Privacy Preserving Query Processing using Third Parties *

Fatih Emekci Divyakant Agrawal Amr El Abbadi Aziz Gülbeden
Department of Computer Science
University of California Santa Barbara
Santa Barbara, CA 93106, USA
{fatih,agrawal,amr,gulbeden}@cs.ucsb.edu

Abstract

Data integration from multiple autonomous data sources has emerged as an important practical problem. The key requirement for such data integration is that owners of such data need to cooperate in a competitive landscape in most of the cases. The research challenge in developing a query processing solution is that the answers to the queries need to be provided while preserving the privacy of the data sources. In general, allowing unrestricted read access to the whole data may give rise to potential vulnerabilities as well as may have legal implications. Therefore, there is a need for privacy preserving database operations for querying data residing at different parties. In this paper, we propose a new query processing technique using third parties in a peer-to-peer system. We propose and evaluate two different protocols for various database operations. Our scheme is able to answer queries without revealing any useful information to the data sources or to the third parties. Analytical comparison of the proposed approach with other recent proposals for privacy-preserving data integration establishes the superiority of the proposed approach in terms of query response times.

1 Introduction

With the ubiquitous availability of the Internet and all the resources it provides, data integration has emerged as an important practical problem especially from a data management point of view [11, 14]. Techniques used for this purpose commonly assume that the data sources are willing to allow access to all their data without privacy concerns during query processing. In real life, however, owners do not want to share all their data due to various reasons such as competition among data owners or possible legal implications requiring confidentiality of the data. For example, consider a community of medical professionals that need to share data without violating the privacy of either the patients or the organizations. One possible collaboration in such an environment is that a medical professional's office might be interested in all test results of its patients. The results might be in some other medical facilities. For such a query, the involved parties are only interested in finding the test results of patients in the intersection of the different offices and nothing else. In fact, offices might not be allowed to share information about patients that are not in the intersection. Similarly, consider another type of data sharing where medical professionals want to classify the patients according to their total number of visits to all the medical professionals but do not want to violate patients' or medical offices' privacy requirements (i.e., parties do not want to reveal themselves or the total number of visits a patient made to their office). As these examples illustrate, there is an increasing need for privacy preserving database operations for these kinds of queries. Privacy preservation means that the data owner wants to restrict the extra information it provides to other parties. In other words, data owners are only willing to share the minimum information required for processing queries correctly.

The goal of privacy preserving query processing is to execute queries over multiple data sources without revealing any extra information to any of the involved data sources. At the end of the query processing, they will only acquire the query result. For example, if the query is asking for the intersection of lists residing at different data sources, after query processing, the data owners will only know the query result (eg., the intersection of the lists but no information about the elements that are not in the intersection). In order to prevent such privacy violations, any privacy preserving technique should provide a method to execute several crucial operations, in particular *intersection*, *join* and *aggregation* which are all fundamental for general purpose query processing.

*This research was funded in parts by NSF grants IIS IIS 02-23022, and CNF-04-23336.

Several techniques have been proposed to preserve the privacy of data sources in the areas of databases and cryptography. In the *trusted third parties* approach, the data owners hand over their data to a trusted third party and the third party computes the results of the queries [1, 22]. This level of trust is unacceptable for privacy preserving query processing. *Secure multi-party computation* is another technique, where, given m parties and their respective inputs $x_1, x_2, .., x_m$, a function $f(x_1, x_2, ..., x_m)$ is computed such that all parties can only learn $f(x_1, x_2, ..., x_m)$ but nothing else [17, 18, 23]. The computation and the communication complexity of this method makes it impractical for database operations working over a large number of elements. For example, the communication needed to compute the intersection of two lists of size 1 million takes around 144 days with this method [4]. Due to the extreme cost of computation and communication, Agrawal et al. [4] proposed a *minimal information sharing* paradigm to perform intersection and join operations. Their proposed solution is based on using encryption/decryption without the need for third parties. However, this technique has two shortcomings: (1) Since encryption/decryption are costly operations in terms of computation, the query response time is very high (in fact, for some real world applications, this approach may require as much as 40 hours of computation); (2) It cannot be used to aggregate data residing at different sites (i.e., it does not support aggregation queries).

Due to the high overhead of cryptographic operations, and in general, the large number of elements that need to be accessed in database operations, cryptography-based techniques are impractical for query processing in a real environment. Therefore, we propose a new computationally efficient approach that uses third parties to solve this problem. Our scheme does not reveal any extra useful information to any of the third parties used in the computation, nor do the data sources get extra information regarding the other parties' data.

In this paper, we use a *hash based P2P system* to select third parties that perform the computations required for processing privacy preserving queries. P2P systems are particularly suitable for this task due to their self organizing and scalable structure. Furthermore, we exploit the fact that it is possible to perform anonymous communication in a P2P system using the overlay network. Combining these properties, we propose a query processing technique over peer-to-peer systems to speed up query response time while preserving the privacy of the data owners. We use P2P systems not as a data store, which has been the focus of most of the P2P systems. Instead, we employ a P2P system for anonymous communication and computation. Analytical comparison of the proposed method with existing methods shows that our method is able to answer queries with a much lower response time, which in turn makes privacy preserving query processing practical for real life applications.

The rest of the paper is organized as follows. In Section 2 we provide some background information. Section 3 formulates the problem. Section 4 describes query processing in an honest-but-curious environment. The analysis is presented in Section 5. The paper concludes with a discussion of future work.

2 Background

In this section we first present some of the related work in privacy. Then we introduce Shamir's secret sharing method [26] that will be used for sharing a secret with third parties securely.

