A Demonstration of Privacy-Preserving Aggregate Queries for Optimal Location Selection

Cihan Eryonucu[a], Erman Ayday[a], and Engin Zeydan[b]

[a]Computer Engineering Department, Bilkent University, Ankara Turkey
[b]Türk Telekom Labs, Istanbul Turkey 34889

***Abstract*—In recent years, service providers, such as mobile operators providing wireless services, collected location data in enormous extent with the increase of the usages of mobile phones. Vertical businesses, such as banks, may want to use this location information for their own scenarios. However, service providers cannot directly provide these private data to the vertical businesses because of the privacy and legal issues. In this demo, we show how privacy preserving solutions can be utilized using such location-based queries without revealing each organization's sensitive data. In our demonstration, we used partially homomorphic cryptosystem in our protocols and showed practicality and feasibility of our proposed solution.**

***Index Terms*—Privacy, Security, Sensitive Data, Homomorphic Encryption.**

I. Introduction

In today's world, businesses want to decide on the best possible facility locations for their services. Service providers and mobile operators hold their customer location information, but they do not often utilize them. Vertical businesses (e.g. banks, retail industries) seek to utilize their customers location information in order to better serve them, for example by opening a new branch near their customers' most visited locations or providing location based campaigns. At the same time, data owners (e.g. mobile operators) are also eager to service these businesses by providing their own customers' mobile network data in order to obtain value. On the other hand, directly sharing this data causes security, privacy as well as regulative issues for both parties. Hence, data should be shared without businesses allowing other businesses to track their customers or reveal their customers' identities.

In this paper, we denote data owners as mobile operators and service providers, and server and businesses who want to use data owner's data as client. There are three aggregate queries which client can execute. First query type is RNN Cardinality Query (RNNQ). RNNQ is simply, given facility location, finds the number of people who are closest to each facility. Second type of query is Average Distance Query(AVQ). AVQ calculates the average distance of the users to each one's closest facility given the facility locations. Last query type is Maximum Distance Query (MAXQ). MAXQ finds the maximum distance between user and its closest facility given the facility locations. In order to provide solutions to this problem, we need to hide user lists of the both client and server from each other for customers' privacy. In addition, we need to hide result of the query from the server because there exists a possibility that a server might share the result of the query with client's competitors.

In our demonstration, we perform operations on encrypted data using properties of the homomorphic encryption in order to preserve privacy of the both parties. We use client-based protocols of the reference [1]. In our client-based solutions, computation is mostly performed on client's side. In these solutions, there exist a one time setup phase which is performed on client. One time setup lowers the communication number compared to the server-based solution. Therefore, client-based solutions are more efficient and preferable compared to server-based solutions.

Our demo scenario can be seen in Figure 1 where U_c is the set of users of client, U_s is users of server and their intersection is set U_I. In order to identify users, server and client must decide on an identifier before running the queries. This can be users' Mobile Station International Subscriber Directory Number (MSISDN) or national ID numbers. We define superset, U, of users which is the all possible identifiers. For example, if we choose MSISDNs for identification, all possible MSISDNs become the superset. Our protocols are secure in semi-honest models as explained in [1]. In our demonstration, we define the sensitive data of the client and server as: (i) Client's user list, (ii) Server's user list, (iii) Location information of server's users, (iv) Result of the query.

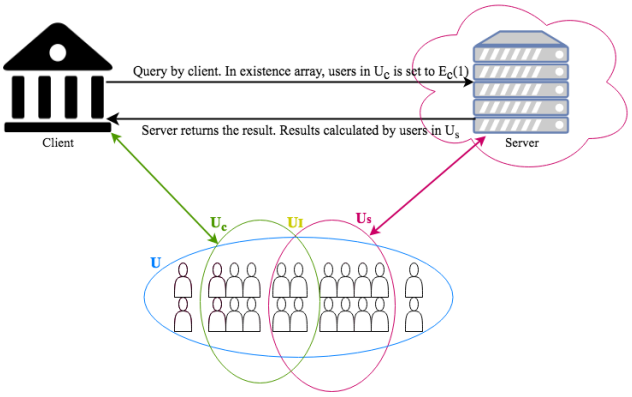

Fig. 1: High Level Demo Architecture

978-1-5386-4725-7/18/$31.00

II. Demonstration Concepts and Approach

For our demonstration, we tested all three query types using client-based model which is efficient in terms of communication and computation cost. We choose MSISDNs as identifiers. Before the queries, setup phase must be completed. In setup phase, client connects to the server and sends the Paillier public key. After the key sharing is accomplished, client also sends the an existence array $[T]_c = [t_i]_c$. Existence array indicates if the user i of client is in the superset or not. Every element in $[T_i]_c$ is either encrypted 0 or 1. If the i'th user is customer of client, $i \in U_c$, then it is $E_c(1)$ otherwise it is $E_c(0)$. In other words, if $i \in U_c$ then $[t_i]_c = E_c(1)$ otherwise $[t_i]_c = E_c(0)$. Setup procedure and an example existence array can be seen in Figure 2. For simplicity purposes, client will not send total number of users and random number as proposed in [1] since we are trying to show how this protocols can be easily implemented and used in real world scenarios. In addition, in our demonstration we completed the setup phase before the queries thus run times does not include setup run times.

