Running Head: COMPUTING JOIN AGGREGATES

Computing Join Aggregates over Private Tables

Rong She[*], Ke Wang[*], Ada Waichee Fu[+], Yabo Xu[*]

[*] School of Computing Science, Simon Fraser University, Canada

{rshe, wangk, yxu}@cs.sfu.ca

[+] Dept. of Computer Science & Engineering, Chinese University of Hong Kong

adafu@cse.cuhk.edu.hk

Abstract

We propose a privacy-preserving protocol for computing aggregation queries over the join of *private* tables. In this problem, several parties wish to share aggregated information over the join of their tables, but want to conceal the details that generate such information. The join operation presents a challenge to privacy preservation because it requires matching individual records from private tables. We solve this problem by using a novel private sketching protocol that securely exchanges some randomized summary information about private tables. This protocol (1) conceals individual private values and their distributions from all non-owning parties, (2) works on many general forms of aggregation functions, (3) handles group-by aggregates, and (4) handles roll-up/drill-down operations. Previous works have not provided this level of privacy for such queries.

Introduction

With explosive growth in data storage and networking, collaboration among various data holders has become more and more important. Such collaboration frequently involves multiple private sources where free-access to private data is difficult or impossible due to privacy concerns. Therefore, techniques that address the need of privacy-preserving collaborative data analysis are critical (Clifton et al., 2004).

For example, a hospital maintains the medical histories of patients, denoted as table H, and a research institute maintains the records of patients' DNA samples, denoted as R. Both tables contain the attribute "Patient_Name" that is not necessarily a key attribute if a patient has several diseases or DNA samples. To establish the relationships between diseases and DNA anomalies, the research institute issues the following query that returns the aggregated number of occurrences for each combination of disease and DNA characteristics:

```
SELECT      H.Disease, R.DNA_Characteristics, COUNT(*)
FROM        H, R
WHERE       H.Patient_Name = R.Patient_Name
GROUP BY    H.Disease, R.DNA_Characteristics
```

Such aggregate queries are fundamental for most statistical and trend analysis. For example, for a certain disease "d" and DNA characteristic "c", in the joined table, if the number of occurrences for <d,c> combination is 70, and the number of occurrences for <c> alone is 100, then it can be deduced that a patient with DNA characteristic "c" has 70% chance of having disease "d". Association and correlation analysis (Agrawal, Imielinski & Swami, 1993) is based on this idea.

On the other hand, due to privacy restrictions such as the HIPAA policies (http://www.hhs.gov/ocr/hipaa/), neither the hospital nor the research institute is willing to disclose its patient-specific information (such as patient names) to the other party. They wish to share the high level aggregated count, but want to conceal the low level details such as who contributed to the count. This is an example of *private join aggregate queries* that we will consider in this paper.

Private Join Aggregate Queries

Join aggregate queries. A join aggregate query has the following form:

SELECT	group-by-list, *agg*(*exp*)
FROM	$T_1, \ldots, T_n$
WHERE	$A_1=A_1'$ and … and $A_m=A_m'$
GROUP BY	group-by-list

where each T_i is a table; A_i and A_i' are *join attributes* from different tables; group-by-list is a list of *group-by attributes* possibly from different tables; *exp* is an arithmetic expression over *aggregation attributes* possibly from different tables; *agg* is an aggregation function. In this paper, we consider the aggregation functions COUNT and SUM. *Conceptually*, to answer such query, the tables in FROM are joined according to the predicates in WHERE, and the joined records are then grouped on the group-by-list. The query result contains one row for each group with *agg*(*exp*) being computed over the group. If the optional GROUP BY is missing, there is only one group and one row in the query result. We can assume that a table contains only join attributes and the attributes that occur in SELECT (including aggregation attributes and group-by attributes); all other attributes are invisible to the query and thus are safe to be ignored for our purpose.

Private join aggregate queries. In a private join aggregate query, each T_i is a *private table* owned by a different party. These parties wish to compute and share the result of a join aggregate query, but no party should learn any information from other parties beyond the query result. The group-by attributes and their values are contained in the query result, thus they are not considered private. Our goal is to conceal all other private attribute values and their distributions. In particular, among other attributes, we want to protect F(*v*) from any non-owning party, where F(*v*) is the occurrence count of a join value *v* in a private table. This level of privacy has not been provided by previous solutions, which will be discussed in more details in the next section.

We assume the *honest-but-curious* behavior (Goldreich, 2001): The parties follow the protocol properly with the exception that they may keep track of all intermediate computations and received messages, and analyze the messages to try to learn additional information. Our focus is on privacy protection in the computation process, not against the answers to queries. The latter has been studied in statistical databases (Adam & Wortman, 1989). Our assumption is that, when the participating parties wish to share the query result, information inferred from such

results is a fair game (Agrawal, Evfimievski & Srikant, 2003). Protection against query results and background knowledge in general requires additional mechanisms and is beyond the scope of this paper.