2.1 Related Privacy Work

Naor et al. [24] proposed a scheme to process intersection queries using oblivious transfer and polynomial evaluation based on cryptographic techniques. The scheme ensures that data sources will only know the intersection but nothing else. Although this scheme can answer intersection queries involving two data sources the communication cost required by this technique is $O(n^2)$ where n is the size of the databases. Therefore, it is impractical to apply in large databases. Hacigumus et al. [19], Hore et al. [20] and Aggarwal et al. [2] propose using third parties as database service providers without the need for expensive cryptographic operations. However the proposed schemes do not allow queries to execute over the data of multiple providers, which is the main focus of our work. Aggarwal et al. [3] show the challenges in finding the kth element in the union of more than two databases while preserving privacy and propose an approximate solution. In addition, several high level design efforts and requirement specifications have been made to support the privacy of individual information while still supporting some degree of sharing [1, 5, 6, 7, 10, 27, 28, 12].

Although related, our work is orthogonal or complementary to privacy preserving data management in data mining and information retrieval. In data mining, several efforts have been made to either preserve the privacy of individuals using randomized techniques [8, 9, 16, 25] or to preserve the privacy of the database while running data mining algorithms over multiple databases [13, 21] using cryptographic techniques such as secure multi-party computation and encryption.

2.2 Shamir's Secret Sharing

Shamir's secret sharing method [26] allows a dealer D to distribute a secret value v_s among n peers $\{Q_1, Q_2, ..., Q_n\}$, such that knowledge of any k ($k \leq n$) peers is required to reconstruct the secret. Since, even complete knowledge of $k - 1$ peers cannot reveal any information about the secret, Shamir's method is information theoretically secure. Dealer D chooses a random polynomial

$q(x)$ of degree $k-1$ where the constant term is the secret value, v_s, and a publicly known set of n random points. The dealer computes the share of each peer as $q(x_i)$ and sends it to peer Q_i. The method is summarized in Algorithm 1.

Algorithm 1 Shamir's Secret Sharing Algorithm

```
1: Input:
2: $v_s$: Secret value;
3: $D$: Dealer of secret $v_s$;
4: $Q$: set of peers $Q_1, ..., Q_n$ to distribute secret;
5: Output:
6: $share_1, ..., share_n$: Shares of secret, $v_s$, for each peer $Q_i$;
7: Procedure:
8: $D$ creates a random polynomial $q(x) = a_{k-1}x^{k-1} + ... + a_1x^1 + a_0$ with
   degree $k-1$ and a constant term $a_0 = v_s$.
9: $D$ chooses publicly known $n$ random points, $x_1, ...x_n$, such that $x_i \neq 0$.
10: $D$ computes share, $share_i$, of each peer, $Q_i$, where $share_i = q(x_i)$.
```

In order to construct the secret value v_s, any set of k peers will need to share the information they have received. After finding the polynomial $q(x)$, the secret value $v_s = q(0)$ can be reconstructed. $q(x)$ can be found using Lagrange interpolation such that $q(x_i) = share_i$ where $i = 1, ..., k$. The key observation is that at least k points and shares are required in order to determine a unique polynomial $q(x)$ of degree $k-1$.

3 Problem Formulation

The problem of query processing across multiple private databases is defined as follows:

> Let $D_1, D_2, ..., D_m$ be the databases of a set of m data source peers P, $P = \{P_1, ..., P_m\}$ and q be a query spanning D_1 through D_m. The problem is to compute the answer of q without revealing any additional information to any of the data source peers.

Intersection, join and aggregation operations are particularly challenging from a privacy preserving query processing point of view since to perform these operations, all of the data residing at different sites may need to be accessed to compute an answer. Therefore, we focus on these operations.

Agrawal et al. [4] solved the intersection and equijoin problem by restricting them to two data sources with some relaxation in an honest-but-curious [17] environment. The relaxation reveals the sizes of the tables or lists in the databases to the other party. The method is based on an encryption/decryption protocol, which requires an extensive amount of computation, and cannot be used for more than two data sources.

To reduce the high computational cost due to encryption/ decryption, and to speed up the query processing time, we propose a solution in an honest-but-curious environment for intersection, equijoin and aggregation queries. Our solution is similar to [15] which solves some of these problems (i.e., only sum and average query processing) over only datawarehouses and reveals more information about the underlying datawarehouse. In our scheme, we use third parties located in a peer-to-peer system, namely Chord [29]. Although we use Chord in our system for the selection of the third parties, our method can be adapted to any structured peer-to-peer system.

Similar to [4], we do not aim to solve the problem of revealing additional information to a peer which poses multiple queries and combines their results in order to obtain information about the data. In addition, we do not solve the problem of data discovery and schema mediation. Solutions to these problems are discussed briefly in [4].

4 Query Processing in an Honest-but-curious Environment

In this section, we introduce our query processing technique and explain how it works in an honest-but-curious environment. An *honest-but-curious environment* is one where the parties involved in a query follow the given protocol; additionally they may keep any result or information they obtain during the course of the protocol. This setting is also known as *semi-honest environment* in the literature. In our scheme, queries are answered in a privacy preserving manner using third parties. At the end of query processing, no additional useful information is revealed to any parties involved in query processing but only the query answer is revealed to the query posers.

One possible approach towards solving this problem is to use a one-way cryptographic hash function such as SHA or MD to find the intersection of two lists. In this scheme, data sources hash the secret values in their respective lists and send the hashed lists to a third party. Then, the third party compares these hashed lists and sends the intersection of two hashed lists to each of the data sources so that they can figure out the intersection. Since the hash function is one-way and if the third party is not curious, the third party cannot learn the content of the lists from the hashed list and the data sources cannot learn anything other than the intersection. However, in practice, the number of one-way hash functions are limited, a curious third party may try all of them to extract the content of the hashed lists. Basically, the third party can check whether an element e exists in the incoming hashed list or not, and thus, it might exhaustively try the entire domain to find out the data of the data source (*Dictionary attack*) if the domain size is small. Even if the third party is not curious, only the equality check could be done by the third party using this approach. Therefore, this solution is inadequate to support aggregation queries (e.g. sum of values of specific keys without revealing values).

