Privacy-Preserving Aggregate Queries for Optimal Location Selection

Emre Yilmaz, Hakan Ferhatosmanoglu, Erman Ayday, and Remzi Can Aksoy

Abstract—Today, vast amounts of location data are collected by various service providers. These location data owners have a good idea of where their users are most of the time. Other businesses also want to use this information for location analytics, such as finding the optimal location for a new branch. However, location data owners cannot share their data with other businesses, mainly due to privacy and legal concerns. In this paper, we propose privacy-preserving solutions in which location-based queries can be answered by data owners without sharing their data with other businesses and without accessing sensitive information such as the customer list of the businesses that send the query. We utilize a partially homomorphic cryptosystem as the building block of the proposed protocols. We prove the security of the protocols in semi-honest threat model. We also explain how to achieve differential privacy in the proposed protocols and discuss its impact on utility. We evaluate the performance of the protocols with real and synthetic datasets and show that the proposed solutions are highly practical. The proposed solutions will facilitate an effective sharing of sensitive data between entities and joint analytics in a wide range of applications without violating their customers' privacy.

Index Terms—Privacy, data encryption, security, integrity, and protection, query processing, algorithm/protocol design and analysis

1 Introduction

Understanding the whereabouts of current and potential customers can provide valuable insights for location-based services, facility location, and competitive business decisions. Increasing amounts of location data from mobile services, applications, and network operators have introduced exciting opportunities for location-enhanced business analytics. The approaches presented in the marketing and operations research literature commonly assume that a business that wants to do analysis owns the data about it. However, this is rarely the case. Location data is typically collected by mobile telecommunication operators and service providers, such as Foursquare. These data owners seek ways to enable other businesses to run location-based analytics queries without violating their customers' privacy. Thus, one needs to prevent the location-based service providers from tracking the users individually, while still allowing other businesses to obtain useful information. Similarly, businesses do not want to share their customer lists with location-based service providers. In this work, we develop efficient privacy-preserving query processing protocols that help to identify the best locations to open new branches considering the distribution of the customer locations.

Optimal location selection is a common location-based analysis that seeks the best location to open a new facility optimizing an objective function given a set of existing facilities and a set of customers. A common approach is to utilize computational geometry techniques on the customer locations with the assumption that the locations are known.

However, third party businesses and analysts cannot use these techniques in real life because customer locations are not always known by these businesses. To perform successful location-based queries, businesses need up-to-date locations that can be gathered from location data owners, such as mobile operators and location-based service providers. For instance, while retail stores or banks may know the home addresses of their customers, they may also like to know their locations during certain time periods in the day. Work addresses of the customers may be missing or out-of-date in their databases. The location information needs to be gathered from data owners while preserving sensitive information of businesses and data owners, as well as the privacy of their customers including their identity and location.

To be consistent, in this paper we refer the location data owner as the server, and the business that requests queries as the client. We refer their customers as the users of the server and the users of the client. The client has existing facilities, such as branches of a bank, and aims to find the optimal location for the new one among several candidates. The client is able to request a fundamental class of queries that can be used in optimal location selection. In these queries, the client only obtains aggregate information about locations of its users without learning the location of any specific user. The client has several candidates for the new facility and it can request the queries for each candidate location and select the best one.

A simple example to these aggregate queries is average distance query, in which the client retrieves the average

- *E. Yilmaz and E. Ayday are with the Computer Engineering Department, Bilkent University, Ankara 06800, Turkey. E-mail: {emre.yilmaz, erman}@cs.bilkent.edu.tr.*
- *H. Ferhatosmanoglu is with the Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK, and the Computer Engineering Department, Bilkent University, Ankara 06800, Turkey. E-mail: hakan.f@warwick.ac.uk.*
- *R.C. Aksoy was with Bilkent University, Ankara 06800, Turkey. He is now with the University of Michigan, Ann Arbor, MI 48109. E-mail: remzican@umich.edu.*

Manuscript received 7 Sept. 2016; revised 12 Jan. 2017; accepted 3 Mar. 2017. Date of publication 12 Apr. 2017; date of current version 13 Mar. 2019. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2017.2693986

distance of its users to their nearest facilities. The nearest facility of each user is the facility that has the minimum distance to that user. The average distance is a valuable information for the client to minimize it for maximizing user benefit. In a non-privacy-preserving solution for this query, the client sends the facility locations and its user list to the server. The server checks the location of each user (who gave informed and explicit consent for this information) and calculates distances to their nearest facilities. At the end of the query, the server returns the average distance and the client obtains useful information for facility location without tracking its users individually. The client can send a different location for the new facility in each query together with the locations of existing facilities. As a result, it can select the best candidate that minimizes the average distance between users and their nearest facilities.

For a privacy-preserving solution, we need to hide the client's user list and the server's user list from each other. We also need to hide the answer to the query from the server. Otherwise, the server learns the best candidate for the new facility and it may share this information with the competitors. We investigate privacy-preserving solutions to aggregate queries which allow analyzing location data in servers and selecting the best facility location. With the proposed solutions, without sharing its user list with the server, the client can obtain aggregate information about user locations and find an optimal place for its new facility among several candidates depending on different objective functions. These objectives are (i) uniformly distributing the cardinality of the reverse nearest neighbors (RNN), i.e., the set of points that has the query point as the closest facility, (ii) minimizing the average distance between each user and her closest facility, and (iii) minimizing the maximum distance between a user and her closest facility.

We define three fundamental aggregate queries for optimal location selection and propose two types of privacy-preserving query-processing protocols for each type of query, utilizing partially homomorphic encryption as a building block. We encrypt the sensitive data of the server and the client, and perform the operations on the encrypted data to preserve the privacy of both parties. First, we explain server-based protocols, in which most computation is performed by the server, and hence the workload of the client is low. This solution is particularly convenient when the client has limited computational power. To decrease the communication overhead in each query, we also propose client-based protocols. In these protocols, the client performs the majority of the computation during the setup phase (which occurs only once). After completion of the setup phase, all queries are processed with low communication overhead. Therefore, our client-based solution is highly efficient when the client undertakes some pre-computations before running its queries.

During the protocols, homomorphic encryption is used for keeping the user list of the client and the query result hidden from the server and keeping the user list of the server and location data hidden from the client. Initially, we describe the protocols to return exact query results. Since the server is unaware of the query result and the queries return aggregate results, some queries may leak information about users. For instance, if the result of a counting query is one, that user can be predicted by the client. To prevent information leak about any single user, we also satisfy differential privacy in our protocols by adding controlled noise to the query result. Therefore, we use homomorphic encryption and differential privacy together to guarantee privacy of individuals during query processing. Our contributions are summarized as follows:

(1) We introduce a practical setting in which the client (e.g., a business) runs a useful class of location-based queries on the database of the server (e.g., a location-based service provider) without violating the privacy of individuals involved both in the client and the server side.
(2) We enhance facility location problems by removing the assumption that the customer locations are known to the businesses. With the proposed solutions, a business can find the best location for a new facility among several candidates without knowing its customer locations.
(3) We introduce two novel query processing protocols for different types of queries, i.e., RNN cardinality query, average distance query, and maximum distance query that can be used as a service to identify optimal facility location. Our protocols utilize homomorphic encryption for protecting privacy of both parties and satisfy differential privacy. We also discuss the impact of differential privacy on the utility of the protocols.
(4) The proposed protocols take advantage of using a potential superset of user space to hide the user lists of both parties. Our solution does not use any computationally expensive cryptographic comparisons such as private equality testing or private set intersection. The performance evaluations show that the proposed protocols are practical, efficient, and scalable. For instance, when the server has 25 million users, executing privacy-preserving RNN cardinality query takes around 10 seconds on a modest computer.

The remainder of this paper is organized as follows: A literature review and background information are given in Section 2. Section 3 presents the system model, the threat model, and the definitions of the aggregate queries for optimal location selection. We describe the server-based solutions in Section 4 and the client-based solutions in Section 5. In Section 6, we explain how to achieve differential privacy in our protocols. We present our experimental results in Section 7. Finally, we conclude in Section 8.

2 Related Work and Background

Since our work is related to optimal location queries and privacy-preserving location-based query processing, we give the literature review of both subjects and explain the major differences between our work and previous works in the literature. The concept of differential privacy and homomorphic encryption schemes are also explained in this section as building blocks of our protocols.

2.1 Optimal Location Queries

Given a set of existing facilities and a set of users, the optimal location query [8] finds a location l for the new facility

with maximum influence. The influence of a point is commonly formalized based on its RNNs [17]. The RNN query finds the set of points that has the query point as the nearest neighbor (NN). There are two variants of RNN queries. In the monochromatic version, all points belong to the same category. In the bichromatic version, points are divided into two categories, such as users and facilities. Given a facility f, the bichromatic RNN query finds the set of users that has f as the nearest facility. The general assumption in optimal location queries is that each user prefers her closest facility. Therefore, the RNN query plays an important role in facility location problems because a facility's RNN is the set of users who prefers this facility.

Businesses run optimal location queries to find the best locations for their new facilities. The definition of "best location" or "location with maximum influence" depends on the type of the facility. In [8], the influence of a location is defined as the total weight of its RNNs. The authors define the problem with weighted users and aim to maximize the total weight of users that are closer to the new location than to their closest facilities. L_1 distance is considered in [8] and they propose three methods to solve the problem. Another solution to maximize the bichromatic RNN for L_2 distance is proposed in [25].

In the literature, there are also other definitions of the "best location" which aim to maximize user benefit and increase service quality. One of them is minimizing the maximum distance between a user and her closest facility [2], [3]. Another objective is minimizing the average distance between each user and her closest facility. The problem is proposed as min-dist optimal-location query in [29]. This query has many real-life applications where it aims to improve the quality of service or reduce the logistics cost by businesses. [29] and [23] solve the problem with L_1 and L_2 distance assumptions, respectively.

In previous works on facility location problems, it is assumed that customer locations are known. In this paper, we assume that customer locations are not known by businesses, but stored in a location-based service provider, and businesses need to analyze location data by requesting queries. We define three aggregate queries for optimal location selection and develop privacy-preserving protocols for them. These queries are defined to analyze the location data and they can be used in optimal location selection. Businesses can decide the best location among the candidates by requesting several queries and comparing the query results.

2.2 Privacy-Preserving Location-Based Query Processing

Today, vast amounts of information are collected and analyzed in databases around the world. Data may be stored by multiple parties and these parties may not be keen on sharing their data with others. In secure multi-party computation (SMC), multiple parties jointly compute a function over their inputs without revealing their inputs to each other. In [7], several SMC problems are identified. One such problem defined in [7] is the privacy-preserving database query, where Alice seeks a match with her private string q in Bob's database T. The privacy requirement is hiding q and the query result from Bob, and hiding T from Alice. The authors develop an efficient solution for the matching problem in [6] by using a semi-trusted third party.