Challenges and Contributions

We present a novel solution to private join aggregate queries based on the *sketching* technique previously studied for join size estimation and aggregation for data streams (Alon, Gibbons, Matias & Szegedy, 1999; Dobra, Garofalakis, Gehrke & Rastogi, 2002; Ganguly, Garofalskis & Rastogi, 2004). However, none of these works involve privacy issues. The basic idea of sketching is that each table maintains a summary structure, called *atomic sketch,* which is later combined to produce an estimator of the query result. More details on sketching are given in a later section. A striking property of atomic sketches is that they are computed locally. Each party can compute its own atomic sketches without any knowledge of data in other paties. This localization property of atomic sketches inspired us to develop a solution for private join aggregate queries by making use of similar sketching techniques.

However, it is not straightforward to adapt the sketching techniques in our privacy-preservation scenario. We will show that a straightforward way of combining atomic sketches, when producing the query estimator, would allow a party to learn the distribution of join values owned by other parties. In the patient example, this means that the hospital or research institute could learn some patient names owned by the other party by examining atomic sketches. We will present a detailed analysis on the source of such privacy leakage and the types of information that need to be concealed in order to prevent such leakage. We then propose a *private sketching protocol* for computing the private join aggregate query. The key idea is that each party holds one "random share" of the same atomic sketch so that collectively they represent the atomic sketch, but individually they are useless. We show how the query can be estimated from such random shares in such a way that no party learns private information (both individual values and distribution of values) from other parties.

We then extend this protocol to multi-party queries, with general arithmetic *exp* and group-by operator, and to support the roll-up/drill-down operations (Gray, Bosworth, Layman & Pirahesh, 1996). Typically, an interesting group-by-list is unknown in advance and must be discovered simultaneously as interesting aggregates are searched. With the roll-up and drill-down operations, the aggregates at a coarser level (with fewer group-by attributes) can be

computed efficiently from the aggregates at a finer level (with more group-by attributes). However, supporting roll-up/drill-down operations at the same level of privacy protection as for basic queries requires an innovative work.

Related Work

Several models have been proposed for general secure computations. In the trusted third party model (Jefferies, Mitchell & Walker, 1995), all parties give the data to a "trusted" third party and have the third party do the computation. Such a third party has to be *completely* trusted and is difficult (often impossible) to find. In the secure multi-party model (Yao, 1986), given two parties with inputs x and y, the goal is to compute a function f(x,y) such that the two parties learn only f(x,y) and nothing else. In theory, any multi-party computation can be solved by building a combinatorial circuit, and simulating that circuit. However, the communication costs are impractical for data intensive problems such as the one considered here.

In Du & Atallah (2001), privacy-preserving cooperative statistical analysis was studied for vertically or horizontally partitioned data. With vertically partitioned data, n records $\{(x_1,y_1),\ldots, (x_n,y_n)\}$ are distributed at two parties such that Alice holds $\{x_1,\ldots,x_n\}$ and Bob holds $\{y_1,\ldots,y_n\}$. In this case, the join relationship is one-to-one and is implicit by the sequential ordering of records. A similar data partition based on a common key identifier is assumed in Du & Zhan (2002) and Vaidya & Clifton (2002). In real world, it is odd that the data owned by two mutually untrusted parties are about exactly same set of entities. In Agrawal, Srikant & Thomas (2005), horizontally partitioned data is considered where each party possesses some records from the same underlying table and the aggregation is over a specified range of attribute values. We consider general join relationships specified by a SQL statement, which cannot be implied by the sequential ordering of records. For example, the patient example has the many-to-many join relationship. Another example is the foreign-key based join. In these cases we have to protect the private join attributes.

Another recent work (Emekci, Agrawal, Abbadi & Gulbeden, 2006) also discussed aggregations such as SUM queries over several private databases; however, the problem it considers is different. It assumes all parties contain the same pair of attributes: a "key" and a "value" field. The SUM query is to aggregate the values from all parties

that correspond to the same key. In contrast, we deal with private tables with different schemas and general join relationships, where a SUM query is defined over the joined table.

The closest work to ours is Agrawal et al. (2003) that studied the private equi-join size problem. It proposed a scheme for encrypting join values but required exchanging the frequency of encrypted join values. As pointed out in Agrawal et al. (2003), if the frequency of some join values is unique, the mapping of the encryption can be discovered by matching the frequency before and after the encryption. Thus the privacy of join values may be compromised. Our solution does not have such problem.

Another related problem is the restriction-based inference control in OLAP queries (Wang, Jajodia & Wijesekera, 2004; Zhang, Zhao & Chen, 2004). The goal of inference control is to prevent values of sensitive data from being inferred through answers to OLAP queries. These works mainly dealt with the privacy breaches that arise from the answers to multiple queries; they do not consider the privacy breaches during query processing. The inference control problem has been studied largely in statistical databases, for example, see Adam & Wortman (1989).

Preliminaries

First, we give a brief review on two basic techniques that are the building blocks of our solution: the sketching technique and the private scalar product protocol.