4.1 Intersection Queries

The intersection query processing problem is stated as follows:

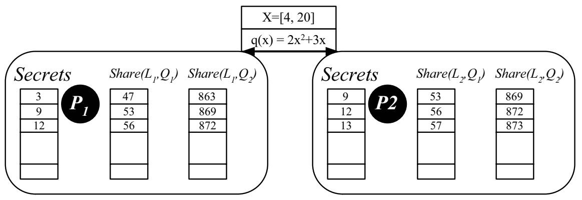

Figure 1. Basic scheme.

Let L_1, L_2,..., L_m be the lists containing secret data stored by a set of data sources $P = \{P_1, P_2, ..., P_m\}$ respectively and a query $q = L_1 \cap L_2 \cap ... \cap L_m$ is posed by P_1. Let $Q = \{Q_1, Q_2, ..., Q_n\}$ be a set of n third parties. The problem is to obtain the answer to q with the help of the third parties in Q without revealing any additional information to any of the third parties in the set Q, and by only providing the query result to the providers in the set P.

Our query processing technique consists of three phases: (1) Distribution phase; (2) Computation at the third parties; and (3) Final computation at the peers.

4.1.1 Distribution Phase

Basic Scheme: After peer P_1 poses the intersection query, P_1 decides on a random polynomial of degree $k-1$, $q(x)$, and chooses n random values $X = \{x_1, ..., x_n\}$ (one for each third party). Then P_1 sends $q(x)$ and X to the other data sources. Each data source P_i has a list of secret elements, $L_i = \{e_1, e_2, ..., e_{|L_i|}\}$; P_i creates n shares $share(L_i, Q_1)$, ... ,$share(L_i, Q_n)$ one for each of the third parties Q_1 through Q_n respectively. P_i creates the shares by applying Shamir's secret sharing algorithm to each of the elements in L_i. For every element e_w in L_i, P_i computes the share of third party Q_j, $sh(e_w, Q_j)$, using Algorithm 1 with $q(x)$ and X (the constant term in $q(x)$ will be replaced by the secret value, e_w, to compute $q(x)$ in Shamir's secret sharing). Therefore, the list of shares of third party Q_j from L_i is $share(L_i, Q_j) = \{sh(e_1, Q_j), ..., sh(e_{|L_i|}, Q_j)\}$. Then P_i sends $share(L_i, Q_j)$ to the third party peer Q_j, which is determined by hashing the value $h(x_j)$ onto the Chord ring. We call this phase the *Distribution Phase*. After this phase, the peers determine the intersection of the sets they received and send the results back as described in the next section.

We illustrate the basic scheme of the distribution phase using an example of two data sources P_1 and P_2 (see Figure 1). Each data source has a list of secrets. The data sources agree on the polynomial $2x^2 + 3x$ and on X values $\{4, 20\}$ in order to distribute their shares to two third parties. For the first element with value 3 in Peer P_1's secret list, it computes the share corresponding to this element to send to Q_1 (i.e. $sh(e_1, Q_1)$) as follows:

$$\begin{aligned} sh(e_1, Q_1) &= q(x_1) + v_1 \\ &= 2 \times 4^2 + 3 \times 4 + 3 \\ &= 47 \end{aligned}$$

The computation is performed similarly for all the secrets using the different X values corresponding to the peers. The resulting shares are shown in Figure 1, which are then sent to the third parties.

Improved Scheme: Although the basic distribution scheme is correct, it has two shortcomings. (1) If the same polynomial is applied on all of the elements in the list, for a curious peer that receives the data, this is equivalent to receiving a list where every element in the list is incremented by the same constant. For example, in Figure 1 the lists received by Q_1 from the different peers are basically the original lists incremented by the constant 44. Consequently, the peer may be able to guess that particular value depending on the data semantics. If the third party can find out the minimum or the maximum data value in the original list it can recover the whole list in its original form. Furthermore, we have to make sure that the same polynomial is applied on the same elements residing at different parties so that they are mapped to the same value. (2) The second shortcoming is that the third party peers used for computation are determined by hashing the x_i values onto the Chord ring. The host choosing the set X may try to influence the randomness in the selection of these values in order to ensure that some particular collaborating peers in Chord get the queries.

To solve the first problem, instead of choosing one polynomial with fixed coefficients, the peers agree on $k-1$ hash functions H_1,..., H_{k-1}, which are used to determine different coefficients for the polynomial used to hide an element v_i in a list. The polynomial for element v_i is constructed as follows:

$$H_{k-1}(v_i)x^{k-1} + H_{k-2}(v_i)x^{k-2} + ... + H_1(v_i)x + v_i$$

This modification in the calculation ensures that the same polynomial is applied on the same elements residing at different data sources, at the same time different polynomials are applied to the different elements of the lists.

In order to address the second problem we should make sure that the data sources collectively create the same sequence of x values, while preventing them from cheating by generating IDs that map to particular peers in the Chord ring. To achieve this, every data source involved in the query generates a vector consisting of n random numbers. The final values of the x values are obtained by adding up these vectors and hashing them through a one way hash function. As a result, all the data sources obtain the same random vector and none of them can influence the result to select particular peers from the Chord ring.