Privacy-preserving location-based queries have been studied in the literature. Cheng et al. [4] propose a privacy-preserving range query protocol to find users within a range with non-zero probability. In [4], each user has a cloaked region to hide her exact location, and the probability of being within a range depends on the intersection of the cloaked regions. A hybrid approach that integrates private set intersection and location cloaking is presented in [26]. For privacy-preserving NN queries, a privacy-aware query processing framework called Casper is presented in [19]. This framework uses a location anonymizer to blur users' exact locations into cloaked regions. Ghinita et al. [14] eliminate the usage of third-party anonymizers by using cryptographic techniques. They utilize private information retrieval techniques to preserve location privacy. In [24], efficient protocols are proposed for privacy-preserving k-NN searches by using several primitive SMC protocols. Yi et al. [28] present solutions for the same problem and use Paillier encryption and location cloaking as building blocks.

For privacy-preserving location-based query processing, one can follow several approaches, such as location perturbation [19], providing k-anonymity by dummy locations [20], data transformation [16], and using cryptography [14], [24], [28]. We follow the cryptographic approach, which provides privacy without compromising utility. However, providing exact query results may cause information leaks in some cases such as counting queries. Therefore, we integrate the principle of differential privacy [9] into the proposed protocols. We explain the notion of differential privacy in Section 2.3.

Existing works on location privacy try to hide the location information that the client (i.e., querying side) has from the server (i.e., location-based service provider). In our scenario, user location information is stored in the server and the server hides this sensitive information from the client. The client wants to analyze its customers' locations in order to find the optimal facility location. One approach to allow analytics on the location data can be publishing anonymized data by the server. However, the client cannot identify its users in anonymized data and anoymized location data can also be vulnerable to de-anonymization attacks [5]. Therefore, the client should retrieve its users' aggregate information via privacy-preserving queries. Since both parties must hide their user lists from each other, the server and the client must find their common users collaboratively without learning these common users. In this work, we propose novel secure two-party protocols that allow analyzing location data in the server. We develop our protocols using potential superset of user space to hide the user lists of both parties.

2.3 Differential Privacy

Differential privacy aims to protect the privacy of individuals while releasing aggregate information about the database. It is based on the neighborhood of databases. Two databases $\mathtt{D}$ and $\mathtt{D}'$ are neighbors if they differ in only one entry. Differential privacy requires that query results for two neighbor databases should be indistinguishable. Let the output of a protocol P on database $\mathtt{D}$ be $P(\mathtt{D})$. The differential privacy is formally defined as follows:

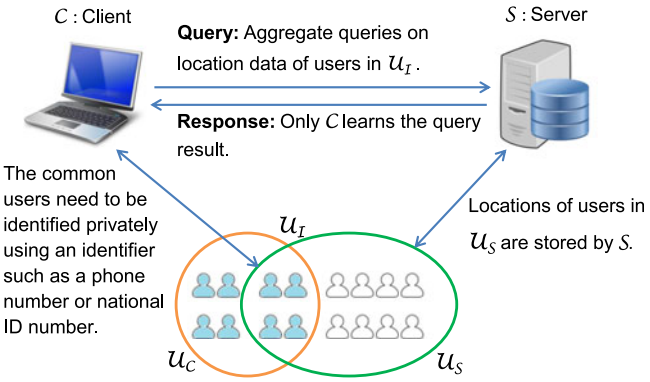

Fig. 1. System model.

Definition 1. *Protocol P satisfies ϵ-differential privacy if for any two neighbor databases $\mathtt{D}$ and $\mathtt{D}'$, and any subset S of output space of P,*

$$\Pr[P(\mathtt{D}) \in S] \leq \Pr[P(\mathtt{D}') \in S] \cdot e^{\epsilon}.$$

A typical way to achieve differential privacy is adding controlled random noise to the query result. For numeric queries, Laplace mechanism can be used to produce the noise drawn from the Laplace distribution. Let $Laplace(\lambda)$ be a sample from Laplace distribution with mean 0 and standard deviation λ. To obtain ϵ-differential privacy, the noise drawn from the Laplace distribution must be calibrated according to the sensitivity of the protocol [10]. The sensitivity of the protocol is the maximum possible change on the output by changing a single record in database. Given a protocol P, the sensitivity of the protocol is defined as follows:

Definition 2. *Let $\mathcal{N}$ be the set of all pairs of neighbor databases*

$$\Delta P = \max_{(\mathtt{D},\mathtt{D}') \in \mathcal{N}} \|P(\mathtt{D}) - P(\mathtt{D}')\|.$$

Therefore, a protocol P satisfies ϵ-differential privacy for the result

$$P(\mathtt{D}) + Laplace\left(\frac{\Delta P}{\epsilon}\right).$$

In Section 6, we show the sensitivity of each considered query and how to achieve differential privacy during the protocols.

2.4 Homomorphic Encryption

In homomorphic encryption, a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext. Cryptosystems that allow homomorphic computation for a limited number of operations such as addition or multiplication are called partially homomorphic. For instance, given two messages x and y, one can compute the encryption of $x + y$ by using the encryptions of x and y in an additive homomorphic encryption scheme. In multiplicative homomorphic schemes, $E(x \cdot y)$[1] can be computed by using $E(x)$ and $E(y)$. Gentry [13] proposed first fully homomorphic encryption scheme that supports both addition and multiplication. Since partially homomorphic schemes are more efficient and calculating the sum is sufficient for our protocols, we are interested in additive homomorphic cryptosystems [1], [21], [22], satisfying $E(x) \cdot E(y) = E(x + y)$. Another homomorphic property of these cryptosystems is that encrypted plaintext $E(x)$ raised to a constant k is equal to encryption of the product of the plaintext x and the constant k, i.e., $E(x)^k = E(x \cdot k)$.

We develop our protocols by using the Paillier cryptosystem [22]. In Paillier, if the public key (PK) is the modulus m and the base g, then the encryption of a message x is $E(x) = g^x \cdot r^m \pmod{m^2}$, for some random $r \in \{0, \ldots, m-1\}$. Using a random value r in encryption ensures that two messages that are the same will encrypt to the same value with only a negligible likelihood. Hence, Paillier provides *semantic security*. m should be selected as the product of two primes p and q. The private keys (SK) of the Paillier cryptosystem are $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^{\lambda} \bmod m^2))^{-1} \bmod m$, where $lcm(a, b)$ is the least common multiple of a and b, and $L(u) = \frac{u-1}{m}$. The decryption of a ciphertext c can be performed using private keys as follows: $D(c) = (L(c^{\lambda} \bmod m^2) \cdot \mu) \bmod m$. Paillier satisfies $E(x) \cdot E(y) = E(x + y)$, because $(g^x \cdot r_1^m) \cdot (g^y \cdot r_2^m) = g^{x+y} \cdot (r_1 + r_2)^m$. As a result of this homomorphic property, multiplying a ciphertext $E(x)$ with $E(0)$ creates another ciphertext which is the fresh encryption of x.

3 Problem Formulation

We present our system model in Section 3.1. Formal definitions of the queries are given in Section 3.2. We describe the threat model in Section 3.3.

3.1 System Model

There is a server ($\mathcal{S}$) (e.g., a location-based service provider) that provides analytics as a service and a client ($\mathcal{C}$) that requests queries. The server is the database owner and has n_s users $\mathcal{U}_{\mathcal{S}} = \{S_1, S_2, \ldots, S_{n_s}\}$. In addition, the server has location information for each S_i at different time periods. The client has n_c users $\mathcal{U}_{\mathcal{C}} = \{C_1, C_2, \ldots, C_{n_c}\}$ and a list of its k existing facilities $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$. The locations of the existing facilities are public and known by the server. The client wants to run aggregate queries such as count, sum, and maximum on the location data of the server, e.g., to analyze the candidate locations for a new branch. The client aims to hide $\mathcal{U}_{\mathcal{C}}$ and the query results from the server. The server also aims to hide $\mathcal{U}_{\mathcal{S}}$ from the client and prevent user tracking by the client. Hence, the client will not learn anything about the location of any specific user; it will only obtain the query result at the end of the protocol.

We sketch out our system model in Fig. 1. To run aggregate queries about its users, the client must identify its users in $\mathcal{U}_{\mathcal{S}}$ using an identifier. Before running queries, the server and the client decide on an identifier such as mobile phone number. Most businesses and service providers know mobile phone numbers of their customers. Another identifier can be national identification number. If the server is a telecommunication company and the client is a bank or a hospital they might use national identification number as the identifier. Let $\mathcal{U}_{\mathcal{I}}$ be $\mathcal{U}_{\mathcal{S}} \cap \mathcal{U}_{\mathcal{C}}$, and n_I be the cardinality of

1. For the rest of the paper, $E(x)$ denotes the encryption of message x.

$\mathcal{U}_\mathcal{I}$. Since the server does not have the location information of users in $\mathcal{U}_\mathcal{C} \backslash \mathcal{U}_\mathcal{S}$, we define our queries for the users in $\mathcal{U}_\mathcal{I}$.

We define three useful types of queries for this context: RNN Cardinality Query (RNNQ), Average Distance Query (AVGQ), and Maximum Distance Query (MAXQ). Since the server knows the user locations, it can calculate the distance between a user and a facility via any distance measure. The main challenges are keeping $\mathcal{U}_\mathcal{C}$ hidden from the server and preventing user tracking by the client.

We propose two types of solutions for each query type, the server-based solutions and the client-based solutions. The server is responsible for most of the computation in the server-based solutions. Hence, they are suitable when the client prefers outsourcing computation. The drawback of server-based solutions over the client-based version is their communication overhead. The client-based solutions reduce communication overhead significantly. In the client-based solutions, most of the computation is performed by the client only in the setup phase. In Sections 4 and 5, we describe the server-based and the client-based protocols which return exact query results. Since exact query results may leak information in some cases such as counting queries, in Section 6 we explain how to add controlled random noise to the query result in each protocol to satisfy differential privacy.

3.2 Query Definitions

3.2.1 RNN Cardinality Query

One of the objectives of optimal location queries is uniformly distributing the workload in facilities. In this case, the new facility should attract users from dense facilities. Attracting a user is equivalent to being the closest facility to the user. This query finds the number of users attracted by each facility. The formal definition of the RNNQ is as follows:

Query 1. Given facility locations, find the total number of users in $\mathcal{U}_\mathcal{I}$ attracted by each facility. In other words, calculate the cardinality of RNN for each facility.

