1,1} + z_{1,1} + q_{1,2} + z_{1,2} + ... + q_{1,n_u} + z_{1,n_u} = \sum_{i \in U_I \& j \in F} d_{i,j} = sum$. For n_I, $q_2 + z_2 = q_{2,1} + z_{2,1} + q_{2,2} + z_{2,2} + ... + q_{2,n_u} + z_{2,n_u} = n_I$ and we know that $q_{2,i} + z_{2,i} = t_i.y_i$. Therefore, C successfully obtained sum and n_I.

Security of RNNQ

We show the security of AVGQ by Λ and Γ. First we investigate λ's and γ's since they are directly shared to each other. S cannot recover t_i and $a_{1,i}$ or $a_{2,i}$ from $\lambda_{1,i}$ or $\lambda_{2,i}$ since S does not any of the variables. C cannot recover any data form $\gamma_{1,i}$ or $\gamma_{2,i}$ too for the same reason. C shares $a_{1,1,i}$ with S. S now has the $a_{1,i}$'s, b_i's and $c_{2,i}$'s. From these parts, S cannot recover $c = a.b$. Same thing applies to C too. Client has the other parts of the shares so it cannot recover $c = a.b$. Since $q_{1,i}/q_{2,i}$ and $z_{1,i}/z_{2,i}$ calculated with these data parts, $q_{1,i}/q_{2,i}$ and $z_{1,i}/z_{2,i}$ indistinguishable. Thus when client add this parts, it obtains the results of AVGQ.

$$q_{ji} = c_{1i} + \lambda_i \,.\, b_{1i} + \gamma_{ji} \,.\, a_{1i} \qquad z_{ji} = c_{2i} + \lambda_i \,.\, b_{2i} + \gamma_{ji} \,.\, a_{2i} + \lambda_i \,.\, \gamma_{ji} \qquad t_i \,.\, y_{ji} = q_{ji} + z_{ji} = c_i + \lambda_i \,.\, b_i + \gamma_{ji} \,.\, a_i + \lambda_i \,.\, \gamma_{ji}$$

$User_i$	t_i	a_i	λ_i
1	1	7	-6
2	0	4	-4
3	1	10	-9
4	1	5	-4
5	0	10	-10
6	0	3	-3
7	1	2	-1
8	0	23	-23
9	1	12	-11

$User_{ji}$	y_{ji}	b_i	γ_{ki}
11	0	2	-2
12	0	16	-16
13	1	5	-4
14	0	13	-13
15	1	8	-7
16	1	24	-23
17	0	57	-57
18	0	3	-3
19	1	8	-7

id	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
c_i	14	64	50	65	80	72	114	69	84
c_{1i}	3.2	34.2	21.7	12.5	76.3	23.8	98.1	30.1	76.5
c_{2i}	10.8	29.8	28.3	52.5	3.7	48.2	15.9	38.9	7.5

$$q_{11} = 3{,}2 + (-6) * 1{,}1 + (-2) * 5{,}1 = -13{,}6$$

$$z_{11} = 10{,}8 + (-6) * 0{,}9 + (-2) * 1{,}9 + (-6) * (-2) = 13{,}6$$

$$q_{11} + z_{11} = t_1 \,.\, y_{11} = 0$$

$$q_{13} = 21.7 + (-9) * 2{,}1 + (-4) * 3{,}3 = -10{,}4$$

$$z_{13} = 28.3 + (-9) * 2{,}9 + (-4) * 6{,}7 + (-9) \,.\, (-4) = 11.4$$

$$q_{13} + z_{13} = t_3 \,.\, y_{13} = 1$$

Figure 6.2: An example run of MPC based RNNQ. Protocol equations are at top. Left and middle table contains existence arrays T/Y, triplet parts a_i/b_i's and Λ/Γ of C/S. Right-most table contains c values along with it shares. Below of the right-most table there are two example calculations executed at step 6.

Chapter 7

Experiments and Results

In this chapter, we analyze the performance and the utility of our protocols. We compare our protocols between each other and discuss each ones advantages and disadvantages. First, we give experimental results and performances. Second, we compare the protocols and discuss them.