Algorithm 2 Distribution Phase

```
1: Input:
2: $X$: Random Values $X = \{x_1, .., x_n\}$;
3: $H$: $k-1$ hash functions to be used in constructing the polynomial, $H = \{H_1, .., H_{k-1}\}$;
4: $L_i$: Secret list at data source $P_i$;
5: Output:
6: $share(L_i, Q_1), ..., share(L_i, Q_n)$: Shares of secret list, $L_i$, for each peer $Q_j$;
7: Procedure:
8: for Each secret value $v_s \in L_i$ do
9:   Find share $sh(v_s, Q_j)$ of each peer $Q_j$ for $v_s$ with Algorithm 1 using $q(x)$ where $q(x) = H_{k-1}(v_s)x^{k-1} + ... + H_1(v_s)x + v_s$ and add $sh(v_s, Q_j)$ into $share(L_i, Q_j)$.
10: end for
11: Locate $Q_1, .., Q_n$ on the Chord ring by hashing $X = \{x_1, .., x_n\}$ on to the ring and send each $share(L_i, Q_j)$ to $Q_j$ directly.
```

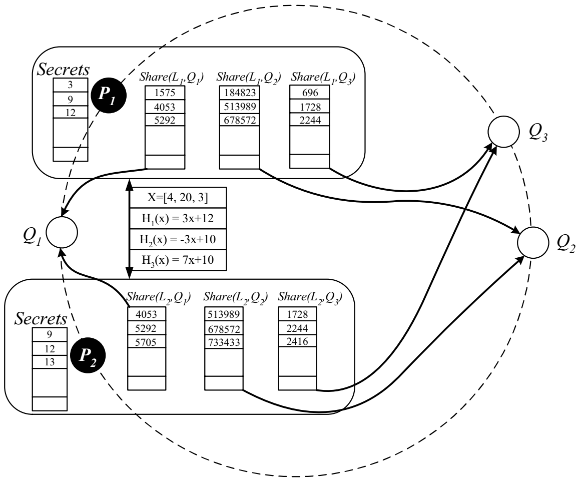

Figure 2. Illustration of distribution phase

The resulting algorithm for the distribution phase is summarized in Algorithm 2 and an example is illustrated in Figure 2 for two data sources P_1 and P_2 with two lists $L_1 = \{3, 9, 12\}$ and $L_2 = \{9, 12, 13\}$. To compute $L_1 \cap L_2$ using three third parties namely Q_1, Q_2 and Q_3, P_1 and P_2 decide on three hash functions, $H_1(x) = 3x + 12$, $H_2(x) = -3x + 10$, $H_3(x) = 7x + 2$. Also three x values are determined in order to randomly select peers from the Chord ring. P_1 chooses the random vector $[3, 8, 4]$ and P_2 chooses the random vector $[1, 12, -1]$; both peers add these values up and produce the values $X = [4, 20, 3]$.

The $share$ of Q_1 for value $v_1 = 3$ that P_1 has, $sh(v_1, Q_1)$,is computed as follows:

$$\begin{aligned} sh(v_1, Q_1) &= H_3(v_1){x_1}^3 + H_2(v_1){x_1}^2 + H_1(v_1)x_1 + v_1 \\ &= (7 \times 3 + 2) \times 4^3 + (-3 \times 3 + 10) \times 4^2 + \\ &\quad (3 \times 3 + 12) \times 4 + 3 \\ &= 1575 \end{aligned}$$

P_1 does the same computation for the other secret values for all the other third parties and sends the list {1575, 4053, 5292} to Q_1, {184823, 513989, 678572} to Q_2, and {696, 1728, 2244} to Q_3. Similarly P_2 computes and sends the list {4053, 5292, 5705} to Q_1 {513989, 678572, 733433} to Q_2, and {1728, 2244, 2416} to Q_3.

4.1.2 Computation Phase

After receiving their shares from the data sources, $P_1, ..., P_m$, every third party, Q_i, calculates bitmaps, $B(L_1, Q_i)$,...,$B(L_m, Q_i)$, corresponding to the lists $share(L_1, Q_i)$,...,$share(L_m, Q_i)$ respectively. These bitmaps are used to indicate which elements of a list are present in the intersection. Q_i traverses the lists that it received and if the jth element of $share(L_g, Q_i)$ is found in all of the lists, then the corresponding entry in the bitmap, $B(L_g, Q_i)[j]$, is set to 1. Formally,

$$B(L_g, Q_i)[a] = \begin{cases} 1 & \text{if } \forall h \exists f \text{ s.t. } share(L_g, Q_i)[a] = share(L_h, Q_i)[f] \\ 0 & \text{otherwise} \end{cases}$$

Then Q_i sends these bitmaps to the corresponding data sources, i.e., $B(L_g, Q_i)$ is sent to P_g. This phase is called the *computation in the third parties* and the computation in Q_i is shown in Algorithm 3 and illustrated in Figure 3 for the example scenario of Figure 2.

Algorithm 3 Computation in the third parties

```
1: Input:
2: $Share_L$: Set of share lists, $Share_L = \{share(L_1, Q_i), .., share(L_m, Q_i)\}$;
3: Output:
4: Set of bitmaps $\{B(L_1, Q_i), .., B(L_m, Q_i)\}$ to send back to the data sources $P = \{P_1, ..., P_m\}$ respectively;
5: Procedure:
6: for each list $share(L_k, Q_i) \in Share_L$ do
7:   for $j = 1; j \leq |share(L_k, Q_i)|$ do
8:     if $share(L_k, Q_i)[j] \in share(L_1, Q_i) \cap ... \cap share(L_m, Q_i)$ then
9:       $B(L_k, Q_i)[j] = 1$
10:    end if
11:  end for
12: end for
13: Send $B(L_1, Q_i), ..., B(L_m, Q_i)$ to $P_1, ..., P_m$ respectively
```

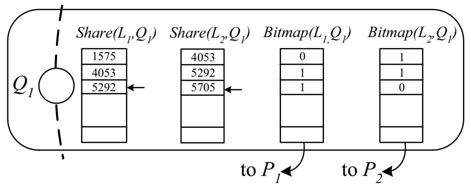

Figure 3. Illustration of computation in third party phase

In Figure 3, after receiving the lists, Q_1 discovers the identical values, (e.g. 4053 and 5292 are shared but the first value 1575 is not), and sends bitmap {0, 1, 1} to P_1, and the bitmap {1, 1, 0} to P_2.