In practice, the client can initially run the RNNQ with existing facilities $\mathcal{F}$ to analyze the distribution of the users. Using the result, the client can determine candidate locations for the new facility F_{k+1}. For candidate locations, the client can run the RNNQ with $\mathcal{F} \cup F_{k+1}$. Hence, the client can observe the total number of users attracted by each candidate location for F_{k+1} and select the location that provides the most balanced distribution.

3.2.2 Average Distance Query

One of the objectives of optimal location queries is minimizing the average distance between each user and her closest facility. For instance, delivery services pay attention to decreasing the average distance between their customers and the nearest shop. The AVGQ is formalized as follows:

Query 2. Given facility locations, find the average distance between users in $\mathcal{U}_\mathcal{I}$ and each one's nearest facility.

In practice, the client can run the AVGQ with $\mathcal{F} \cup F_{k+1}$, where F_{k+1} is a candidate location for the new facility. Hence, the client can select the optimal location for F_{k+1}, which minimizes the average distance.

3.2.3 Maximum Distance Query

Another objective of optimal location queries is minimizing the maximum distance between a user and her closest facility. In this objective, the aim is to optimize the worst-case cost of reaching the nearest facility. The MAXQ is formalized as follows:

Query 3. Given facility locations, find the maximum distance between a user in $\mathcal{U}_\mathcal{I}$ and her nearest facility.

In practice, the client can run MAXQ with $\mathcal{F} \cup F_{k+1}$, for candidate F_{k+1} locations. The client can select the optimal location for F_{k+1}, which minimizes the maximum distance.

3.3 Threat Model

In our model, both the server and the client are considered "semi-honest". Therefore, both parties follow the protocol correctly; however, they may try to learn additional information by analyzing the data. That is, the server may try to determine the client's user list, and similarly, the client may try to determine the individual locations of its users during the protocol (by using the messages they receive throughout the protocol). On the other hand, both the server and the client follow protocol execution honestly by forming correct messages, input, and output parameters for each other. This is a reasonable assumption in the problem setting since both parties are motivated to produce the correct result. The server sells the service and the correct result increases the client's satisfaction. Also, the client finds the best facility location if the query results are correctly calculated.

The proposed solutions are secure two-party protocols in which the server and the client wish to compute the query result securely without sharing their inputs with the opposing party. Both the server and the client have sensitive data that should be hidden from the other party. We formally list the sensitive data as follows:

(1) Input of the client: $\mathcal{U}_\mathcal{C}$.
(2) Input of the server: (a) $\mathcal{U}_\mathcal{S}$ and (b) *location information of users in* $\mathcal{U}_\mathcal{S}$.
(3) Output of the protocol: *Query result*.

We aim to hide all of the above sensitive data (from unauthorized parties) in our protocols. The parties must not learn the input of each other. At the end of the protocols, only the client must get the query result and the server must not learn it. The privacy of the server is assured if sensitive data 2 is hidden from the client, and the privacy of the client is assured if sensitive data 1 & 3 are hidden from the server. We prove the security of our proposed protocols in the semi-honest model using the simulation paradigm defined in [15].

While the locations of existing facilities are typically public, the location of a new facility can be sensitive data for the client. In this case, the client can run the query with some dummy locations to provide K-anonymity [20], which provides indistinguishability among K locations. Since the query result is hidden from the server, all of the K locations are indistinguishable for the server.

One potential threat to the server's sensitive data may be obtaining information via exhaustive client queries. By using non-existing facilities, the client can try to obtain information about location of some users. For instance, the

TABLE 1
Symbols Used in Protocols

m_s, m_c	modulus in Paillier generated by ($\mathcal{S}$, $\mathcal{C}$)
g_s, g_c	base in Paillier generated by ($\mathcal{S}$, $\mathcal{C}$)
PK_s, PK_c	public keys of $\mathcal{S}$ and $\mathcal{C}$
SK_s, SK_c	private keys of $\mathcal{S}$ and $\mathcal{C}$
$E_s(x)$, $E_c(x)$	Encryption of message x using (PK_s, PK_c)
$[x]_s$, $[x]_c$	denotes x is encrypted using (PK_s, PK_c)
$D_s([x]_s)$, $D_c([x]_c)$	Decryption of ciphertext x using (SK_s, SK_c)
$d(a,b)$	Distance between points a and b
$\mathcal{U}_\mathcal{S}$, $\mathcal{U}_\mathcal{C}$	user sets of $\mathcal{S}$ and $\mathcal{C}$
$\mathcal{U}$	superset of $\mathcal{U}_\mathcal{S}$ and $\mathcal{U}_\mathcal{C}$
$\mathcal{U}_\mathcal{I}$	$\mathcal{U}_\mathcal{S} \cap \mathcal{U}_\mathcal{C}$
n, n_s, n_c, n_I	total number of users in ($\mathcal{U}, \mathcal{U}_\mathcal{S}, \mathcal{U}_\mathcal{C}, \mathcal{U}_\mathcal{I}$)
$\mathcal{F}$	set of existing facilities of $\mathcal{C}$
k	total number of existing facilities
q, $\mathcal{Q}$	result (value, set) of the query
w	random number greater than q in MAXQ

client can divide the whole region into two regions and select the center of each region as a facility location. When the client performs RNNQ with these facility locations, it learns the total number of users in each region. The client can divide each region into smaller regions in subsequent queries, until each region has at most one user. At the end, the client learns the small regions which contains a user and it may predict the user in a small region with background knowledge. Therefore, if the total number of facilities in the query is very small or very large, the client may obtain information about user locations.

We assume the locations of k existing facilities of the client are public and known by the server. The server decides two threshold values θ_1 and θ_2 such that the client can add at most θ_1 new facilities or remove at most θ_2 existing facilities in a query.[2] Thus, when the client sends the locations of the facilities, the server aborts the protocols in following cases:

- if the total number of facilities is greater than $k + \theta_1$,
- if the total number of facilities is less than $k - \theta_2$,
- if the facilities in the query do not include at least $k - \theta_2$ existing facilities of the client.

There is a tradeoff between utility and privacy in the selection of these threshold values. Selecting small θ_1 and θ_2 increases privacy, however, the utility of the protocols decreases due to rejection of more queries. Therefore, there cannot be an optimal threshold value for the protocols.

Moreover, when the query result includes a small number of users, the client can make an estimate about these users. For instance, there may be only one user whose nearest facility is a particular facility in RNNQ. Hence, if the RNN cardinality of a facility is one in RNNQ, the client can predict that user using its background knowledge. To prevent such privacy leaks in our protocols, we explain how to provide differential privacy in Section 6.

Finally, we also assume that during the protocol, communication is encrypted between the server and the client against an eavesdropper and that the server and the client (s) do not collude.

2. θ_1 and θ_2 are design parameters of RNNQ, AVGQ, and MAXQ to be decided by the server.

4 Server-Based Query Processing Protocols

In this section, we propose server-based solutions that preserve the privacy while processing the queries in Section 3.2. We introduce the high-level overview of the server-based protocols in this section and we give the detailed steps of the protocols in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2017.2693986. We present the security analysis of server-based protocols in Section 4.1. Table 1 shows the symbols used in the protocols.

The underlying protocols utilize the additive homomorphic property to hide sensitive data from other parties by calculating the sum of the encrypted values without decrypting them. We utilize Paillier cryptosystem as an additive homomorphic scheme satisfying $E(x) \cdot E(y) = E(x + y)$. In the server-based protocols, the server creates a public and private key pair (PK_s, SK_s), and shares the public key with the client. The client can encrypt any value or perform homomorphic operations on the ciphertexts, but only the server can decrypt encrypted messages. The server performs the majority of the encryptions in the protocols.

In the setup phase, the server generates (PK_s, SK_s) for Paillier cryptosystem. In addition, the server selects a superset $\mathcal{U} = \{U_1, \ldots, U_n\}$ of $\mathcal{U}_\mathcal{S}$ such that $\mathcal{U}_\mathcal{S} \subset \mathcal{U}$. The aim of selecting $\mathcal{U}$ is hiding $\mathcal{U}_\mathcal{S}$ (sensitive data 2(a)) from the client. For instance, let the identifier used in the protocols be mobile phone numbers. Location-based service providers such as Foursquare and mobile telecommunication operators, and most businesses such as banks, hotels, and retailers typically know the mobile phone numbers of their customers. Hence, they can use mobile phone numbers as identifiers. Assume the phone numbers consist of seven digits and there are 50 different mobile operator codes. When the superset $\mathcal{U}$ contains all possible mobile phone numbers, n becomes 500 million. Since $\mathcal{U}$ contains all possible numbers, it completely protects $\mathcal{U}_\mathcal{S}$ from the client. Another example is using national identification numbers as identifier. If national id numbers consist of nine digits and the superset $\mathcal{U}$ contains all possible id numbers, n becomes one billion. The server shares $PK_s = (g_s, m_s)$ and $\mathcal{U}$ with the client. Note that all multiplications and exponentiations of ciphertexts in the server-based protocols are calculated in $\bmod\, m_s^2$.

Fig. 2 shows the overview of the setup phase and the protocols. Here, we briefly explain the steps of the server-based solutions and illustrate these steps with an example scenario for RNNQ/S. The server-based protocols consist of 10 steps. Steps 1, 4, 7, and 9 are the communication steps. In the first step, the client sends the query and the facility locations ($\mathcal{F}$) to the server. Step 2 is the calculation of distances between facilities and users. The server determines the nearest facility for each user. Since encrypted values cannot be decrypted by the client, the server computes encrypted values based on nearest facility of each user in Step 3 to hide $\mathcal{U}_\mathcal{S}$ and user locations (sensitive data 2(a) & 2(b)) from the client. Using the encrypted values, the client calculates the ciphertext of the query result by utilizing homomorphic properties of Paillier cryptosystem in Step 5. To hide $\mathcal{U}_\mathcal{C}$ and

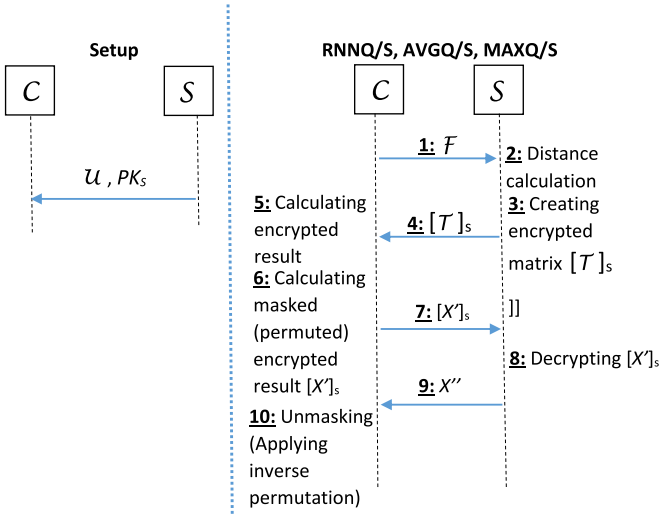

Fig. 2. Overview of the server-based protocols.

the query result (sensitive data 1 & 3) from the server, the client masks the encrypted query result in Step 6 before sending to the server for decryption. The server decrypts the encrypted masked result in Step 8 and obtains the masked result. Due to masking in Step 6, the server cannot deduce the query result. In Step 10, the client applies unmasking and finds the query result.