7.1 Performance

We show the number of computations by each query with each protocol type in Table 7.1. This table shows that the relation of each query's run time to the query parameters. For example, Paillier based RNNQ depends on n_s whereas MPC based RNNQ depends on n_u. Note that different cryptosystems have different operations which have different costs. For example, Paillier encryption is different than Lattice encryption or Paillier multiplication is different than Lattice homomorphic *add* (Hm. *add*). This table only shows the query dependencies to the execution parameters.

To investigate the cost of privacy in our protocols, we implemented the non-privacy preserving protocols for our both queries. In this way, we can see that the additional computation cost that our protocols brings. In non-privacy preserving

solution, client sends its user list to server. Server calculates the distance of users to facilities and calculate the query result according to the query type. Lastly, server returns the query result to the client. We implemented the non-privacy preserving solution using Java. Our client and server was in the same machine but running in the different processes. Our machine was a Mac OSX with Intel i5 processor. We include the results in Table 7.2.

Table 7.1: Computations Numbers of Each Query

	RNNQ	AVGQ	Setup
Paillier	$k.n_s$ dist. comp. n_s mult k enc. k mult. k dec.	$k.n_s$ dist. comp. n_s exp. $2.n_s$ mult 2 enc. 2 mult. 2 dec.	n_u enc. 1 key gen.
Lattice	$k.n_s$ dist. comp. n_s Hm. *add* k enc. k Hm. *add* k dec.	$k.n_s$ dist. comp. n_s Hm. *plain_mult* $2.n_s$ Hm. *add* 2 enc. 2 Hm. *add* 2 dec.	n_u enc. 1 key gen.
MPC	$2.k.n_u$ add $2.k.n_u$ split $k.n_s$ dist. comp. $7.k.n_u$ add $7.k.n_u$ mult. k add	$4.n_u$ add $4.n_u$ split $k.n_s$ dist. comp. $14.n_u$ add $14.n_u$ mult. 2 add	$r.n_u$ key gen. $r.n_u$ split

We evaluate each protocol with the following criteria. First, runtime of the queries is investigated. Investigating query runtimes are one of the main goal of this work and it is important since real life applications mostly deal with this property. We additionally measure the setup phase times.. Second, we look for memory usage of the query that is how much memory used during the execution of the queries. Memory usage values are total memory of usage of both server and client. Third, we look for communication efficiency (Bandwidth in the Table 7.3). We measure how much data is transmitted during the protocols. Most of this transmitted data is ciphertexts in Paillier and lattice based protocols, so bandwidth is mostly effected by ciphertext size. In MPC based protocols, parties send the masked existence matrix to other parties, thus superset size n_u and facility number k effects the bandwidth in MPC protocols. Note that

we exclude one-time setup cases in memory efficiency field. Next, we look each protocols ability to dynamically update their existence arrays. Protocols and their operations are based on the existence arrays and encrypted existence arrays as we explained in previous sections. In real life, some businesses' customers are very dynamic meaning that new people constantly becoming customers of the businesses' at the same time some customers stops being customers of businesses. Hence, existence arrays may need some updates according to these customer changes. We look protocols ability to change their existence arrays without use of an intensive calculations. Lastly, we investigate the ability to handle multi client, one server scenario. In other words, we look for whether protocols can run with one server with multiple client scenarios at the same time without setbacks for any client. We summarize our findings in Table 7.2 and Table 7.3.

We already tested the performance of the Paillier based protocols in our demo paper [22]. We implemented Paillier based protocols in Java and we used the Paillier cryptosystem implementation from [24]. We used two different machines for client and server to make the setting more realistic. Our client was in Ankara, Turkey and server was in London, UK. Client was Mac OS X with 1.6 GHz Intel i5 processor. Server was Ubuntu with 1.8 GHz Intel Xeon processor. Our parameters for scheme are as follows. Our modulus n was 1024 bit which means each ciphertext is 2048 bit long. We artificially generated data since we are testing the protocol's feasibility and correctness. Superset size is one million. Server has $250,000$ users and client has $50,000$ users. In our queries there are 5 facilities.