The time complexity of this algorithm is linear with the size of the longest share received, i.e. $O(max(|share_{L_1}|, ..., |share_{L_m}|)$ after sorting. However, for brevity we did not state that in Algorithm 3.

4.1.3 Final Computation Phase

After receiving the bitmaps from all the third parties, each data source P_i determines which elements in its list are present in the result of the intersection query. Formally if the jth element of all bitmaps coming from third parties is set to 1 then, the corresponding element at entry j in the list is added to the answer set. Each peer performs this computation with its list using the bitmaps to find the answer to the query. This phase of final computation and processing in data source P_1 is summarized in Algorithm 4.

Algorithm 4 Final Computation

```
1: Input:
2: $L_1$: List of data source $P_1$
3: $B$: Set of bitmaps coming from the third parties such that $B[i]$ is a bitmap
   coming from peer $Q_i$, $B(L_1, Q_i)$.
4: Output:
5: $Answer$: Intersection of lists involved in the query.
6: Procedure:
7: for $j = 1; j \leq | L_1 |; j++$ do
8:   if $B[i][j] = 1$ for all $i \leq n$ then
9:     add $L_1[j]$ to $Answer$
10:  end if
11: end for
12: return $Answer$
```

In order to decide whether an element is in the intersection or not, all the bitmaps coming from all third party peers must be 1 for that element. This follows from Shamir's secret sharing method and can be stated as:

Lemma 1 *Given a polynomial $q(x)$ of degree $k-1$, a set of n random points X, m secrets $s_1, ..., s_m$, and n shares for n third parties computed by Algorithm 2 for each of secret values. $s_1 = s_2 = ... = s_m$ if the shares coming from $s_1, s_2, ..., s_m$ are equal at all the third parties.*

For the example in Figure 2, P_1 will receive the bitmap $\{0, 1, 1\}$ and P_2 will receive the bitmap $\{0, 1, 1, 0\}$ from all the third parties. Consequently peers P_1 and P_2 are able to determine the result of the intersection query, which is $\{9, 12\}$.

4.2 Equijoin

The equijoin query processing problem is defined as follows:

> Assume P_1 has a table T_1 and P_2 has a table T_2 each of which has a specific attribute A. Then, the goal is to compute $T_1 \bowtie T_2$ such that both parties do not learn any extra information other than the query result. Query poser P_1 will learn only the tuples t such that $t \in P_2$ for which $t.A \in T_1.A$. In other words, P_2 shares a list, l_v, of tuples in T_2 for each value $v \in T_2.A$ with P_1 if $\exists t \in T_1$ such that $t.A = v$, and nothing else.

We also propose to solve the equijoin query processing problem using third parties that are chosen from the P2P system. We describe and solve the problem for two peers, but the scheme can be generalized to n peers in the same way the intersection query is performed. At the end of query processing, third parties will not be revealed any extra useful information. Both the peers, P_1 and P_2, are only provided with the answer to the query.

After peer P_1 poses a query, $q = T_1 \bowtie T_2$, to P_2, P_1 and P_2 decide on $k-1$ hash functions to generate a polynomial of degree $k-1$ for each element in the table; they also agree on n random values $X = \{x_1, .., x_n\}$ as described in Section 4.1.1. Initially P_1 constructs a list L_1 from T_1 consisting of every element e in $T_1.A$. Similarly, P_2 constructs a list L_2 from $T_2.A$. During the first phase P_1 and P_2 find the intersection $L_1 \cap L_2$ using the technique described in Section 4.1; then P_2 sends all tuples $t \in T_2$ where $t.A \in L_1 \cap L_2$ to P_1. Therefore, our technique consists of two phases (1) Intersection phase and (2) Join computation phase. In the intersection phase P_1 and P_2 compute the intersection set of their values and after that P_2 sends all tuples which join with tuples in P_1 during the join computation phase.

During the course of the query processing, no extra useful information gets revealed. First P_1 and P_2 compute the intersection of the two lists. In the join computation phase, P_2 only sends information about the intersection of lists. As a result, nobody gains extra useful information.

It can be argued that P_2 can cheat by sending incorrect values after the intersection is determined, however we do not target to overcome this problem since P_2 has complete control over its own data, and it can change the data as it wishes before or during query processing.

4.3 Aggregation

The traditional aggregation operation is usually used to find the aggregate of a list of values such as SUM, AVERAGE or MIN/ MAX. Therefore, one kind of privacy preserving aggregation can be thought of as computing the aggregate of values in the union of lists coming from different data sources such that data sources will only know the final aggregate but nothing else. To answer traditional aggregation queries, each data source can compute its local aggregate and the final aggregate can be computed such that none of the data sources will know the local aggregate of each other but the final aggregate value (*Secure multipatry computation* [17] or the technique described in this section can be used to compute the final aggregate value). However, data sources may not be willing to execute aggregation operations over their whole data. For example, a company may not be willing to share the expenditure of a customer if he/she is not a customer of the other companies involved in the query. Similarly, consider a scenario where 3 doctors want to collaborate to classify their patients according to the number of total visits per year, therefore they share the number of visits for a specific patient with another doctor if that patient is also in his/her database without revealing their identities. The traditional aggregation operation is not strong enough to support these needs. Therefore, we define a new type of aggregation operation and develop a corresponding privacy preserving aggregation query processing method.