Let the identifier used by the server and the client consists of one digit, and id numbers of the users of the server be 1,3,5,6,7,9. The server can select the superset $\mathcal{U} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ such that $\mathcal{U}_S \subset \mathcal{U}$. Assume that we have two facilities F_1 and F_2. When the client requests RNNQ/S, the server determines the nearest facility of its six users. Let F_1 be the nearest facility of the users 1, 6, and 9, F_2 be the nearest facility of the users 3, 5, and 7. In Step 3, the server computes $[T_1]_s = \{E_s(0), E_s(1), E_s(0), E_s(0), E_s(0), E_s(0), E_s(1), E_s(0), E_s(0), E_s(1)\}$ for F_1 and $[T_2]_s = \{E_s(0), E_s(0), E_s(0), E_s(1), E_s(0), E_s(1), E_s(0), E_s(1), E_s(0), E_s(0)\}$ for F_2. The server sends these encrypted values $[\mathcal{T}]_s$ to the client in Step 4. Let id numbers of the users of the client be 1,2,3,5. In Step 5, the client calculates two ciphertexts for two facilities by multiplying the ciphertexts of its users in $[\mathcal{T}]_s$. That is, $[x_1]_s = [T_{1,1}]_s \cdot [T_{1,2}]_s \cdot [T_{1,3}]_s \cdot [T_{1,5}]_s$ and $[x_2]_s = [T_{2,1}]_s \cdot [T_{2,2}]_s \cdot [T_{2,3}]_s \cdot [T_{2,5}]_s$. These values are the encryption of the query results such as $[x_1]_s = E_s(1)$ and $[x_2]_s = E_s(2)$. Let two random values selected by the client in Step 6 be 15 and 11. The client encrypts these random values and sends $[x'_1]_s = [x_1]_s \cdot E_s(15)$ and $[x'_2]_s = [x_2]_s \cdot E_s(11)$ to the server. The server decrypts these values in Step 8 and obtains $x''_1 = 16$ and $x''_2 = 13$. When the client receives these masked values, it subtracts the random values and obtains $q_1 = 1$ and $q_2 = 2$. Therefore, the client learns the RNN cardinality of F_1 and F_2.

4.1 Security Analysis of Server-Based Protocols

In this section, we prove the security of the server-based protocols in the semi-honest model. Semi-honest parties follow the protocol correctly; however, they may try to learn additional information by analyzing the messages they receive throughout the protocol. In general, in secure two-party protocol, the goal of the parties is to compute a desired output pair $f(x, y) = (f_1(x, y), f_2(x, y))$ from their inputs x and y without revealing them to each other. The first party wants to obtain $f_1(x, y)$ and the second party wants to obtain $f_2(x, y)$ at the end of the protocol. During the protocol, the view of a party consists of its input, its random-tape, and sequence of incoming messages throughout the protocol. A protocol privately computes $f(x, y)$ if a party's view can be simulated from its input and output [15].

More formally, let Π be a secure two-party protocol for computing $f(x, y)$. The views of the parties are denoted as $\text{VIEW}_1^{\Pi}(x, y)$ and $\text{VIEW}_2^{\Pi}(x, y)$. Then, the security of a deterministic protocol in semi-honest model is defined as follows [15]:

Definition 3. *The protocol Π privately computes $f(x, y)$ if there exist probabilistic polynomial-time simulators Sim_1 and Sim_2 such that*

$$\{Sim_1(x, f_1(x, y))\} \stackrel{c}{\equiv} \{\text{VIEW}_1^{\Pi}(x, y)\}$$

$$\{Sim_2(x, f_2(x, y))\} \stackrel{c}{\equiv} \{\text{VIEW}_2^{\Pi}(x, y)\},$$

where $\stackrel{c}{\equiv}$ implies computational indistinguishability. Therefore, a party's privacy is guaranteed if there exists a simulator that can generate a view indistinguishable from the view of the opposing party. In the following, we prove the security of the server-based protocols using this simulation paradigm.

Let the client be the first party and the server be the second party in our protocols. The private input x of the client is $\mathcal{U}_C$ and the private input y of the server is $\mathcal{U}_S$ and the user locations. $\mathcal{F}$ is also the input of the protocol, which is commonly known by the server and the client. As discussed in Section 3.3, it should not be hidden from the server to prevent attacks via exhaustive client queries. In addition, PK_s, PK_c, and $\mathcal{U}$ are also known by the server and the client as background information. As discussed before, $\mathcal{U}$ is the superset of the users for keeping the user list of parties from each other. The client should get *query result* as $f_1(x, y)$ at the end of the protocol while the server receives no output (i.e., $f_2(x, y) = \perp$).

Since the steps of the server-based protocols are similar as shown in Fig. 2, we consider RNNQ/S in the proof. The security of the other protocols can be proved similarly. In RNNQ/S, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ is the query result where q_i is the total number of users in $\mathcal{U}_{\mathcal{I}}$ whose nearest facility is F_i. Therefore, the view of the client (VIEW_1) consists of $\mathcal{U}_C$, $\mathcal{F}$, $[\mathcal{T}]_s$, and $\mathcal{Q}$. To prove that the server's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_1 such that $Sim_1(\mathcal{U}_C, \mathcal{F}, \mathcal{Q})$ is computationally indistinguishable from VIEW_1. Since $[\mathcal{T}]_s$ contains $n \cdot k$ Paillier ciphertexts, Sim_1 can generate $n \cdot k$ random numbers between 0 and m_s^2 and these numbers are computationally indistinguishable from the ciphertexts in $[\mathcal{T}]_s$ due to the semantic security of Paillier cryptosystem.

On the other hand, the view of the server (VIEW_2) consists of $\mathcal{U}_S$, user locations, $\mathcal{F}$, and X''. To prove that the client's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_2 such that $Sim_2(\mathcal{U}_S$, user locations, $\mathcal{F})$ is computationally indistinguishable from VIEW_2. This is satisfied by letting Sim_2 generate k random numbers between 0 and m_s to

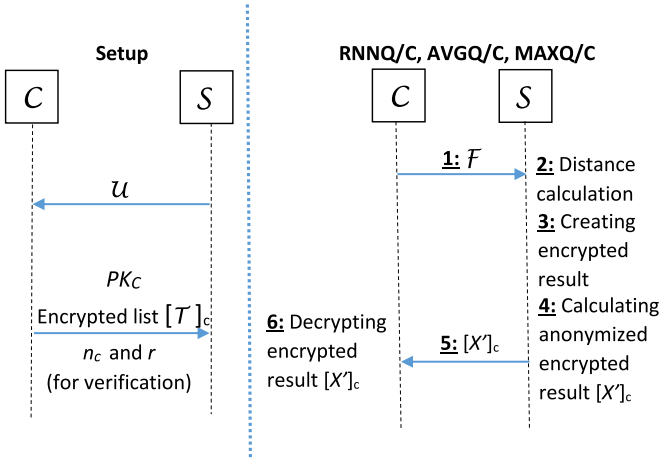

Fig. 3. Overview of the client-based protocols.

simulate X'' because X'' contains k values $\{q_1 + v_1, \ldots, q_k + v_k\}$ where each v_i is a randomly selected number by the client. Thus, we conclude that RNNQ/S protocol securely processes RNN Cardinality queries in semi-honest model.

Although the server-based protocols preserve privacy in semi-honest model, they can be vulnerable to the attack of a malicious client. A malicious client can calculate the encrypted result in Step 5 for a specific customer U_i. Therefore, the client can obtain information about the location of U_i such as the nearest facility of U_i and its distance to the nearest facility. However, in any case, it is not possible to find the exact location of U_i. To prevent the defined attack by malicious clients while providing the exact query result, we propose client-based protocols in Section 5. Moreover, Section 6 explains satisfying differential privacy in server-based protocols. To protect the privacy of individuals from these kinds of attacks, differential privacy gives a guarantee that presence or absence of an individual will not affect the final output of the algorithm significantly. When the queries return noisy results instead of exact results, a malicious client cannot obtain the nearest facility of a specific user U_i and its distance to the nearest facility. For instance, let F_1 be the nearest facility of U_i. Then, the exact query result is $(1, 0, 0, \ldots, 0)$ for the defined attack. However, adding a noise to each of these values will prevent the information leak about U_i. Therefore, differential privacy provides privacy guarantees against such attacks from the malicious client.

5 Client-Based Query Processing Protocols

In the protocols defined in Section 4, the data is encrypted with the public key of the server. The server computes most of the encryptions, which dominates the computation cost. In this section, we propose protocols using the public and private keys (PK_c, SK_c) of the client, where the client computes the majority of the encryptions, however, instead of performing encryptions during each query, the client performs encryptions in the setup. This makes the setup phase of these protocols more costly than the protocols in Section 4, however, query processing in these protocols is more efficient in terms of computation and communication costs. The protocols defined in this section also return exact query results as in Section 4. We describe achieving differential privacy during the client-based protocols in Section 6.

In the setup phase, the client generates a public and private key pair (PK_c, SK_c) for Paillier cryptosystem. The client shares $PK_c = (g_c, m_c)$ with the server. All multiplications and exponentiations of ciphertexts in the client-based protocols are calculated in $\bmod\ m_c^2$. In addition, the server selects a superset $\mathcal{U} = \{U_1, \ldots, U_n\}$ and shares with the client, as described in Section 4. Then, for each $U_i \in \mathcal{U}$, the client calculates $[T_i]_c = E_c(0)$ if $U_i \notin \mathcal{U}_\mathcal{C}$ and $[T_i]_c = E_c(1)$ if $U_i \in \mathcal{U}_\mathcal{C}$. The client sends $[\mathcal{T}]_c = \{[T_1]_c, \ldots, [T_n]_c\}$ to the server. Let r_i be the random number used in the calculation of $[T_i]_c$. To prevent malicious client attack described in Section 4.1, the client sends the total number of its users (n_c) and $r = \prod_{i=1}^{n} r_i$ to the server. The server multiplies all $[T_i]_c$ values and obtains a ciphertext which should be equal to encryption of n_c. That is, $E_c(n_c) = g_c^{n_c} \cdot r^m = \prod_{i=1}^{n} [T_i]_c \pmod{m_c^2}$. The server encrypts n_c with the random value r and verifies the total number of the client's users. If $\prod_{i=1}^{n} [T_i]_c$ is not equal to $E_c(n_c)$ or n_c is less than a threshold value, the server aborts the protocol. Therefore, a malicious client cannot get the query result for a specific user.