For lattice based protocols, we used Simple Encrypted Arithmetic Library (SEAL) version 2.2.0 [25] [26]. We implemented lattice based protocols using C++. Our machine was Mac OS X with Intel i5 processor. We picked 1024 as our polynomial modulus. We again generated the data artificially. For lattice based protocols our user numbers are 10 times smaller compared to Paillier based and multiparty computation based protocols. In lattice based protocols, superset size is $100,000$, server has $25,000$ and client has $5,000$ users. We executed queries with 5 facilities. Reason for that is, lattice based protocols allocates huge amounts of memory that our machines cannot not handle. However, in table 7.2, we normalized the runtimes and memory values to scale the values as other protocols. Memory

cost is coming form the size of the encrypted existence array which we have 1 million encrypted values. It increases and decreases with respect to superset size. Runtime of the queries depends on the user number of server. It increases linearly with respect to the user number of server. Hence, we tested the protocols using less number of users but in the end we normalized the values to compare lattice based protocols with other protocols. SEAL uses FV cryptosystem [14] which is a SWHE scheme meaning that there is limited number of operations on ciphertexts which we can conduct. However, our queries mainly uses addition of two ciphertext with AVGQ uses *plain_mult* operations additionally. Both of the operations, especially *add*, generates very little noise. During our experiments, both RNNQ and AVGQ used only 2% of their noise budget.

Multiparty computation based protocols is written in Java and tested on Mac OS X with Intel i5 processor. We generated 10 million triplets, $c = a.b$, and superset size is 1 million. Server has $250,000$ and client has $50,000$ users. There are again 5 facilities. We artificially generated the data here as well.

Table 7.2: Result of different methods for the queries. All times are in seconds except for setup

	RNNQ Runtime	AVGQ Runtime	Setup
Non-Privacy	0.12	0.13	-
Paillier	9.12	94.50	11 h
Lattice	6.03	53.05	2.5 h
MPC	0.47	0.36	0.13 s

Table 7.2, basically shows the total runtime of the all the proposed queries including the non-privacy preserving queries. Runtime measurement starts with query request and ends with client obtaining the result. Measuring the total time of the queries is one of the simplest way to compare the existing protocols. It also shows the setup times for the queries. In Paillier and lattice based protocols, setup phase involves encrypting the n_u values which may be large. Usually encrypting is a costly operation. Hence setup phase takes a lot time. Setup phase of MPC based protocols, involves only generating large amount of $c = a.b$ triplets which are computationally cheap compared to encryptions.

Table 7.3: Summary of the proposed protocols

	Memory	Bandwidth	Dynamically Update	Multi-Client
Paillier	594 MB	1.25 KB	Partially	Yes
Lattice	63 GB	326 KB	Partially	Yes
MPC	558 MB	332 MB	Yes	Yes

7.2 Discussion

In this section, we discuss the experimental results. Moreover, we give a scenario that exploits the advantages of each protocol.

MPC based protocols are clearly faster than the other protocols. MPC based queries are almost fast as non-privacy preserving protocols. Hence we can say that it is much more suitable to the real life applications where responsiveness is more important to the users. Lattice based protocols are faster compared to the Paillier based ones. Remember, only difference between lattice based and Paillier protocols are the cryptosystem. Lattice based protocols use huge amounts of memory. Most of the modern systems cannot fit the query into their memory during the query execution unless superset size is relatively small. MPC based protocols uses very small memory since there is no encryption of any kind. Paillier based protocols has the greatest setup time since one encryption took roughly 40 ms in our experiments. Lattice based protocols encrypt the data faster compared to Paillier based ones. One encryption take roughly took 9 ms in our experiments. MPC based protocols have the fastest setup phase by far, making it preferable in many cases. In terms of communication efficiency, bandwidth, Paillier based protocols are considerably efficient compared to other protocols. This is because each ciphertext is 2048 bits in Paillier and 64 KB in lattice based encryption. For MPC, $2(k+1)n_u$ numbers transmitted over the network since n_u is a large number there is a lot of network traffic. Paillier and lattice based protocols holds the encrypted existence arrays. Changing one user might expose that user to the server since if client sends the new encrypted value for that user server can understand status of the user changed. One solution for that is freshly encrypting the whole existence array or some parts of the array. However, this may be costly

because encrypting is a expensive operation in terms of time. Hence, Paillier and lattice based cannot update their existence arrays with small amount of work so they can partial dynamically update their users. Table 7.2 shows that it requires large amount time to encrypt the whole existence arrays. At the other hand, MPC based protocols can easily update their users since they mask their existence arrays as Γ and Λ in every time there is a query. All protocols can support multi-clients. In Paillier and lattice based protocols, client sends the encrypted existence arrays with clients PK. Server can hold each clients data and use it when necessary. In MPC based protocols, existence arrays send when there is query. Server will calculate its existence arrays according to the query locations so there will not be any problem.