Basic Sketching Technique

Sketching is a randomized algorithm that estimates the join aggregate result with an unbiased random variable, called *sketch*. The sketch is obtained by multiplying some *atomic sketches*, which are computed at each table. The expected value of the sketch is shown to be equal to the aggregate result with bounded variance (Alon et al. 1999; Dobra et al. 2002).

As an example, consider the query SUM(A) over 3 tables T_1, T_2 and T_3, with join conditions $T_1.J_1=T_2.J_2$ and $T_2.J_3=T_3.J_4$, where A is an *aggregation attribute* in T_1. The table containing A (in this case, T_1) will be called the *aggregation table*. Each pair of the join attributes (J_1, J_2) or (J_3, J_4) is called a *join pair*. J_2 and J_3 may be the same

attribute in T_2, but conceptually they belong to different join pairs and are treated separately. Let D_i denote the domain of J_i. Each join pair shares the same domain. For simplicity, we assume $D_i=\{1,\ldots,|D_i|\}$. For any table T_i, let JS_i be the set of join attributes in T_i. Suppose JS_i contains *m* join attributes, then a *value instance V* on JS_i is a set that contains one value for each of the *m* attributes, i.e. $V=\{x_1,\ldots,x_m\}$ where each x_i is a value of a distinct join attribute in JS_i. Let $T_i(V)$ be the set of records in T_i having the value instance *V* on JS_i. For an aggregation table T_i, we define $S_i(V)$ to be the sum of aggregation attribute values over all records in $T_i(V)$; for any non-aggregation table T_i, we define $F_i(V)$ to be the number of records in $T_i(V)$. Thus, T_1 will have $S_1(V)$ defined; T_2 and T_3 will have $F_2(V)$ and $F_3(V)$ defined. The sketch is constructed as follows:

- **The *ε* family**: For each join pair (J_i, J_j), select a family of 4-wise independent binary random variables $\{\varepsilon_k, k=1,\ldots,|D_i|\}$, with each $\varepsilon_k \in \{1,-1\}$. That is, each join value k is associated with a variable ε_k whose value is randomly selected from {1,-1} and any 4 tuple of such ε variables is jointly independent. The set of values for all ε_k variables is called a *ε family*. In this example, there are two independent ε families, one for each join pair. In table T_i, with a join value instance *V*, for each join value x in *V* (x is a value of some join attribute J), there is one ε_x variable from J's ε family. Let $E_i(V)= \Pi_{x \in V} \varepsilon_x$.
- **Atomic sketches**: There is one atomic sketch for each table. For the aggregation table T_1, its atomic sketch is $X_1=\Sigma_V[S_1(V)\times E_1(V)]$, i.e. the sum of $S_1(V)\times E_1(V)$ over all distinct *V* in T_1, called *S-atomic sketch* (S for summary); the atomic sketches for T_2 and T_3 are $X_2= \Sigma_V[F_2(V)\times E_2(V)]$ and $X_3=\Sigma_V[F_3(V)\times E_3(V)]$, called *F-atomic sketches* (F for frequency). Hence there is exactly one atomic sketch for each table, either a S-atomic sketch or a F-atomic sketch, depending on whether the table contains aggregation attribute or not.
- **The sketch:** The sketch is defined as $\Pi_i (X_i)$, the multiplication of atomic sketches over all tables. The expected value of the sketch can be shown to be equal to SUM(A) with bounded variance. We refer interested readers to Alon et al. (1999) and Dobra et al. (2002) for details.

Because sketch is a random variable, the above computation must be repeated many times to get a good average. Alon, Matias & Szegedy (1996) suggests a procedure of *boosting* where the number of trials is α×β. For every α trials, the average of their sketches is computed, resulting in β averages. The final estimator is the median of these β averages. Note the ε families are chosen independently in each trial. We will refer to this process as *αβ-boosting*. The time complexity of sketching with αβ-boosting is $O(\alpha\times\beta\times\Sigma_i|T_i|)$, where $|T_i|$ denotes the number of records in table T_i. Experiments from previous works and our experiences show it is usually accurate (error rate < 5%) with

moderate size of α (~50) and β (~5).

Sketching is suitable when the join size is larger than a certain "sanity bound" (Alon et al. 1999; Ganguly et al. 2004). This condition typically holds for aggregation queries in data mining applications where data volume is large.

Private Shared Scalar Products

The *private scalar product* protocol was first discussed in Du & Atallah (2001)0. Given two d-dimensional vectors $\vec{U}$ =<U_1,…,U_d> and $\vec{V}$ =<V_1,…,V_d> owned by two honest-but-curious parties, this protocol computes $\vec{U} \times \vec{V} = \sum_{i=1..d} U_i \times V_i$, such that the two parties obtain no additional knowledge other than $\vec{U} \times \vec{V}$. In other words, the inputs $\vec{U}$ and $\vec{V}$ are concealed from the non-owning parties.