Once the client sends n ciphertexts to the server, any of the aforementioned queries can be performed with small computation and communication overheads. We can assume that the users of the client do not change frequently. Small number of changes on the user list do not have a notable effect on query results as well. Hence, the client can update the encrypted list $[\mathcal{T}]_c$, when there is a significant change on its user list. In addition, when the client decides an update in $[\mathcal{T}]_c$, it is not necessary to update all values in $[\mathcal{T}]_c$. The client can only update a subset of users that contains the users to be changed. For instance, if the superset $\mathcal{U}$ includes 100 million users, to change 100 users in $[\mathcal{T}]_c$, the client can update a subset of $[\mathcal{T}]_c$ containing one million users instead of all users in $[\mathcal{T}]_c$.

Fig. 3 shows the overview of the setup phase and the protocols. The protocols in this section consist of 6 steps. The server and the client communicate in Steps 1 and 5. Step 2 is the calculation of distances as in server-based protocols. In Step 3, the server utilizes homomorphic properties of Paillier cryptosystem to calculate the encryption of the query result by using encrypted values in $[\mathcal{T}]_c$. Before sending the encrypted result to the client, the server anonymizes the result by multiplying it with the encryption of zero in Step 4. This multiplication does not alter the result; it only prevents the server from tracking users by the client. Therefore, the server hides user locations from the client. In Step 6, the client obtains the query result after decryption. Since the server only receives the locations of the facilities during query processing, it is not possible for the server to determine query result.

5.1 RNN Cardinality Query (RNNQ/C)

Let q_i be the total number of users in $\mathcal{U}_\mathcal{I}$ whose nearest facility is F_i. This query returns the q_i values for each facility $F_i \in \mathcal{F}$. Hence, $\mathcal{Q} = \{q_1, \ldots, q_k\}$ is the query result. Fig. 4 illustrates the steps of the protocol for the same example scenario explained in Section 4. The protocol is defined as follows:

- *Step 1:* $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- *Step 2:* $\mathcal{S}$ checks the facility locations and aborts the protocol if $\mathcal{C}$ adds more than θ_1 new facilities or

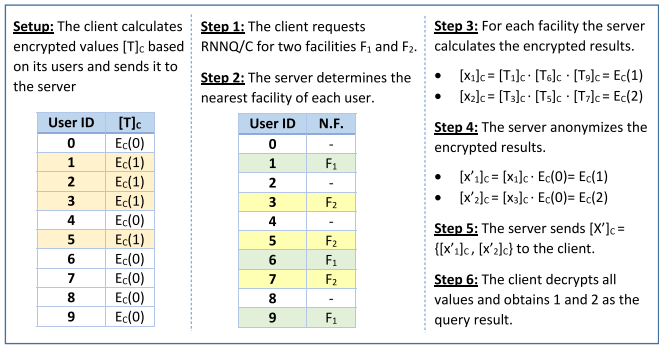

Fig. 4. An example scenario for RNNQ/C.

removes more than θ_2 existing facilities as described in Section 3.3. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$.

- *Step 3:* For each facility F_j, $\mathcal{S}$ calculates the $\left[x_j\right]_c$ value by multiplying $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$ and the nearest facility of U_i is F_j. At the end of this step, $\mathcal{S}$ forms $[X]_c = \{[x_1]_c, \ldots, [x_k]_c\}$ where $[x_i]_c$ is the encryption of q_i. In this step, $\mathcal{S}$ computes the encrypted result.
- *Step 4:* $\mathcal{S}$ encrypts 0 using k different random values and calculates $\left[x'_i\right]_c = [x_i]_c \cdot E_c(0)$ for each $i \in \{1, \ldots, k\}$.
- *Step 5:* $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, \ldots, [x'_k]_c\}$ to $\mathcal{C}$.
- *Step 6:* $\mathcal{C}$ decrypts all $[x'_i]_c$ values in $[X']_c$, and clearly, $D_c([x'_i]_c)$ is equal to q_i. $\mathcal{C}$ obtains $\mathcal{Q} = \{q_1, \ldots, q_k\}$.

5.2 Average Distance Query (AVGQ/C)

Let q be the average distance between users in $\mathcal{U}_\mathcal{I}$ and each one's nearest facility. The protocol is defined as follows:

- *Step 1:* $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- *Step 2:* As described in RNNQ/C protocol, the server aborts the protocol if it detects a threat. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility.
- *Step 3:* $\mathcal{S}$ calculates the multiplication of $[T_i]_c^{d_i}$ values and the multiplication of $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$. That is, $[x_1]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S}} [T_i]_c^{d_i}$ and $[x_2]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S}} [T_i]_c$. Clearly, $[x_1]_c$ is equal to $E_c(q \cdot n_I)$ and $[x_2]_c$ is equal to $E_c(n_I)$.
- *Step 4:* $\mathcal{S}$ calculates $[x'_1]_c = [x_1]_c \cdot E_c(0)$ and $\left[x'_2\right]_c = [x_2]_c \cdot E_c(0)$.
- *Step 5:* $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, [x'_2]_c\}$ to $\mathcal{C}$.
- *Step 6:* $\mathcal{C}$ decrypts $[x'_1]_c$ and $[x'_2]_c$. Clearly, $D_c([x'_1]_c)$ is equal to $q \cdot n_I$ and $D_c([x'_2]_c)$ is equal to n_I. $\mathcal{C}$ obtains q after division.

5.3 Maximum Distance Query (MAXQ/C)

Let q be the maximum distance between a user in $\mathcal{U}_\mathcal{I}$ and her nearest facility. The protocol is defined as follows:

- *Step 1:* $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- *Step 2:* As described in RNNQ/C protocol, the server aborts the protocol if it detects a threat. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility. Let max be the maximum distance between a user in $\mathcal{U}_\mathcal{S}$ and her nearest facility. $\mathcal{S}$ selects a value w, which is greater than max.
- *Step 3:* For each $j \in \{1, .., w\}$, $\mathcal{S}$ calculates the multiplication of $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$ and $d_i = j$. That is, $\mathcal{S}$ computes $[x_j]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S} \& d_i = j} [T_i]_c$. If there is no such U_i, $\mathcal{S}$ sets $[x_j]_c = E_c(0)$. Therefore, $\left[x_j\right]_c$ is equal to the encryption of the total number of users in $\mathcal{U}_\mathcal{I}$ whose distance to the nearest facility is equal to j. The query result q is equal to the maximum j value such that $D_c(\left[x_j\right]_c) \neq 0$.
- *Step 4:* At the end of the protocol, $\mathcal{C}$ should not learn anything more than the query result. To hide the $\left[x_j\right]_c$ values from $\mathcal{C}$, $\mathcal{S}$ randomizes the $\left[x_j\right]_c$ values by exponentiation. $\mathcal{S}$ selects w random values $\{v_1, \ldots, v_w\}$. Then, $\mathcal{S}$ calculates $\left[x'_i\right]_c = [x_i]_c^{v_i}$ for each $i \in \{1, 2, \ldots, w\}$. If $[x_i]_c$ is the encryption of 0, $\left[x'_i\right]_c$ is the encryption of 0. Therefore, q is still equal to the maximum j value such that $D_c([x'_j]_c) \neq 0$.
- *Step 5:* $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, \ldots, [x'_w]_c\}$ to $\mathcal{C}$.
- *Step 6:* $\mathcal{C}$ decrypts all $\left[x'_i\right]_c$ values. $\mathcal{C}$ obtains q, since it is equal to the maximum j value such that $D_c([x'_j]_c) \neq 0$.

5.4 Security Analysis of Client-Based Protocols

In this section, we prove the security of the client-based protocols using the simulation paradigm described in Section 4.1. To prove the security of the protocols we need to show that there exists two probabilistic polynomial-time simulators Sim_1 and Sim_2 for simulating the views of the client and the server, respectively. In client-based protocols, the view of the client only consists of its input and output. The server only sends the encrypted query result to the client in Step 5. Since, the encrypted result is anonymized in Step 4, $[X']_c$ does not contain any information about the users. Therefore, the view of the client can obviously be simulated by Sim_1 and the privacy of the server is assured.

The view of the server (VIEW$_2$) consists of $\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F}$, and $[\mathcal{T}]_c$. To prove that the client's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_2 such that $Sim_2(\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F}) \stackrel{c}{\equiv}$ VIEW$_2$. This is satisfied by letting Sim_2 generate n random numbers between 0 and m_c^2. These numbers are computationally indistinguishable from the ciphertexts in $[\mathcal{T}]_c$ due to the semantic security of Paillier cryptosystem. Hence, we conclude that the client-based protocols privately process the queries in semi-honest model.

6 Protocols with Differential Privacy

Differential privacy is a framework to formalize privacy in statistical databases. The security proofs indicate that the proposed protocols reveal no more information than the output of the queries. However, providing aggregate statistical information about a database may reveal information about the individuals in the dataset. All of the queries (RNNQ, AVGQ, and MAXQ) that are studied in this paper return aggregate results and these query results may cause information leaks in some cases. For example, RNNQ returns the cardinality of $RNN(F_i)$ for each facility F_i in $\mathcal{F}$.

If the RNN cardinality of a facility is 1, this user can be predicted with background knowledge. However, only a region containing the user's location can be inferred. In any case, it is not possible to find the exact location of a user.

The protocols defined in Sections 4 and 5 return exact query results. To achieve differential privacy in these protocols, we need to add controlled random noise to the query result. As discussed in Section 2.3, one needs to define the sensitivity of a query to determine the amount of noise to be added to the result of a query. Now, we show the sensitivity of each considered query and how to add the noise during the protocols.

RNNQ. Returns the total number of users attracted by each facility. It can be thought as a histogram query [10] and its sensitivity is 2. When there is a single change in the database, RNN of at most two facilities may change. Therefore, we add a noise $Laplace(\frac{2}{\epsilon})$ to the RNN cardinality of each facility.

AVGQ. Returns two values: (i) the total number of users in $\mathcal{U}_I$ (n_I) and (ii) the total distance between each user and her nearest facility ($q \cdot n_I$). Thus, we need to calculate the sensitivity for both subqueries. Since the total number of users is a counting query, the sensitivity for n_I is 1. For the total distance, the sensitivity is the maximum distance (max) between a user in $\mathcal{U}_S$ and her nearest facility. Therefore, we add the noise from $Laplace(\frac{1}{\epsilon})$ to n_I and $Laplace(\frac{max}{\epsilon})$ to $q \cdot n_I$.