7.2.1 Paillier Based Protocols Use Case

Paillier based protocols has the slowest runtime but also the best in terms of communication efficiency. Therefore, Paillier based protocols are better when they exploit their advantage of communication efficiency. One scenario is when the network between client and server is either poor or unreliable or latency is high, Paillier based protocols can be used since Paillier based protocols transmit very small data compared to others. Paillier based protocols uses smaller memory compared to lattice based protocols. If memory is a constraint in a certain scenario with network limitations, Paillier based protocols may be preferred.

7.2.2 Lattice Based Protocols Use Case

Lattice based protocols has mediocre runtime but memory consumption is high. It has a good communication efficiency as the Paillier based protocols. In addition, it is post-quantum resistant and allows multiplication of ciphertexts which is essential for our future complex queries. Client may request queries consecutively sometimes. In such cases, Lattice based protocols outperforms other queries. Although MPC based protocols are faster, in each query, MPC based protocols

need to transfer huge chunks of data and transmitting such a large data could take more time than the query itself. In scenarios that users of the client are static, lattice based protocols are preferable. For example, consider a case that client sends 5 RNNQ request. Paillier based query would take 45 seconds meanwhile lattice based would take only 30 seconds roughly.

7.2.3 MPC Based Protocols Use Case

MPC based protocols are superior in many ways compared to other protocols except for communication efficiency. It is the fastest both in query times and setup times, uses the least memory and can easily update its user list. MPC based protocols can be used in many cases. For example, in cases when high rotation of customers exists for LBSP where many new customers arrive and existing ones leave, Paillier and Lattice based protocols became inefficient due to their long setup phase. On the other hand, MPC based protocols complete the setup phase and query less than one second. In addition, MPC based protocols can be used in machines with low memory power such as mobile phones since it requires small memory.

7.3 Improvements on Implementation of Paillier Based Protocols

To make our Paillier based protocols more realistic to the real world application and further improve its efficiency, we reorganized and re-write the implementation of [1]. Firstly, we placed the client and the server to the different machines and different locations. In the real life, LBSPs stores their data in big data centers which are usually located in big cities [27]. For this reason, we placed our server to the London, UK and our client to the Ankara, Turkey. To show that our protocols can be practical, we improved the efficiency of our test code. We further tested our protocols with different scenarios which differ by user numbers.

Our implementation also includes a basic graphical user interface for client and server. Screenshots of our GUI can be seen in figure 7.1.

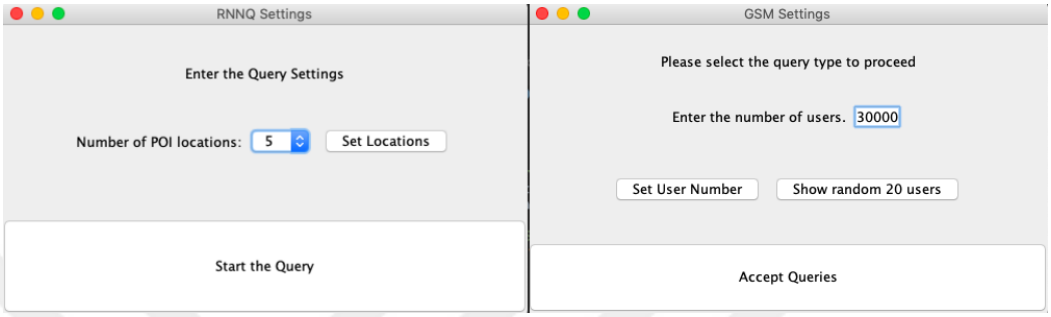

Figure 7.1: Screenshots of our demo paper implementations are above. Client GUI is at left just before sending the query request. Client can change the number of locations and set their coordinates. Server GUI is at right in initialization phase. Server can set their user number. User coordinates are generated randomly. Server can see users coordinates.