In some applications, a scalar product $\vec{U} \times \vec{V}$ is needed as a part of the entire computation, but the value of $\vec{U} \times \vec{V}$ needs to be concealed in the process from all parties. Such problems can be addressed by the *private shared scalar product* (SSP) protocol (Goethals, Laur, Lipmaa & Mielikainen, 2004), or *SSP protocol* subsequently. With this protocol, each party obtains one random share of $\vec{U} \times \vec{V}$, denoted as R_1 and R_2, such that $R_1+R_2=\vec{U} \times \vec{V}$, but (R_1+R_2) is concealed from both parties. R_1 and R_2 are *complementary* in that they are random shares of the same scalar product $\vec{U} \times \vec{V}$ and only knowing both can infer the result of the scalar product, thus the value of $\vec{U} \times \vec{V}$ is unknown to both parties. Note that a random share can be positive or negative and their range can be the real domain which has nothing to do with the value of $\vec{U} \times \vec{V}$, thus it is impossible to guess $\vec{U} \times \vec{V}$ from any single share. Several two-party SSP protocols with linear complexity have been proposed (Clifton, Kantarcioglu, Vaidya, Lin & Zhu, 2002; Du & Atallah, 2001; Du & Zhan, 2002). The SSP protocol for multi-parties was studied in Goethals et al. (2004). In the rest of this paper, we use SSP($\vec{V}_1$,…, $\vec{V}_k$) to denote the SSP protocol on input vectors $\vec{V}_1$,…, $\vec{V}_k$.

Method

We start with analyzing the privacy breaches in the standard sketching process. The purpose is to derive the

requirements on the types of information that must be concealed in the query computation. We then show how to conceal such information using our protocol. For illustration purposes, we consider the basic join aggregate COUNT(*) over two private tables (i.e., computing the join size of two tables). The analysis can be extended to general join aggregates over multiple tables in a similar way.

Assume that Alice holds table T_1 and Bob holds T_2 with one common join attribute J. Let D_1 be the set of distinct join values in T_1 and D_2 be the set of join values in T_2. Thus, J's active domain $D=D_1 \cup D_2$. First, a ε family of binary random variables for J is selected. Both parties use this same ε family to compute their atomic sketches. For simplicity, assume that D has two values, v_1 and v_2 (thus $|D|=2$); there are two variables $\{\varepsilon_1, \varepsilon_2\}$ in the ε family. Let $F_i(v)$ denote the number of records having the join value v in table T_i, i.e. $F_i(v)$ values represent the distribution of join values. $F_1(v)$ belongs to Alice and should be concealed from Bob; $F_2(v)$ belongs to Bob and should be concealed from Alice.

The two parties compute their atomic sketches X_i as follows:

Alice: $X_1 = F_1(v_1) \times \varepsilon_1 + F_1(v_2) \times \varepsilon_2$, (1)

Bob: $X_2 = F_2(v_1) \times \varepsilon_1 + F_2(v_2) \times \varepsilon_2$. (2)

So far, the computation of X_1 and X_2 is done locally, using the shared ε family and locally owned $F_i(v_j)$ values ($\{F_1(v_1), F_1(v_2)\}$ for Alice and $\{F_2(v_1),F_2(v_2)\}$ for Bob). Because the ε family is just some random value, knowing it will not lead to any private information about the other party.

Next, the sketch $X_1 \times X_2$ needs to be computed. Unfortunately the standard sketching technique will disclose information about $F_i(v)$ in this computation process, and thus fail on privacy protection. To see this, suppose that Alice sends her atomic sketch X_1 to Bob. Now, Bob knows X_1, his own X_2, and the ε family. In Equation (1) and (2), with only $F_1(v_1)$ and $F_1(v_2)$ being unknown, Bob can infer some knowledge about $F_1(v_i)$ using Equation (1). For example, knowing $\varepsilon_1=1$ and $\varepsilon_2=-1$, if X_1 is positive, Bob knows that v_1 is more frequent than v_2 by a margin of X_1 in Alice's table. The problem may be less obvious when there are more than two values in D, however, it still leaks non-zero knowledge about $F_1(v_j)$ values. Furthermore, in the αβ-boosting process, the above computation is repeated $\alpha \times \beta$ times and there will be one pair of Equation (1) and (2) for *each* of the $\alpha \times \beta$ trials. With the ε family being independently chosen in each trial, each pair of these equations provides a new constraint on the unknown $F_1(v_j)$ values. If the number of trials is equal to or greater than $|D|$, Bob will have a sufficient number of Equation (1)

to solve all $F_1(v_j)$ values owned by Alice. Therefore, even for a large domain D, the privacy breach is severe if Bob knows both atomic sketches.

On the other hand, even if Bob knows only his own X_2, given the result of $X_1 \times X_2$, he can easily get X_1. Now, if the individual sketch $X_1 \times X_2$ in each trial is also concealed from Bob, because both parties agree to share the final result which is an average of $X_1 \times X_2$, by comparing the final result with his own X_2's, Bob may still infer some approximate knowledge on X_1, allowing him to find the approximate values of $F_1(v_j)$ as described above. The situation is symmetric with Alice. Therefore, to prevent any inference on other party's $F_i(v_j)$, *all atomic sketches should be unknown to all parties, which implies that the ε families should also be concealed from all parties*.