MAXQ. Returns w[3] values containing zero and non-zero elements. The largest index of a non-zero element is the result of the query. In MAXQ each of w values can be considered as a counting query, and hence the sensitivity of each one is 1. Therefore, we add $Laplace(\frac{1}{\epsilon})$ to each w values.

In the server-based protocols, the server adds noise to the query result in Step 8. Before sending the masked result X'' to the client, the server adds noise to the masked result. When the client applies unmasking in Step 10, it obtains the noisy result instead of the exact result.

In the client-based protocols, the server adds noise to the query result in Step 4. Before sending the encrypted result to the client, the server anonymizes the result by multiplying it with the encryption of zero in the client-based protocols. Instead of encrypting zero values, the server encrypts the values drawn from the Laplace distribution and multiplies the encryption of the noise with the encrypted query result. Due to homomorphic properties of Paillier cryptosystem, the noise will be added to the query result in plaintext. When the client decrypts query result in Step 6, it obtains the noisy result instead of the exact result.

7 Evaluation

In this section, we analyze the complexity, performance, and the utility of the proposed protocols. As there is no existing work that solves the stated problems, we only show the feasibility of our solutions. First, we analyze the computation complexity and the communication costs theoretically in Section 7.1. In Section 7.2, we present the experimental efficiency evaluation of each protocol with respect to different parameters. In Section 7.3, we show the utility of the protocols when differential privacy is achieved.

3. w is a random number that is selected by the server in the MAXQ/S and MAXQ/C protocols

TABLE 2
Computation Performed in Proposed Protocols

	$\mathcal{S}$	$\mathcal{C}$
RNNQ/S	$n_s \cdot k$ dist. $n \cdot k$ enc. k dec.	$n_c \cdot k$ mult. k enc.
AVGQ/S	$n_s \cdot k$ dist. $2 \cdot n$ enc. 2 dec.	$2 \cdot n_c$ mult. 2 enc. 1 div.
MAXQ/S	$n_s \cdot k$ dist. $n \cdot w$ enc. w dec.	$w \cdot (n_c - 1)$ mult. w exp. 2 per.
RNNQ/C	$n_s \cdot k$ dist. k enc. $n_s + k$ mult.	k dec.
AVGQ/C	$n_s \cdot k$ dist. 2 enc. $2 \cdot n_s$ mult. n_s exp.	2 dec. 1 div.
MAXQ/C	$n_s \cdot k$ dist. n_s mult. w exp. $\leq w$ enc.	$w - q + 1$ dec.

7.1 Complexity Analysis

In this section, we analyze the computation and communication costs of the proposed protocols in Sections 4 and 5. Achieving differential privacy as described in Section 6 does not change the communication costs of the protocols. Moreover, its effect on computation time is negligible because only overhead to achieve differential privacy is producing the noise drawn from the Laplace distribution. Therefore, we give the computation costs of the protocols as described in Sections 4 and 5.

Server-Based Protocols. Table 2 shows the total number of operations performed during server-based protocols in terms of total number of encryptions, decryptions, multiplications, exponentiations, distance calculations, and permutations. In all protocols, encryptions dominate the computation times. The number of encryptions is proportional to n and the server performs at least n encryptions in each query. However, the server encrypts 0 or 1 in each encryption and it can encrypt these values offline before the protocol. When the server uses precomputed $E_s(0)$ and $E_s(1)$ values in these protocols, computation cost reduces significantly. In addition, all of these ciphertexts must be transferred to the client in each query. Hence, the computation costs of RNNQ/S, AVGQ/S, and MAXQ/S are $n \cdot k$, $2 \cdot n$, and $n \cdot w$ ciphertexts, respectively.

Client-Based Protocols. In the setup phase of the client-based protocols, n encryptions are computed by the client. The client sends these n ciphertexts to the server in the setup. Therefore, the communication overhead of the setup is n ciphertexts. After completion of the setup phase, all queries can be processed with small computation and communication overheads. Table 2 shows computation costs of client-based protocols in each query. Total number of encryptions in each query is very small with respect to the server-based protocols. The computation costs of RNNQ/C, AVGQ/C, and MAXQ/C are k, 2, and w ciphertexts, respectively.

7.2 Efficiency

We have implemented the protocols in Java and we used the implementation in [18] for Paillier cryposystem. All experiments were performed on a 64-bit Windows 7 machine with 2.6 GHz Intel Core i5 processor and 4 GB of RAM. We used 1,024-bit modulus m_s and m_c in our tests and each ciphertext consists of 2,048 bits. All distances were calculated by the server in the euclidean metric.

In our experiments, we used real datasets [27] containing 227,428 check-ins in New York City and 573,703 check-ins

in Tokyo. The x and y coordinates were scaled to integer values from 1 to 10,000. Since the total number of users in the datasets is less than 5,000, we considered each check-in location as the location of a separate user. Therefore, $n_s = 227,428$ in NYC dataset and $n_s = 573,703$ in Tokyo dataset. We randomly chose 20 percent of them as the users of the client. For existing facilities, we used the locations of 20 restaurants of a fast food chain in New York and 10 restaurants of a fast food chain in Tokyo.

For synthetic datasets, the x and y coordinates of the user locations and facility locations were selected randomly as integer values from 1 to $maxCoordinate$. The user id values in the superset $\mathcal{U}$ were selected as the numbers from 1 to n. We randomly chose n_s of them as the users of the server and n_c of them as the users of the client. The key parameters in the implementation were n, n_s, n_c, n_I, k, $maxCoordinate$, m_s, and m_c (introduced in Table 1). w is another parameter in the Maximum Distance Query, which depends on the value of $maxCoordinate$. We present the experimental evaluation of the server-based protocols in Section 7.2.1 and the client-based protocols in Section 7.2.2.

7.2.1 Server-Based Protocols

When we use 1,024-bit m_s in Paillier encryption, one million encryption nearly takes 2 hours and 45 minutes and the size of one million ciphertexts is 250 MB. For the protocols RNNQ/S, AVGQ/S, and MAXQ/S, the computation times and communication costs are directly proportional to $n \cdot k$, $2 \cdot n$, and $n \cdot w$, respectively. Therefore, when n is one million, the computation time of each protocol is more than 2 hours and 45 minutes. Moreover, when n is one million, the amount of data exchanged during each protocol is more than 250 MB.

In our experiments, we set $n = 1,000,000$, $n_s = 100,000$, $n_c = 20,000$, $k = 25$, and $maxCoordinate = 10,000$ in the synthetic dataset. Running time of RNNQ/S with these parameters is high because it requires $n \cdot k$ encryptions for the encrypted matrix $[\mathcal{T}]_s$. However, all of these ciphertexts in $[\mathcal{T}]_s$ are either the encryption of 0 or the encryption of 1. Therefore, the encrypted values in $[\mathcal{T}]_s$ can be computed offline by the server. When the server precomputes $E_s(0)$ and $E_s(1)$ values before the protocol, the remaining computation takes 10 seconds for these parameters. For the NYC and Tokyo datasets, the query takes 20 and 25 seconds, respectively. Similarly, for the synthetic dateset with given parameters, AVGQ/S takes 13.5 seconds, when the server computes $E_s(0)$ and $E_s(1)$ values before the protocol. Since the computation time of AVGQ/S is directly proportional to n_s, the query takes nearly 35 and 70 seconds for the NYC and Tokyo datasets, respectively. The computation time of MAXQ/S mostly depends on the value of w, which is a randomly selected number by the server. The server performs w decryptions and the client performs w exponentiations and $w \cdot (n_c - 1)$ multiplications. For instance, when w is selected as 500, the computation time of MAXQ/S is nearly 5 minutes and it increases linearly when w increases. When w remains same, we observed the similar computation times for the real datasets.

Our experimental results show that the computation at the client's side is low in server-based query processing protocols. Step 3 of these protocols necessitates calculating an encrypted matrix $[\mathcal{T}]_s$. This step dominates the computation time of server-based protocols. However, the encrypted values can be computed offline by the server. To do offline computation, the server does not need to know the facility locations. The server can compute $E_s(0)$ and $E_s(1)$ values before the protocol. When the client sends the facility locations in a protocol, the server uses previously computed ciphertexts in the encrypted matrix $[\mathcal{T}]_s$. Hence, if the server computes these encryptions offline before the protocol, the remaining computation takes a few minutes on a single computer for millions of users. In addition, the computation time of calculating $[\mathcal{T}]_s$ can be reduced via parallel computations because all encryptions are independent. Server-based protocols can be preferable when the client cannot afford to perform the computations or when the client wants to outsource all the computations to the server. However, as we have shown, the encrypted values in $[\mathcal{T}]_s$ must be transferred to the client in each query.

7.2.2 Client-Based Protocols

In Section 7.1, the computation complexity of each client-based protocol is given. When the client performs several queries, some of these computations are common. For instance, the server calculates the distance between each facility and each user in each query. For the same facilities in separate queries, the server does not need to calculate the same distance values. In our system model, the locations of the existing facilities are considered as public and known by the server. The client can share these locations in the setup phase. Since the server knows the locations of the existing facilities, we assume that all the distances between users and existing facilities were calculated and the nearest facility of each user was determined by the server before the execution of protocols. In each query, the client sends a possible location for adding a new facility and the server only calculates the distance between the new location and each user. The server only updates the nearest facilities of the users who are attracted by the new facility. Therefore, we evaluate the following for each protocol under different parameter settings:

- *Precomputation time.* Most of the computation given in Table 2 can be precomputed by the server because the locations of the existing facilities are known by the server. Hence, we evaluate the precomputation time of each protocol separately.
- *Query processing time.* Once the server completes the precomputation, processing of each query requires low computation overhead. We evaluate the query processing time when the client sends a possible location for adding a new facility.
- *Amortized computation time.* When the client requests n_q queries, amortized computation time of a query is equal to ((precomputation time) + n_q· (query processing time)) / n_q.