A. RNN Cardinality Query

In our protocol for RNNQ, query returns k results which is the number of facility locations client requested. Each result q_i is the i'th location's cardinality result. Procedures in the Figure 3 is matched with procedure below. Figure 4 also shows details of query, particularly step 4. Our protocol is as follows:

1) Client sends the query request for RNN Cardinality Query.
2) Client sends the facility locations as an array of location objects.
3) Server calculates distances between its users $\in U_s$ and facility locations then decides each user's nearest facility location as shown at top right table of Figure 4.
4) After server determines each user's nearest facility, server calculates the result array $[X]_c = [x_j]_c$. Each $[x_j]_c$ is calculated by multiplying all $[t_i]_c$ values where i'th user's nearest facility is j. Using Paillier homomorphic cryptosystem, multiplying a ciphertext gives a addition in plaintext. Server counts the number of users by this operation. If the user exists both in server's and client's list, the server adds encrypted ones whereas if it only exists in server's list, server adds encrypted zeros. An example run of this procedure can be seen in Figure 4. In top left table, users sets and their members can be seen. In top right table, nearest facility of each user of server can be seen. In bottom table, operation in step 4 of RNNQ can be seen. This is step is similar in AVQ and MAXQ
5) As the last operation on server, server multiplies all $[x_j]_c$ with encrypted zeros. Server encrypts each $[x_j]_c$ with different $E_c(0)$ so that server uses different random values for each one. This multiplication only masks the result and will not change it since server adds zero value to result.[2]
6) Server sends the result as a BigInteger array.
7) Client receives the encrypted results and decrypt them one by one using its private key.

B. Average Distance Query

In our protocol for AVQ, query returns the average distance of the users to their nearest facility. Procedures in the Figure 3 are matched with procedure below. Our protocol's workflow as well as descriptions are as follows:

1) Client sends the query request for average distance query.
2) Client sends the facility locations as an array of location objects.
3) Server calculates distances between their users $\in U_s$ and facility locations, then decides each user's nearest facility location.
4) After server determines each user's nearest facility, server calculates two values $[x_1]_c$ and $[x_2]_c$. $[x_1]_c$ is calculated by multiplying all $[t_i]_c^{d_i}$ values which represents i'th user's distance to the nearest facility is d_i. In Paillier homomorphic cryptosystem, raising ciphertext to the power of some number gives multiplication with that power in plaintext. Therefore, server adds the distances of users to the nearest facility by this operation since if the user exists both in server's and client's lists, server multiplies encrypted one with its distance and add them all. $[x_2]_c$ is calculated by multiplying all $[T_i]_c$

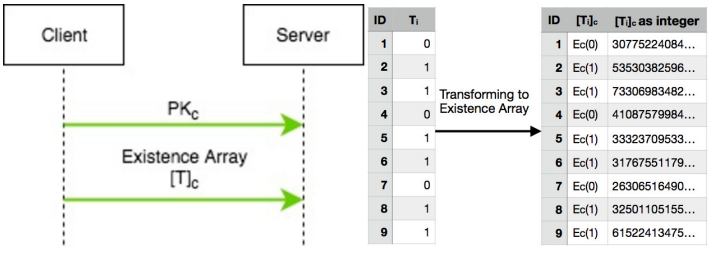

Fig. 2: Setup Phase For All Three Queries and Illustration of Existence Array

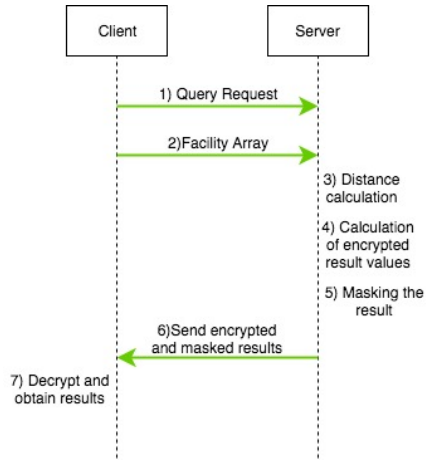

Fig. 3: Overview of the Protocols

$U = 1,2,3,4,5,6,7,8,9$	$U_l = 3,5,8$	Users near to F1	Users near to F2
$U_c = 2,3,5,6,8,9$	$U_s = 1,3,4,5,7,8$	1	3
		5	4
		7	8
$[x_1]_c$	$[t_1]_c * [t_5]_c * [t_7]_c = E_c(0) * E_c(1) * E_c(0) = E_c(1)$		153321054327…
$[x_2]_c$	$[t_3]_c * [t_4]_c * [t_8]_c = E_c(1) * E_c(0) * E_c(1) = E_c(2)$		662792933918…

Fig. 4: EXAMPLE OF STEP 4 OF RNNQ.

values. $[x_1]_c$ is equal to sum of distance of the all users nearest facility. $[x_2]_c$ is equal to the number of all users who exist in both client's and server's user lists, in other words total number of users whose distance is calculated.