Chapter 8

Conclusion and Future Work

We proposed two new schemes for privacy-preserving optimal location queries and compared them with our existing Paillier based scheme. We proposed two privacy-preserving queries which one aims to distribute the users to the facilities evenly and the other aims to minimize the average distance of users to their closest facility. In our protocols, we hide user lists of client and server, u_c and u_s, from each other. We further hide locations of users of servers from client and we hide query result from server. We implemented, evaluated and compared the performance of the three proposed schemes. We showed that these queries can be executed in three different privacy-preserving way. All three types of protocols have different advantages and suitable for different situations. For future works, more complex queries can be designed since RNNQ and AVGQ are simple queries to show that our schemes are practical and feasible. Queries such as filtering the users according to some feature, i.e age or gender, can be added. In addition, our methods can be used in digital advertisement systems. Scenario of these kind of systems are similar to ours which client provide user data and advertisers hold the advertise history of users. Problem here is the intersection of these two user sets in privacy preserving manner. We plan to extend and apply our protocols for these kind of scenarios which requires logical query processing.

Bibliography

[1] E. Yilmaz, H. Ferhatosmanoglu, E. Ayday, and R. C. Aksoy, "Privacy-preserving aggregate queries for optimal location selection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[2] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Advances in Cryptology — EUROCRYPT '99*, pp. 223–238, 1999.

[3] Y. Du, D. Zhang, and T. Xia, "The optimal-location query," in *International Symposium on Spatial and Temporal Databases*, pp. 163–180, Springer, 2005.

[4] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *ACM Sigmod Record*, vol. 29, pp. 201–212, ACM, 2000.

[5] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: Query processing for location services without compromising privacy," in *Proceedings of the 32nd international conference on Very large data bases*, pp. 763–774, VLDB Endowment, 2006.

[6] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pp. 620–629, IEEE, 2005.

[7] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[8] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 571–588, 2002.

[9] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *2008 The 28th International Conference on Distributed Computing Systems*, pp. 311–319, IEEE, 2008.

[10] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 121–132, ACM, 2008.

[11] J. Wang, Z. Cai, Y. Li, D. Yang, J. Li, and H. Gao, "Protecting query privacy with differentially private k-anonymity in location-based services," *Personal and Ubiquitous Computing*, vol. 22, no. 3, pp. 453–469, 2018.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[13] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*, vol. 20. Stanford University Stanford, 2009.

[14] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption.," *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[15] I. Damgård and M. Jurik, "A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system," in *International Workshop on Public Key Cryptography*, pp. 119–136, Springer, 2001.

[16] E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont, "Protecting and evaluating genomic privacy in medical tests and personalized medicine," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pp. 95–106, ACM, 2013.

[17] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping." Cryptology ePrint Archive, Report 2011/277, 2011.

[18] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based." Cryptology ePrint Archive, Report 2013/340, 2013.

[19] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[20] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164, IEEE, 1982.

[21] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*, pp. 420–432, Springer, 1991.

[22] C. Eryonucu, E. Ayday, and E. Zeydan, "A demonstration of privacy-preserving aggregate queries for optimal location selection," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 1–3, IEEE, 2018.

[23] N. P. Smart, "Multi party computation: From theory to practice." "Workshop on Real-World Cryptography" presentation at Stanford University, 2013, 2013.

[24] K. Liu, "Paillier's cryptosystem in java." `https://www.csee.umbc.edu/~kunliu1/research/Paillier.html`, 2017. [Online; accessed 09-October-2017].

[25] C. R. G. at Microsoft Research, "Simple encrypted arithmetic library (seal)." `https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/`. [Online; accessed 23-January-2018].

[26] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library - SEAL v2.1," in *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, pp. 3–18, 2017.

[27] A. W. Services, "Global infrastructure of aws." `https://aws.amazon.com/about-aws/global-infrastructure/`, 2019. [Online; accessed 1-July-2019].