The privacy preserving aggregation query processing problem is defined as follows:

> Let T_1, T_2,..., T_m be the tables stored by a set of source peers $P = \{P_1, P_2, ..., P_m\}$ ($m \geq 3$) respectively containing a key and a value field. The peers would like to learn the aggregation of the values in their databases that have the same key. Let $Q = \{Q_1, Q_2, ..., Q_n\}$ be a set of n third parties. Then the problem is to obtain the answer to the query with the help of the third parties in Q without revealing any additional information to the third party peers in the set Q, and by only providing the aggregation result to the source peers in the set P.

For example, suppose three peers, P_1, P_2 and P_3, are involved in an aggregation SUM query. If P_1 and P_2 have values V_1 and V_2 for a particular key K but P_3 does not have an entry for K, then at the end of the query processing, P_1 and P_2 will learn that sum for K is $V_1 + V_2$ but they will not know that the key K is present in only two of the databases. P_3, on the other hand, will not learn anything at the end of the query processing. Similar to the protocols presented in the previous sections, the third parties will also not be able to derive any useful information, such as K, V_1, V_2 or V_3, from the messages exchanged during query processing.

This type of aggregation queries is specific to the privacy preserving query domain, and has not been proposed before to the best of our knowledge. It is impossible to execute this type of query with traditional cryptographic solutions such as secure multi-party computation without relaxing the privacy requirements. Cryptographic techniques need to allow data sources to know which of the data sources have a specific *Key*, so that they could compute the aggregation using secure multi-party computation for that *Key*. Even with this relaxation, it is impractical to apply this technique over large tables.

Our solution, although privacy preserving does reveal more information if a specific Key exists in only one of the data sources, P_i, then P_i would know it is the only owner of Key (the problem definition implicitly states that none of the data sources would know how many of the data sources have a specific key). If $(Key, Value)$ exits only in P_i and the domain of $Value$ is the nonnegative numbers, then the sum for Key will be $Value$ so that P_i will know it is the only owner of Key. The main observation behind our aggregation query processing is that Shamir's secret sharing method allows one to compute any linear combination of secrets. We make use of this property to perform aggregation queries such as AVERAGE, and SUM queries over multiple private databases. In addition, we propose another technique using Shamir's secret sharing method to support MIN, MAX and MEDIAN queries. In our scheme, the data sources involved in the query will only learn the aggregation query result while third parties will not be able to learn any useful information.

We first outline the general solution and then further discuss each of the aggregate functions separately. After the query is posed, the data sources decide on $k-1$ hash functions to determine the polynomials to apply on the key values, and they also choose n random values $X = \{x_1, ..., x_n\}$ as described in Section 4.1.1. Then, they construct two polynomials named $q(x)$ and $r(x)$ for each $(Key, Value)$ pair to hide Key and $Value$ respectively. For each $(Key, Value)$, the polynomials $q(x)$ and $r(x)$ are constructed as follows for min/max/median queries:

$q(x) = H_{k-1}(Key)x^{k-1} + ... + H_1(Key)x + Key$
$r(x) = H_{k-1}(Key)x^{k-1} + ... + H_1(Key)x + Value$

For sum and average queries, $r(x)$ is a randomly selected polynomial of degree $(k-1)$ selected by each data source independently. Therefore, each data source has its own different $r(x)$. The reason for this is to hide the number of data sources having the same Key which cannot be done using the same $r(x)$. Then, the $(Key, Value)$ share of each third party is calculated using $q(x)$ and $r(x)$ such that the share of the ith party is $(q(x_i), r(x_i))$.

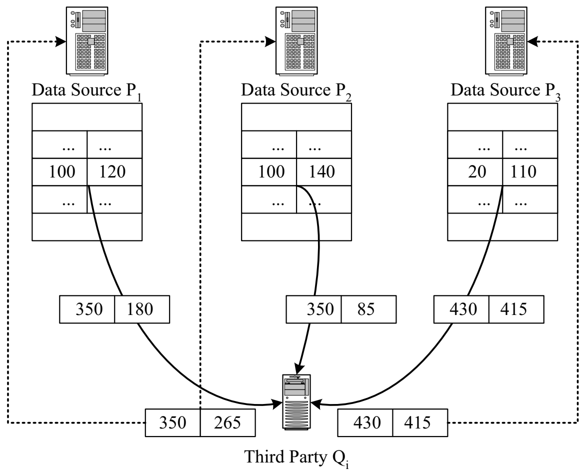

Figure 4. Illustration of Aggregation

Figure 4 shows an example with three data sources, P_1, P_2 and P_3. Data sources compute the shares for the third party Q_i using the polynomials described above. Then each share is sent to Q_i, which is determined by hashing the value $h(x_i)$ onto the Chord ring.

After the distribution phase, the third party peers create two column tables, the first column for Key and the second for the aggregation result (i.e., aggregation of secret values), and they perform the aggregation operation on the values they have received. At the end, they send the aggregation result back to the corresponding data source. As shown in Figure 4, the aggregation for the tuples with the same key is calculated by the third parties and sent back to the corresponding data sources. After this, the data sources compute the actual aggregation result.

During the third step, the data sources in set P receive their results from the n third parties. Since they know the values of X, they can solve the equation set in order to obtain the aggregation query result.