Suppose that all of atomic sketches X_i and ε families are concealed from all parties. If the sketch $X_1 \times X_2$ is known to Bob, Bob may still succeed in inferring X_1 in some extreme cases. For example, knowing $X_1 \times X_2 = 0$, and $F_2(v_1) = 10$ and $F_2(v_2) = 5$, since $X_2 \neq 0$ for any value of ε_1 and ε_2, Bob can infer that $X_1 = 0$. Additionally, from Equation (1), $X_1 = 0$ holds only if $F_1(v_1) = F_1(v_2)$ (because ε_1 and ε_2 are from $\{1,-1\}$). Consequently, Bob learns that the two join values are equally frequent in Alice's table. Therefore, to prevent the above inference, *the individual sketch $X_1 \times X_2$ in each trial should be concealed from all parties*.

Therefore, the only non-local information that a party is allowed to know is the final query result which will be shared at the end. Because the final result is something that has been averaged over many independent trials, disclosing one average will not let any party infer the individual sketches or underlying atomic sketches. Note that with the current problem definition where parties agree to share the final result, we cannot do better than this.

Since the ε families must be concealed from Alice and Bob, we need to use a *semi-trusted* third party (Kantarcioglu & Vaidya, 2002), called Tim, to generate the ε families. To fulfill its job, Tim must also be an honest-but-curious party who does not collude with Alice or Bob. In real world, finding such a third party is much easier than finding a trusted third party. Now we have three parties: Alice, Bob and Tim. The protocol needs to make sure that Tim cannot learn private information about Alice or Bob or the final query result, i.e. Tim knows nothing about atomic sketches, sketches or $F_i(v_j)$ values. The only thing Tim knows is the ε families which are just some random variables. On the other hand, Alice owns $F_1(v_j)$ and Bob owns $F_2(v_j)$, both should know nothing about the other party's $F_i(v_j)$, the ε families, atomic sketches or individual sketches. These requirements are summarized in the following definition.

Definition 1. Information Concealing (IC-conforming) Sketching Protocol. Let Y denote an average of sketches over α trials in the αβ-boosting. There will be β number of such Y's in total. A sketching protocol satisfying the following requirements is called *Information Concealing* sketching protocol, or we say that the sketching protocol is *IC-conforming*: Alice learns only Y and local $F_1(v_i)$; Bob learns only Y and local $F_2(v_i)$; Tim learns only the ε families; atomic sketches X_i and individual sketches $X_1 \times X_2$ are concealed from all parties. ■

Theorem 1. The computations in an IC-conforming sketching protocol conceals $F_i(v)$ from all non-owning parties throughout the computation process. ■

Proof: First, Tim knows only the ε families and nothing about $F_i(v)$. Consider Alice and Bob. From the IC-conformity, Alice knows only Y and $F_1(v_i)$ and Bob knows only Y and $F_2(v_i)$, where Y is the average of sketches over α trials. $F_1(v_i)$ and $F_2(v_i)$ does not go beyond the information derived from the local table. An average Y of sketches $X_1 \times X_2$ over α trials, where $\alpha \geq 2$, provides no clue on an individual sketch $X_1 \times X_2$ and whether $X_1 \times X_2$ is equal to 0 because each sketch is a signed quantity and is computed with a ε family that is independently and randomly chosen in each trial. Even if there is a non-zero chance that Tim chooses the same ε family in all α trials, therefore Y is equal to each individual sketch, Alice and Bob will never know that this actually happens because the ε families are unknown to them.

The only additional knowledge gained by Alice and Bob is the value of Y, which may approximate the query result $F_1(v_1) \times F_2(v_1) + \ldots + F_1(v_k) \times F_2(v_k)$. However, this approximation alone does not allow any party to solve the other party's $F_i(v)$ because there are many solutions for the unknown $F_i(v)$. It does not help to use different averages Y in the αβ-boosting because they are instances of a random variable and do not act as independent constraints. ■

Basic IC-Conforming Sketching Protocol for Two-Party COUNT(*) Queries

We now present a IC-conforming protocol for two-party COUNT(*) queries. More general queries will be considered in the next section.

Let Bob be the querying party. The overall computation with αβ-boosting is as follows:

```
1. for i=1 to β do
2.     for j=1 to α do
```

3. ε-phase;
4. S-phase;
5. α-phase;
6. β-phase;

The α×β trials are divided into β groups with each group containing α trials. Each trial has the *ε-phase* and the *S-phase* (line 3 and 4). The ε-phase generates the ε family and the S-phase computes atomic sketches. For each group, the *α-phase* computes the average of the sketches in the α trials (line 5). Finally the *β-phase* finds the median of the β averages (line 6), which is the query estimator. Below, we explain each phase.