In the setup phase of the client-based protocols, the client computes n ciphertexts and shares them with the server. Therefore, the computation cost and the communication cost of the setup phase of the client-based protocols are directly proportional to n. One ciphertext consists of 2,048 bits and one encryption takes 10 ms on the machine

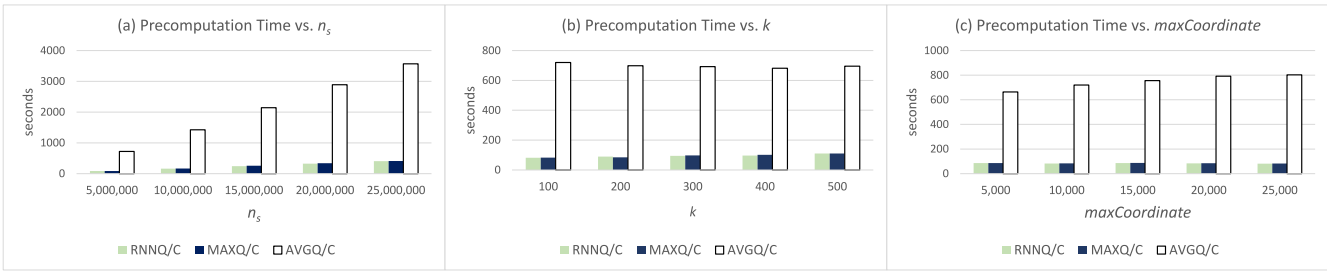

Fig. 5. Precomputation times of client-based protocols.

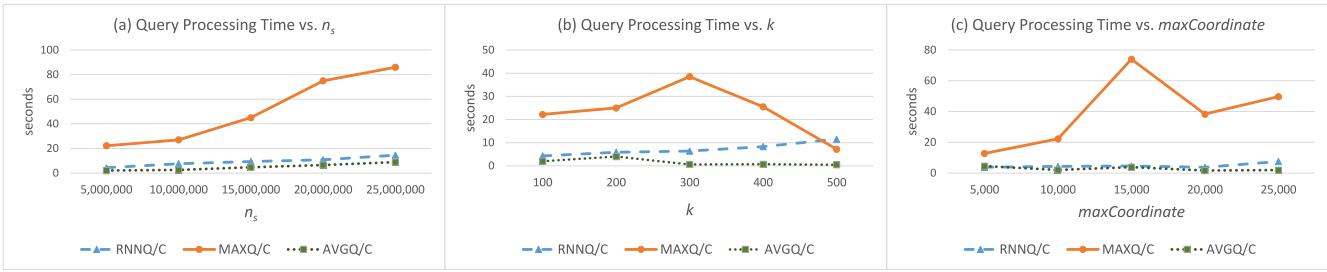

Fig. 6. Query processing times of client-based protocols.

mentioned above. Therefore, if n is one million, the execution time of the setup phase is nearly 2 hours and 45 minutes[4] and the amount of data sent by the client to the server is 250 MB.

Table 2 shows computation costs of client-based protocols including precomputation and query processing. We evaluate the performance of the protocols with respect to n, n_s, n_c, n_I, k, and $maxCoordinate$. As evident in Table 2, the parameters n, n_c, and n_I have no effect on query processing times of the protocols. In our experiments, we observed the similar computation times for different values of these parameters. Therefore, increasing one of these parameters does not change the precomputation time and the query processing time of client-based protocols. For the other parameters n_s, k, and $maxCoordinate$, we analyze their effects on the precomputation time, the query processing time and the communication cost of each client-based protocol. In our experiments, we set $n = 200{,}000{,}000$, $n_s = 5{,}000{,}000$, $n_c = 1{,}000{,}000$, $n_I = 500{,}000$, $k = 100$, and $maxCoordinate = 10{,}000$, unless stated otherwise.

RNNQ/C. Table 2 shows the computation cost of the protocol, where n_s and k are the determining parameters. In this protocol, $n_s \cdot k$ distance calculations, n_s multiplications and k encryptions can be precomputed by the server. The client computes $[X]_c = \{[x_1]_c, \ldots, [x_k]_c\}$ before the protocol. When the client requests a query for a new facility location (F_{k+1}), the server only calculates the distance between F_{k+1} and each user. Then, the client multiplies the $[T_i]_c$ values of the users whose nearest facility is F_{k+1} and calculates $[x_{k+1}]_c$. The client also multiplies the inverse of $[T_i]_c$ values of the same users with the x_j values of their previous nearest neighbors. Therefore, during query processing the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, and nearly $\frac{n_s}{k}$ modular inverse calculations. The client also performs k decryptions during query processing.

4. Computation time can be further reduced via parallel computations.

Although encryption and decryption are more expensive operations than multiplication, the most time consuming part in the precomputation time is n_s multiplications because n_s is much higher than k in our experiments. Therefore, the precomputation time mostly depends on n_s and slightly depends on k. Fig. 5a illustrates the effect of n_s on precomputation time. The precomputation time increases from 82 to 407 seconds, when n_s increases from 5 million to 25 million. As evident in Fig. 5b, the effect of k is not sharp as n_s. For instance, the precomputation takes 82 seconds when k is 100. When k becomes 500, the time increases to 110 seconds. For the NYC and Tokyo datasets, the precomputation time is 4 and 9.6 seconds, respectively, due to the lower n_s values in these datasets.

The query processing time of RNNQ/C depends on the values of n_s and k. Although an increase in k decreases the workload of the server, it increases the total number of decryptions performed by the client. Figs. 6a and 6b shows the query processing time for different values of n_s and k. These two variables are not the only factors that determine the query processing time because the total number of operations also depends on the total number of users attracted by the new facility. Query processing takes 1.9 and 2.5 seconds, for the NYC and Tokyo datasets, respectively.

We also evaluate the amortized computation time of RNNQ/C for 100 queries. For the parameters given above, the precomputation takes 82 seconds and the query processing takes 4.3 seconds. Hence, the amortized computation time per query is 5.1 seconds.

During query processing, k ciphertexts and the facility locations are shared between the server and the client. Hence, k is the most crucial parameter for the communication cost. When the total number of facilities is 100, the amount of shared data is nearly 25 KB.

AVGQ/C. In this protocol, the client obtains the total number of users in $\mathcal{U}_\mathcal{I}$ and the total distance between each user and her nearest facility. Until there is a change on the total number of users, there is no need to compute it in each query.

Hence, the total number of users in $\mathcal{U}_\mathcal{I}$ can be precomputed by the server and the client. This part of precomputation requires n_s multiplications, one encryption, and one decryption. In addition, the server can precompute $n_s \cdot k$ distance calculations, n_s exponentiations, n_s multiplications, and one encryption for the computation of the total distance. During query processing the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, nearly $\frac{n_s}{k}$ exponentiations, and nearly $\frac{n_s}{k}$ modular inverse calculations. The client performs one decryption and one division during query processing.

Due to the high number of exponentiations, the precomputation time of AVGQ/C is higher than the other client-based protocols. Fig. 5a depicts the precomputation time for different values of n_s. Query processing takes nearly 12 minutes when the server has 5 million users and the time changes linearly with respect to the value of n_s. For the smaller n_s values in NYC and Tokyo datasets, the precomputation takes 41 and 92 seconds, respectively. In addition, Fig. 5c shows that the value of $maxCoordinate$ affects the precomputation time slightly. As $maxCoordinate$ increases, exponent values in the computation also increase.

The query processing time of AVGQ/C is directly proportional to n_s and inversely proportional to k. Figs. 6a and 6b shows the query processing time for different values of n_s and k. Since there is no encryption and only one decryption in query processing, AVGQ/C has the lowest query processing time among three client-based protocols. Query processing takes nearly 1 second, for both NYC and Tokyo datasets. Similar to RNNQ/C, the total number of users attracted by the new facility affects the query processing time. The precomputation takes 720 seconds and the query processing takes 2 seconds for the parameters given above. Hence, the amortized computation time per query is 9.2 seconds for 100 AVGQ/C queries.

During the protocol, two ciphertexts and the facility locations are shared between the server and the client. Therefore, the communication cost is low because the facility locations are sent as plaintext and the total number of ciphertexts is two. When the total number of facilities is 100, the amount of shared data is less than 1 KB during query processing.

MAXQ/C. In MAXQ/C, the server can precompute $n_s \cdot k$ distance calculations, n_s multiplications and w encryptions. During query processing, the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, nearly $\frac{n_s}{k}$ modular inverse calculations, and w encryptions. The client also performs $w - q + 1$ decryptions during query processing.

w and n_s are crucial parameters in the precomputation time of MAXQ/C. Similar to RNNQ/C protocol, the precomputation time mostly depends on n_s because n_s is much higher than w in our experiments. Fig. 5a shows the precomputation time with respect to n_s. Since n_s multiplications dominate the computation cost, the precomputation time of MAXQ/C is similar to RNNQ/C. We also observed similar results for the real datasets.

Due to the randomness in the selection of w, the query processing time of MAXQ/C is not directly proportional to n_s or $maxCoordinate$. w is a randomly selected value that is greater than max, which is the maximum distance between a user in $\mathcal{U}_\mathcal{S}$ and her nearest facility. In our experiments, w is selected randomly in the range $[max, 2 \cdot max]$. Therefore, the parameter $maxCoordinate$ affects the value of w, and hence the query processing time. The query processing time of MAXQ/C for different values of $maxCoordinate$ is given in Fig. 6c. As evident in Fig. 6c, query processing time is not directly proportional to $maxCoordinate$. For instance, when $maxCoordinate$ increases from 15,000 to 20,000, the computation time decreases due to a decrease in w and an increase in q. Therefore, the distance of each user to her nearest facility and the query result q also affect the query processing time. In addition, Fig. 6b shows the query processing time for different values of k. Smaller k values may result in higher query processing times because max value may increase in smaller k values. For instance, in real datasets we have smaller k values such as 10 and 20. As a result, the query processing times are 17 and 60 seconds, for the NYC and Tokyo datasets, respectively.

For the parameters given above, the precomputation takes 83 seconds and the query processing takes 22.2 seconds. Therefore, the amortized computation time per query is 23 seconds for 100 MAXQ/C queries. The communication cost of the protocol is w ciphertexts and the facility locations. When w is selected as 5,000, the amount of shared data is nearly 1.25 MB.

All these results show the practicality of our proposed scheme in real-life settings. Any of the aforementioned queries can be performed in less than a minute on datasets that include millions of individuals.

7.3 Utility versus Differential Privacy

In Section 6, we explain how to achieve differential privacy in the proposed protocols by adding controlled noise to the query results, which affects the accuracy of the results. To measure the utility of the protocols under differential privacy, we selected 100 candidate locations for the new facility and observed the results after executing the protocols. We divided the whole region into a 10×10 grid and selected the center of each grid as a candidate location for the new facility. First, we applied the protocols without adding any noise and ranked the 100 candidate locations with respect to their optimality. Then, we executed the protocols by adding controlled noise and observed the impact of differential privacy on the utility. We evaluated the utility of differential privacy for the real and synthetic datasets. For synthetic datasets, we set the parameters as given in Section 7.2.