5) Server masks the two values again with encrypted zeros. Server encrypts each value with two different $E_c(0)$, so that it uses different random values for each one.
6) Server sends the result as a BigInteger array.
7) Client receives the encrypted results and decrypts them using its private key. After that it divides decrypted $[x_1]_c$ to decrypted $[x_2]_c$ to get average distance.

C. Maximum Distance Query

In our protocol for MAXQ, query returns one result which is the maximum distance of the users to its nearest facility. Procedures in the Figure 3 are matched with procedure below. Our protocol workflow and descriptions are as follows:

1) Client sends the query request for maximum distance query.
2) Client sends the facility locations as an array of location objects.
3) Server calculates distances between their users $\in U_c$ and facility locations and finds the max value between each user and its nearest facility.
4) After server finds the max, it selects a value w which is greater then max. Server calculates each $[x_j]_c$ values. $[x_j]_c$ is basically equal to the number of users whose distance to their nearest facility is j. $[x_j]_c$ is multiplication of $[t_i]_c$ values (i.e. multiplication of the i'th user that exists in the server's user list with the distance of j where j takes ranges from 1 to w).If there exists no users satisfying the above equation, then $[x_j]_c$ is encryption of zero. In our demo, we are not putting $E(0)_c$ to all $[x_j]_c$ values initially. This lowers the computation. Query result is equal to the greatest j possible when $[x_j]_c$ is not zero.
5) For client to learn only its query result, server randomizes all $[x_j]_c$ values by raising $[x_j]_c$ to the power of some random r. If $[x_j]_c$ value is zero, then it will not alter the result. Since client does not search the value of $[x_j]_c$ but highest j, this will not change the result.
6) Server sends the result as a BigInteger array.
7) Client receives the encrypted results and decrypts all $[x_j]_c$ values starting from the $[x_w]_c$ until the $[x_1]_c$. Client stops decrypting whenever it finds a non-zero element. Since client searches for the highest index of non-zero element of all $[x_j]_c$ values, starting from the last is efficient as client stops when it encounters the first non-zero element.

III. DEMONSTRATION SETUP

In this demo, we demonstrate a use case scenario utilizing the above three query types. For our demonstration, we use Java for implementation and utilize Pailler implementation of [**?**]. We test the queries using two different machines. Client is Mac OSX with 1.6 GHz Intel core i5 processor. Server is 64-bit Ubuntu with 1.8 GHz Intel Xeon processor. For considering a real network scenario, we use a server that is located in London and our client machine is located in Turkey during experiments. Our modulus length is 1024 bits and each ciphertext is 2048 bits. The users and their location are generated in our computers, therefore in this demo data is artificial. We use two different settings for the test. Settings details are as follows: (i) First setting has 100 users in its client and server and 5 facility locations. (ii) In second setting, server has $250,000$ users and client has $50,000$ users. In this setting there are again 5 facility locations.

IV. ANALYSIS OF DEMONSTRATION RESULTS

Analysis results of the above two settings for the mentioned three query types are summarized in Table I. For AVQ, we can see the difference and computation cost in the second setting where computation time has significantly increased. However, RNNQ's run-time has not change much where there exists just a slight increase due to non-existence of exponentiation operation in RNNQ. It is observed that MAXQ has the highest computation time compared to all other query types. However, it has run-times as lows as 30 seconds and as high as 350 seconds. This is due to the fact that it performs w exponentiation operations apart from the server that multiplies $w * n_c$ values at worst case where n_c is number of users of client. For the non-privacy preserving solution, for both settings run-times of all three queries are between 0.045 and 0.89 mainly due to non-existence of encryption/decryption process.

TABLE I: COMPARISONS OF COMPUTATION TIME. NON-PRIVACY PRESERVING RUN-TIMES ARE AT RIGHT.

	RNNQ	AVQ	MAXQ
Setting-1	0.5/0.102 s	0.4/0.105 s	144.37/0.112 s
Setting-2	9.12/0.119 s	94.5/0.128 s	183.68/0.133 s

REFERENCES

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

[2] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," pp. 223–238, 1999.