ε-phase. In this phase, Tim generates the ε family. Let D_1 be the set of join values in T_1 (Alice's table) and D_2 be the set of join values in T_2 (Bob's table). $D=D_1 \cup D_2$. To generate the ε family, Tim needs to know $|D|$, $|D_1|$ and $|D_2|$, and the correspondence between the ε variables and the join values. Alice and Bob can hash their join values by a *cryptographic hash function* (Stinson, 2006) such that Tim does not need to know the actual join values in D. A cryptographic hash function H has the following properties: (1) *pre-image resistant*: given a hash value H(v), it is computationally infeasible to find v; (2) *collision resistant*: it is computationally infeasible to find two different inputs v_1 and v_2 with $H(v_1)=H(v_2)$. In real life, industrial-strength cryptographic hash functions with these properties are available (NIST, 2002). The ε-phase is given as follows. Alice and Bob create two hashed sets, S_1 and S_2, of their join values using a shared cryptographic hash function H, and send the hashed sets to Tim separately (line 1 and 2). Tim then generates a ε family $\vec{E}_1$ for Alice and another ε family $\vec{E}_2$ for Bob.

1. Alice and Bob compute locally the hashed sets $S_1 = \{H(v) \mid v \in D_1\}$ and $S_2 = \{H(v) \mid v \in D_2\}$ where H is concealed from Tim.
2. Alice and Bob send S_1 and S_2 to Tim.
3. Tim computes $S=S_1 \cup S_2$.
4. Tim assigns a unique ε variable to each value in S, generating a ε family $\vec{E}_1$ from S_1 and a ε family $\vec{E}_2$ from S_2.

Security analysis. Alice and Bob do not receive information from any party. With the cryptographic hash function H, Tim will not be able to learn original join values from the hashed sets. Since Tim does not know H, it is impossible for Tim to infer whether a join value exists in T_1 or T_2 by enumerating all possible values. This scenario

is different from Agrawal et al. (2003) where all parties know the hash functions. What Tim does learn is the domain size like $|D_1|$, $|D_2|$, $|D_1 \cup D_2|$ and $|D_1 \cap D_2|$. But such general knowledge will not help Tim to infer atomic sketches or the sketches. Therefore, this phase is IC-conforming.

S-phase. In this phase, we compute the atomic sketches X_1 for T_1 (or for Alice) and X_2 for T_2 (or for Bob). For i=1,2, let $\vec{F}_i$ denote the vector of $F_i(v)$ values for all distinct v from D_i, arranged in the same order as in $\vec{E}_i$. X_i is given by the scalar product $\vec{F}_i \times \vec{E}_i$, where $\vec{F}_i$ is owned by T_i and $\vec{E}_i$ is owned by Tim. To conceal $\vec{E}_i$, $\vec{F}_i$ and X_i, Alice and Tim compute $\vec{F}_1 \times \vec{E}_1$ using the SSP protocol; similarly, Bob and Tim compute $\vec{F}_2 \times \vec{E}_2$ using the SSP protocol. This is shown as follows.

1. Alice and Tim compute SSP($\vec{E}_1, \vec{F}_1$). Alice obtains RA and Tim obtains TA, where RA+TA=X_1, the atomic sketch for Alice.
2. Bob and Tim compute SSP($\vec{E}_2, \vec{F}_2$). Bob obtains RB and Tim obtains TB, where RB+TB=X_2, the atomic sketch for Bob.

At the end of this phase, Alice obtains one random share RA, Bob obtains one random share RB, Tim obtains two random shares TA and TB, such that RA+TA=X_1 and RB+TB=X_2, where X_1 and X_2 are the atomic sketches for Alice and Bob.

Security analysis. The SSP protocol ensures that $\vec{F}_i$, $\vec{E}_i$, and X_i are concealed as required. Tim obtains two non-complementary random shares from *different* atomic sketches, which are not useful to infer any atomic sketch. Thus, this phase is IC-conforming.

α-phase. In this phase, we compute the average of the sketches for every α trials. In the jth trial, the sketch is given by $X_{1j} \times X_{2j}$, where X_{1j} and X_{2j} are the atomic sketches for T_1 (Alice) and T_2 (Bob). However, at the end of S-phase, no party obtains X_{1j} or X_{2j}; rather, Tim obtains TA_j and TB_j, Alice obtains RA_j and Bob obtains RB_j, such that $X_{1j}=TA_j+RA_j$ and $X_{2j}=TB_j+RB_j$. In addition, the IC-conformity requires concealing the individual sketches (i.e. $X_{1j} \times X_{2j}$) from all parties. Our approach is to compute the average directly from these random shares while preventing any party from obtaining both complementary random shares.