Average/Sum operations: In this set of operations, every single secret that has the same key contributes to the aggregation result. The data source P_j constructs a random polynomial $r(x) = a_j x_i^{k-1} + b_j x_i^{k-2} + ... + Value$ to hide the secret values for each $(Key, Value)$ pair in addition to polynomial $q(x)$, which is used to hide Key. After generating these polynomials, it sends the shares of the third parties by computing the share of Q_i as $(q(x_i), [a_{P_j} x_i^{k-1} + b_{P_j} x_i^{k-2} + ... + v_{P_j}])$ for each secret key-value pair, where $v_{P_j} = Value$ and $q(x) = H_{k-1}(Key)x^{k-1} + ... + H_1(Key)x + Key$. After Q_i receives the shares of m data sources, it sends the sum of values which have the same key. Assume l of the m data sources have the same Key with secret values v_1 through v_l correspondingly. Then the sum for that *Key* is in the following form:

$$\begin{array}{c} a_1 x_i^{k-1} + b_1 x_i^{k-2} ... + v_1 + \\ a_2 x_i^{k-1} + b_2 x_i^{k-2} ... + v_2 + \\ \vdots \\ a_l x_i^{k-1} + b_l x_i^{k-2} ... + v_l \end{array}$$

Therefore Q_i sends its results $RES_i = (a_1 + a_2 + ... + a_l)x_k^{k-1} + ... + SUM$, where SUM is the sum of secret values ($SUM = v_1 + v_2 + ... + v_l$) for the values that have the same key.

Each data source receives n results from each of the third parties:

$$\begin{array}{c} RES_1 = (a_1 + a_2 + ... + a_l)x_1^{k-1} + ... + SUM \\ RES_2 = (a_1 + a_2 + ... + a_l)x_2^{k-1} + ... + SUM \\ \vdots \\ RES_n = (a_1 + a_2 + ... + a_l)x_k^{k-1} + ... + SUM \end{array}$$

Since $X = \{x_1, x_2, .., x_n\}$ is known by the data source, there are a total of k unknown coefficients including SUM and $n \geq k$ equations. Therefore, SUM can be found by using any k of the above equations.

The random polynomial selection used to hide secret values ensures that data sources cannot determine how many data sources have the same key (i.e., l). They can only compute the sum of the coefficients of the sum of the random polynomials without knowing the number of polynomials included in that sum.

For the average query Q_i send $RES_i = [(a_1 + a_2 + ... + a_l)x_k^{k-1} + ... + SUM]/l$ where $RES_i = \frac{(a_1+a_2+...+a_l)}{l} x_k^{k-1} + ... + AVG)$ and $AVG = \frac{v_1+v_2+...+v_l}{l}$. Therefore, each data source receives n results from each of the third parties:

$$\begin{array}{c} RES_1 = \frac{(a_1+a_2+...+a_l)}{l} x_1^{k-1} + ... + AVG \\ RES_2 = \frac{(a_1+a_2+...+a_l)}{l} x_2^{k-1} + ... + AVG \\ \vdots \\ RES_n = \frac{(a_1+a_2+...+a_l)}{l} x_k^{k-1} + ... + AVG \end{array}$$

Again since $X = \{x_1, x_2, .., x_n\}$ is known by the data sources, there are k unknown coefficients including AVG and $n \geq k$ equations. Therefore, AVG can be found by using any k of the above equations. Again data sources cannot determine how many of the data sources have the same key (i.e., k).

Min/Max/Median operations: For these operations, only one of the secret values is sought. The order of the secret values needs to remain the same in the shares of the third parties to be able to use third parties to compute Min/Max/Median. To hide the secret value of a $(Key, Value)$ pair, data source P_j uses the following polynomial: $H_{k-1}(Key)x^{k-1} + ... + H_1(Key)x + v_j$, where v_j is its secret $Value$.

The key observation for computing the answer to this set of operations is that if $(Key, V_1), (Key, V_2)$,..,(Key, V_n) are the $(Key,\ Value)$ pairs at data sources $P_1, P_2, .., P_n$ respectively and $V_1 < V_2 < ... < V_n$, the shares of a third party, $sh_1, sh_2, .., sh_n$, coming from $V_1, V_2,\ ..,\ V_n$ respectively preserve the order ($sh_1 < sh_2 <...< sh_n$). This result follows from the way shares are computed . The shares, sh_i and sh_j for two secret values, v_i and v_j, are $sh_i = H_{k-1}(Key)x^{k-1} +... +H_1(Key)x + v_i$ and $sh_j = H_{k-1}(Key)x^{k-1} +... +H_1(Key)x + v_j$. Therefore, if $v_i \leq v_j$ then $sh_i \leq sh_j$, i.e., the calculation is order preserving.

Depending on the query, a third party Q_i returns the minimum/maximum/median of its shares for Key as the answer to the query. Without loss of generality, we can assume that the query asks for minimum. Each of the third parties receives $(Key, Value)$ pairs from m data sources. After that, they return the minimum value for the pairs that have the same key.

After the third parties send in the results back, each data source receives the results from all third parties associated with Key, and determines the value of the minimum by using the result of a third party Q_i, RES_i. The result of the third party Q_i is in the following form:

$$RES_i = H_{k-1}(Key)x_i^{k-1} + ... + H_1(Key)x_i + MIN$$

Since the data sources know all the functions, $H_1, .., H_{k-1}$, Key and the value x_i they can determine the minimum, MIN, from the above equation.

One could argue that the difference between values are revealed to the third party with this scheme. Although this information is useless, this information leakage could be prevented. All data sources could divide their $Value$s by the same random number so that the third party could not know the exact difference between values (since it does not know the random number).

5 Query Response Time Analysis

The cost of query processing in an honest-but-curious environment for intersection queries is the sum of the cost of the distribution phase, the cost of the computation in third parties phase, and the cost of the final computation phase.

Let $L_1, ..., L_m$ be the lists of the m data sources involved in the query. Without loss of generality, we can assume that the list L_1 has the maximum number of elements. Therefore, the computation in the distribution phase requires $O(|L_1|)$ time. The computation in third parties also requires $O(|L_1|)$ comparisons. Finally the cost of the computation in the final phase is also $O(|L_1|)$ (these comparisons are for checking if the bitmap entry is 1 or not). Therefore, the total computation needed is $O(|L_1|)$ with approximately a total amount of $3 \times |L_1|$ operations being performed which will require $\frac{3 \times |L_1|}{CPU\ Speed}$ seconds.