RNNQ. The objective of using RNNQ is uniformly distributing the cardinality of the RNNs. When the new facility attracts users from dense facilities, the workloads of dense facilities decrease. Hence, balancing workload reduces the wait times by avoiding overloads. We measured the standard deviation of the cardinalities of the RNNs. We sorted 100 candidate locations with respect to the standard deviation after adding the possible location as the new facility. The best candidate is the location that minimizes the standard deviation. We also sorted the candidate locations after achieving differential privacy. Fig. 7 shows the rankings of the candidates for real and synthetic datasets after adding controlled noise. We used three different ϵ values such as 0.01, 0.1, and $\ln 2$, which are typically chosen values in the literature. As evident in Fig. 7, the utility increases when ϵ increases. When ϵ is $\ln 2$, the ranking of the candidates are almost same as the rankings without adding any noise.

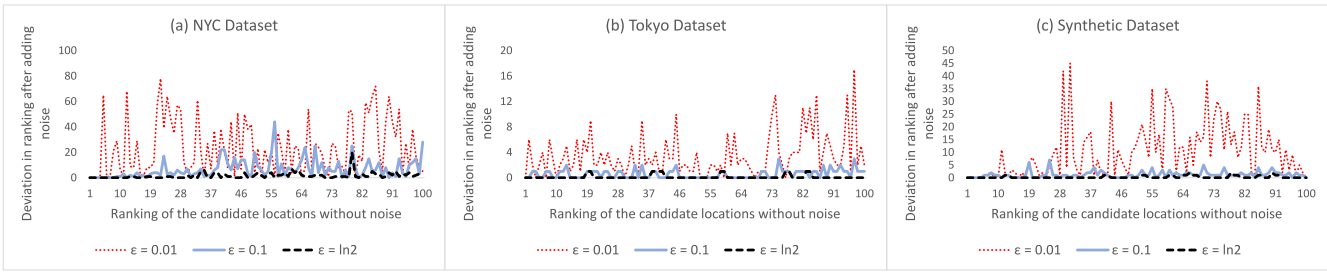

Fig. 7. Deviation in the rankings of the 100 candidate locations after achieving differential privacy in RNNQ. x axis represents the ranking of the candidate locations when RNNQ is performed with exact query results. y axis represents the change in the ranking of each candidate location when differential privacy is achieved in RNNQ. For instance, in Fig. 7a the point $(5, 65)$ for $\epsilon = 0.01$ shows that the 5th best candidate location for the new facility becomes 70th best location after adding noise to the query result. Depending on the maximum change in the ranking, the range of y axis varies for each dataset.

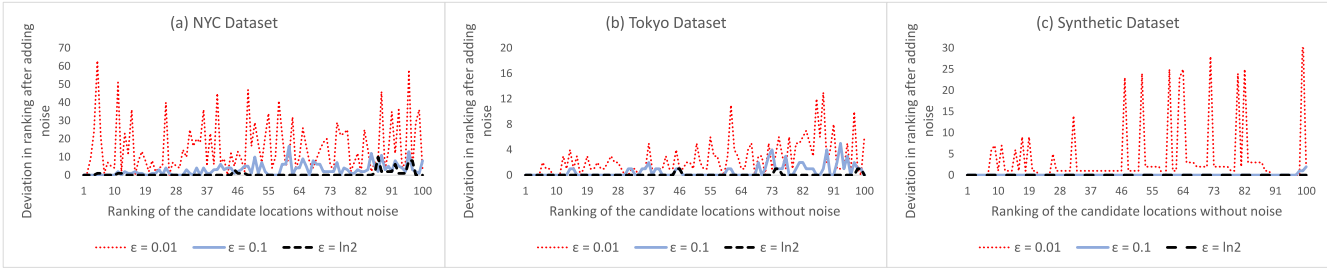

Fig. 8. Deviation in the rankings of the 100 candidate locations after achieving differential privacy in AVGQ.

Although the deviations in the rankings increase for the smaller values of ϵ, the best candidate is same most of the time after achieving differential privacy. Therefore, the utility of RNNQ under differential privacy is remarkable for large ϵ values and acceptable for small ϵ values.

AVGQ. We sorted 100 candidate locations with respect to the average distance value returned from the query. The best candidate is the location that minimizes the average distance. Fig. 8 shows the rankings of the candidates after adding controlled noise. The deviation on the rankings is less than RNNQ. Therefore, the utility of AVGQ under differential privacy is better than RNNQ for all ϵ values.

MAXQ. This query returns w values containing zero and non-zero elements. The largest index of a non-zero element is the result of the query. After adding the noise to each of w values, most of the zero values becomes non-zero. Therefore, adding the noise changes the query result significantly. In our experiments, we observed that the query result becomes w or $w - 1$ most of the time after adding the noise. Since w is a randomly selected value, the query returns a random result in each execution. Hence, the utility of MAXQ under differential privacy is very low.

Discussion. Our experimental results show that achieving differential privacy in RNNQ and AVGQ causes low utility loss. Since these queries contain counting subqueries, running them under differential privacy increases the privacy of individuals with a negligible computational overhead. On the other hand, MAXQ is not suitable for differential privacy because the query result changes significantly after adding noise to each counter value in the query. To prevent high utility loss of differential privacy in MAXQ, only non-zero values should be randomized as described in Section 5.3.

8 Conclusion

We proposed novel protocols for privacy-preserving analysis of location data in a location-based service provider (referred as the server) by a business (referred as the client) as a service. We defined three queries addressing different objectives in optimal location selection: (i) to minimize the average distance between each user and her closest facility, (ii) to minimize the maximum distance between a user and her closest facility, and (iii) to uniformly distribute the workload in facilities. We developed two homomorphic encryption-based solutions: (i) a server-based solution, in which most of the computation is performed by the server, and hence the workload of the client is low, and (ii) a client-based solution, in which the client performs the majority of the computation during the setup phase (which only occurs once) and after completion of the setup phase, all queries are processed quickly. We showed that the proposed protocols keep the client's user list and the query result hidden from the server, and the location information stored at the server hidden from the client. The security provided by all protocols relies on the underlying security of the Paillier cryptosystem (which relies on the decisional composite residuosity assumption) proved in [22]. We also showed that it is possible to achieve differential privacy in the proposed protocols with low utility loss. Using the proposed protocols will facilitate sharing location information between entities without compromising customer privacy. We evaluated the efficiencies of the proposed protocols through experiments for each considered query type and showed that the proposed protocols are feasible, efficient, and scalable.

Acknowledgments

This work is supported in part by Turk Telekom.

References

[1] J. Benaloh, "Dense probabilistic encryption," in *Proc. Workshop Sel. Areas Cryptography*, 1994, pp. 120–128.
[2] J. Cardinal and S. Langerman, "Min-max-min geometric facility location problems," in *Proc. Eur. Workshop Comput. Geometry*, 2006, pp. 149–152.

[3] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long, "Efficient algorithms for optimal location queries in road networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 123–134.
[4] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar, "Preserving user location privacy in mobile data management infrastructures," in *Privacy Enhancing Technologies*. Berlin, Germany: Springer, 2006, pp. 393–412.
[5] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Sci. Rep.*, vol. 3, 2013, Art. no. 1376.
[6] W. Du and M. J. Atallah, "Protocols for secure remote database access with approximate matching," in *E-Commerce Security and Privacy*. Berlin, Germany: Springer, 2001, pp. 87–111.
[7] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems," in *Proc. Workshop New Secur. Paradigms*, 2001, pp. 13–22.
[8] Y. Du, D. Zhang, and T. Xia, "The optimal-location query," in *Advances in Spatial and Temporal Databases*. Berlin, Germany: Springer, 2005, pp. 163–180.
[9] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2008, pp. 1–19.
[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. 3rd Conf. Theory Cryptography*, 2006, pp. 265–284.
[11] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theoretical Comput. Sci.*, vol. 9, no. 3/4, pp. 211–407, 2014.
[12] L. Garber, "Analytics goes on location with new approaches," *Computer*, vol. 4, pp. 14–17, 2013.
[13] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.
[14] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: Anonymizers are not necessary," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 121–132.
[15] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[16] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Advances in Spatial and Temporal Databases*. Berlin, Germany: Springer, 2007, pp. 239–257.
[17] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," *ACM SIGMOD Rec.*, vol. 29, pp. 201–212, 2000.
[18] K. Liu, "Paillier's cryptosystem in Java," (2006). [Online]. Available: http://www.csee.umbc.edu/%7Ekunliu1/research/Paillier.html
[19] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: Query processing for location services without compromising privacy," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 763–774.
[20] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in *Proc. IEEE INFOCOM*, 2014, pp. 754–762.
[21] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1998, pp. 308–318.
[22] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.
[23] J. Qi, R. Zhang, Y. Wang, A. Y. Xue, G. Yu, and L. Kulik, "The min-dist location selection and facility replacement queries," *World Wide Web*, vol. 17, no. 6, pp. 1261–1293, 2014.
[24] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, 2008, pp. 311–319.
[25] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu, "Efficient method for maximizing bichromatic reverse nearest neighbor," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 1126–1137, 2009.
[26] Z. Wu, L. Yu, J. Zhu, H. Sun, Z. Guan, and Z. Chen, "A hybrid approach for privacy preservation in location based queries," in *Web-Age Information Management*. Berlin, Germany: Springer, 2013, pp. 315–326.
[27] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 45, no. 1, pp. 129–142, Jan. 2015.
[28] X. Yi, R. Paulet, E. Bertino, and V. Varadharajan, "Practical k nearest neighbor queries with location privacy," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 640–651.
[29] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive computation of the min-dist optimal-location query," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 643–654.

Emre Yilmaz received the BS degree in computer science and engineering from Sabanci University, Istanbul, Turkey, in 2008 and the MS degree in computer science from ETH Zurich, Switzerland, in 2010. He is continuing working toward the PhD degree in computer engineering at Bilkent University, Ankara, Turkey. His research interests include data privacy, cryptography, and big data analytics.

Hakan Ferhatosmanoglu received the PhD degree in computer science from the University of California Santa Barbara, in 2001. He is a professor with the University of Warwick, UK. His research is on scalable data management and analytics for multi-dimensional data. He received Research Awards from the US Department of Energy, US National Science Foundation, The Academy of Science of Turkey, and Alexander von Humboldt Foundation.

Erman Ayday received the MS and PhD degrees from the Georgia Institute of Technology, in 2007 and 2011, respectively. He is an assistant professor with Bilkent University, Turkey. Before that, he was a post-doctoral researcher with École Polytechnique Fédérale de Lausanne, Switzerland. His research interests include privacy-enhancing technologies (including big data and genomic privacy), wireless network security, trust and reputation management, and recommender systems.

Remzi Can Aksoy received the BS degree in computer engineering from Bilkent University, Turkey, in 2016. He started the graduate studies in computer science at University of Michigan, Ann Arbor. His research interests include data privacy and big data.