After α trials, let $\overrightarrow{RA}$ be the vector of $<RA_1,\ldots,RA_\alpha>$ and let $\overrightarrow{RB}$, $\overrightarrow{TA}$, $\overrightarrow{TB}$ be defined analogously. Alice

owns $\overrightarrow{RA}$, Bob owns $\overrightarrow{RB}$ and Tim owns $\overrightarrow{TA}$ and $\overrightarrow{TB}$. The average over the α trials is:

$$Y = \frac{\sum_{j=1}^{\alpha}(X_{1j} \times X_{2j})}{\alpha} = \frac{\sum_{j=1}^{\alpha}[(RA_j + TA_j) \times (RB_j + TB_j)]}{\alpha}$$
$$= \frac{\sum_{j=1}^{\alpha}(RA_j \times RB_j + TA_j \times RB_j + RA_j \times TB_j + TA_j \times TB_j)}{\alpha}$$
$$= \frac{\overrightarrow{RA} \times \overrightarrow{RB} + \overrightarrow{TA} \times \overrightarrow{RB} + \overrightarrow{RA} \times \overrightarrow{TB} + \overrightarrow{TA} \times \overrightarrow{TB}}{\alpha}$$

The numerator is the sum of several scalar products of the random share vectors possibly owned by different parties. To compute these scalar products, if we allow the input vectors to be exchanged among parties, a party obtaining both complementary random shares immediately learns the atomic sketch, thereby violating the IC-conformity. To conceal the input vectors, we use the SSP protocol again to compute these scalar products as follows.

1. Alice and Bob compute SSP($\overrightarrow{RA}$, $\overrightarrow{RB}$);
2. Tim and Bob compute SSP($\overrightarrow{TA}$, $\overrightarrow{RB}$);
3. Tim and Alice compute SSP($\overrightarrow{TB}$, $\overrightarrow{RA}$);
4. Tim computes $\overrightarrow{TA} \times \overrightarrow{TB}$ (no SSP is needed);
5. Tim sums all his random shares and $\overrightarrow{TA} \times \overrightarrow{TB}$, send the result to Alice;
6. Alice adds all her random shares to the sum from Tim, forwards the result to Bob;
7. Bob adds all his random shares to the sum from Alice, divides it by α.

On line 1-3, each party obtains two non-complementary random shares. Then, these random shares and $\overrightarrow{TA} \times \overrightarrow{TB}$ are summed in a forwarding chain (line 5-7), starting from Tim and ending at the querying party Bob. In the end, Bob has the average Y over the α trials.

Security analysis. The SSP protocols ensure that $\overrightarrow{RA}$, $\overrightarrow{RB}$, $\overrightarrow{TA}$, $\overrightarrow{TB}$ are concealed from a non-owning party; therefore, no party learns atomic sketches. After SSP computations, a party may obtain several non-complementary random shares. For example, Alice obtains one random share of $\overrightarrow{RA} \times \overrightarrow{RB}$ and one random share of $\overrightarrow{TB} \times \overrightarrow{RA}$, which together does not help her learn anything. A party may receive a partial sum during sum forwarding. However, each partial sum always contains two or more non-complementary random shares. It is not possible for

the receiver to deduce individual contributing random shares from such a sum. Therefore, this phase is IC-conforming.

β-phase. Repeating the α-phase β times would yield the averages $Y_1, \ldots, Y_\beta$ at Bob. In the β-phase, Bob finds the median of $Y_1, \ldots, Y_\beta$, which is the estimator of the query result.

Security analysis. This phase is done entirely by Bob alone and there is no information exchange at all. Thus the level of privacy at all parties is unchanged.

Since all phases of our sketching protocol are IC-conforming, we conclude that our protocol is a private sketching protocol that is IC-conforming.

IC-Conforming Sketching Protocol for Multi-Party COUNT(*) Queries

Consider n parties $P_1, \ldots, P_n$, where each party P_i holds one private table T_i. Let JS_i be the set of join attributes in T_i. Let $v_i=(x_1,\ldots,x_k)$ be a vector of join values on JS_i in T_i such that each x_i is from a distinct join attribute in JS_i. $F_i(v_i)$ denotes the frequency count of v_i in T_i and $E_i(v_i) = \prod_{x \in v_i} \varepsilon_x$. With each distinct v_i on JS_i corresponding to one dimension, $\vec{F}_i$ and $\vec{E}_i$ denote the vectors of $F_i(v_i)$ and $E_i(v_i)$. The atomic sketch for T_i is given by $\vec{F}_i \times \vec{E}_i$. The sketch, i.e. the multiplication of the atomic sketches for all T_i, is then given by:

$$\sum_{(v_1,\ldots,v_n)} Q(v_1,\cdots,v_n) \qquad (3)$$

where $Q(v_1,\cdots,v_n) = F_1(v_1)\times\cdots\times F_n(v_n)\times E_1(v_1)\times\cdots\times E_n(v_n)$ and $(v_1,\ldots,v_n)$ represents all possible combinations of $v_1,\ldots,v_n$ values.

We can show that the expected value of expression (3) equals to COUNT(*), which is the query result. Let $\sigma(v_1,\ldots,v_n)$ denote the function of whether $(v_1,\ldots,v_n)$ satisfies the equi-join conditions, i.e., $\sigma(v_1,\ldots,v_n)$=true if and only if $(v_1,\ldots,v_n)$ belongs to a tuple in the joined table. The expression (3) can be separated into two parts: $\sum_{\sigma(v_1,\ldots,v_n)=true} Q(v_1,\cdots,v_n)$, and $\sum_{\sigma(v_1,\ldots,v_n)=false} Q(v_1,\cdots,v_n)$.