Communication contributes to most of the total time spent during the execution of the protocol, the shares are first sent to the third parties and then the bitmaps are received from them. The cost of sending shares is $\frac{n \times (|L_1| + ... + |L_m|) \times b}{Bandwidth}$, where n is the number of third parties and b denote the size of a share coming from a secret value. The cost of collecting the bitmaps is $\frac{n \times (|L_1| + ... + |L_m|)}{Bandwidth}$. Therefore, the total communication cost is:

$$\frac{n \times (b + 1) \times (|L_1| + ... + |L_m|)}{Bandwidth}$$

Hence the query response time for intersection queries:

$$\approx \frac{3 \times |L_1|}{CPU\ Speed} + \frac{n\,(b + 1)\,(|L_1| + ... + |L_m|)}{Bandwidth}$$

For the equijoin operation, first the peers need to find the intersection and then they need to transfer the matching elements. Let v denote the size of the data associated with each element. In the worst case the intersection may contain all the elements, in which case the query response time is:

$$\approx \frac{3 \times |L_1|}{CPU\ Speed} + \frac{n\,(b + 1 + v)\,(|L_1| + ... + |L_m|)}{Bandwidth}$$

The cost of the aggregation operation is the same as the intersection operation, with additional computation cost that is necessary to solve the linear system of the received values to determine the random polynomials used by each host.

6 Conclusion

In this paper, we proposed a solution for processing queries across private databases while preserving the data providers' privacy. Our scheme is able to compute the answers to queries without violating the privacy requirements of the data sources in an honest but curious environment. Our approach uses third parties without revealing any information to these parties. These third parties are selected from a P2P system, Chord. The secrets are distributed to the third parties using Shamir's secret sharing method on every element. Using Shamir's secret sharing also enables us to perform aggregation queries in addition to the intersection and equijoin queries, where we introduced a novel type of aggregation queries needed in privacy preserving data sharing. The scheme we propose is practical for executing queries over large databases as opposed to prior work in this field. The analytical evaluation demonstrates that our approach incurs significantly less communication and computation costs when compared with cryptographic approaches.

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu. Enabling privacy for the paranoids. In *Proc. of the 30th Int'l Conference on Very Large Databases VLDB*, pages 708–719, Aug 2004.

[2] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *CIDR*, pages 186–199, 2005.

[3] G. Aggarwal, N. Mishra, and B. Pinkas. Privacy-preserving computation of the k'th-ranked element. In *Proc. of IACR Eurocrypt*, pages 40–55, 2004.

[4] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proc. of the 2003 ACM SIGMOD international conference on on Management of data*, pages 86–97, 2003.

[5] R. Agrawal, P. J. Haas, and J. Kiernan. A system for watermarking relational databases. In *Proc. of the 2003 ACM SIGMOD international conference on Management of data*, pages 674–674. ACM Press, 2003.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *28th Int'l Conf. on Very Large Databases (VLDB), Hong Kong*, Aug 2002.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Implementing p3p using database technology. In *In Proc. of the 19th Int'l Conference on Data Engineering, Bangalore, India*, Mar 2003.

[8] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450. ACM Press, 2000.

[9] S. Agrawal and J. R. Haritsa. A framework for high-accuracy privacy-preserving mining. In *to appear in International Conference on Data Engineering 2005*, 2005.

[10] M. Bawa, R. Bayardo Jr., and R. Agrawal. Privacy preserving indexing of documents on the network. In *Proc. of the 29th Int'l Conference on Very Large Databases VLDB*, pages 922–933, Aug 2003.

[11] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration of heterogeneous information sources. *Data Knowl. Eng.*, 36(3):215–249, 2001.

[12] E. Bertino, B. C. Ooi, Y. Yang, and R. H. Deng. Privacy and ownership preserving of outsourced medical data. In *ICDE*, 2005.

[13] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, 2002.

[14] U. Dayal and H. Hwang. View definition and generalization for database integration in a multidatabase system. In *IEEE Transactions on Software Engineering*, volume 10, pages 628–644, 1984.

[15] F. Emekci, D. Agrawal, and A. E. Abbadi. Abacus: A distributed middleware for privacy preserving data sharing across private data warehouses. In *ACM/IFIP/USENIX 6th International Middleware Conference*, 2005.

[16] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–228. ACM Press, 2002.

[17] O. Goldreich. Secure multi-party computation. *Working Draft*, jun 2001.

[18] L. M. Haas, R. J. Miller, B. Niswonger, M. T. Roth, P. M. Schwarz, and E. L. Wimmers. Transforming heterogeneous data with database middleware: Beyond integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.

[19] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In *SIGMOD Conference*, 2002.

[20] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proc. of the 30th Int'l Conference on Very Large Databases VLDB*, pages 720–731, 2004.

[21] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proc. of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 36–54. Springer-Verlag, 2000.

[22] M. W. N. Jefferies, C. Mitchell. A proposed architecture for trusted third party services. *Cryptography Policy and Algorithms Conference*, July 1995.

[23] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *ACM Symposium on Theory of Computing*, pages 590–599, 2001.

[24] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254. ACM Press, 1999.

[25] S. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proc. of the 28th Int'l Conference on Very Large Databases*, pages 682–693, Aug 2002.

[26] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[27] R. Sion, M. Atallah, and S. Prabhakar. Rights protection for relational data. In *Proc. of the 2003 ACM SIGMOD international conference on Management of data*, pages 98–109. ACM Press, 2003.

[28] R. Sion, M. J. Atallah, and S. Prabhakar. Resilient rights protection for sensor streams. In *Proc. of the 30th Int'l Conference on Very Large Databases VLDB*, pages 876–887, 2004.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM*, pages 149–160, 2001.