Similar to the proof in Alon et al. (1999), the first part is equal to COUNT(*) because $E_1(v_1)\times\cdots\times E_n(v_n)=1$ for

$\sigma(v_1,\ldots,v_n)$=true, and the expected value of the second part is 0 since $E_1(v_1)\times\cdots\times E_n(v_n)$ has the expected value of 0 for $\sigma(v_1,\ldots,v_n)$=false. This shows that expression (3) is an unbiased estimator of the query result. Therefore it can be used to estimate the query. Below, we extend our IC-conforming protocol to compute this estimator in a privacy preserving manner.

The ε-phase is similarly done as in the two-party computations, except that $\vec{F}_i$ is owned by P_i and $\vec{E}_i$ is owned by Tim. In the S-phase, we use SSP($\vec{F}_i, \vec{E}_i$) to compute the atomic sketch $\vec{F}_i \times \vec{E}_i$. At the jth trial, P_i obtains the random share R_{ij} and Tim obtains the random share RT_{ij} such that $\vec{F}_i \times \vec{E}_i = R_{ij}+RT_{ij}$. The average of the sketches over α trials is:

$$Y = \frac{\sum_{j=1}^{\alpha}[(R_{1j} + RT_{1j})\times\cdots\times(R_{nj} + RT_{nj})]}{\alpha}$$

The numerator is the sum of 2^n scalar products of the form: $\vec{W}_1 \cdots\cdot \vec{W}_n$, where $\vec{W}_i$ is either $\vec{R}_i = (R_{i1},...,R_{i\alpha})$ owned by P_i, or $\overrightarrow{RT}_i = (RT_{i1},...,RT_{i\alpha})$ owned by Tim. In the α-phase, we can use the SSP protocol to securely compute each of these 2^n scalar products, except that we need to use the n-party SSP protocol such as the one in Goethals et al. (2004). The steps afterwards are the same as in the two-party case, and the security analysis can be similarly done and the same level of security is achieved.

Cost Analysis

Running time. Let $|T_i|$ be the number of records in table T_i. Let $|C_i|$ be the number of distinct vectors v on JS_i in T_i. Let |D| be the total number of distinct join values (on single join attributes) from all tables. Let C_H denote the computation cost of one hash operation. Let $C_{SP}(d)$ denote the computation cost and S(d) denote the communication cost for executing the SSP protocol on d-dimensional vectors. For example, for the two-party SSP protocol in Du & Zhan (2002), S(d) is 4d and $C_{SP}(d)$ is O(d). The total running time of our n-party IC-conforming sketching protocol (with αβ-boosting) is as follows. Note such running time is the total time for computations in all parties.

- *ε-phase*. Hashing and generating the ε families takes $O(C_H\times|D|+\Sigma_i|T_i|+\alpha\times\beta\times|D|)$ time. Note that the hashing is done only once for all trials, but the ε family is generated independently in each trial.

- *S-phase*. Computing atomic sketches takes $O(\alpha\times\beta\times\Sigma_i C_{SP}(|C_i|))$ time.
- *α-phase*. Computing the β averages takes $O(\beta\times 2^n\times C_{SP}(\alpha))$ time as each average involves 2^n calls to the SSP protocol, where n is the number of parties. Note that the factor 2^n usually does not impose an efficiency problem because n is typically very small, say 2-3. In our problem setting, each party refers to an individual or organization that owns some private tables. In most applications, a join aggregate query joins only a small number of tables. This is different from a large number of "record owners" where each owner owns only one record.
- *β-phase*. finding the median of β averages takes $O(n\times\beta)$ time.

Space requirements. The storage space for keeping $\vec{F}_i$ and $\vec{E}_i$ is proportional to the dimensionality of these vectors, which is $|C_i|$ for party P_i. This space is reused in every trial. The space required in the α-phase is $O(\alpha)$ because α is now the dimensionality of the input vectors to the SSP protocols. The space required in the β-phase is $O(\beta)$.

Communication cost. The communication cost is $n\times|D|$ for the n parties sending hashed sets to Tim, $\alpha\times\beta\times\Sigma_i S(|C_i|)$ for computing atomic sketches, $\beta\times 2^n\times S(\alpha)$ for computing all β averages.

Extensions

Join Aggregates on General Attribute Expressions

SUM(A) queries. So far, we have presented the protocol for computing the COUNT(*) query defined on the joined table. Note such protocol can be easily adapted to SUM(A) query on a single aggregation attribute A, since sketch techniques have been shown applicable to SUM(A) in a very similar way, as discussed in the third section where the preliminaries of sketching techniques are discussed. The only difference is that the table with the aggregation attribute A will maintain a S-atomic sketch instead of a F-atomic sketch. Such change only influences

the local computations of the atomic sketch in that table. All later computations that are based on atomic sketches are the same.

SUM($A_1 \times \ldots \times A_m$) queries. For join aggregates of the form SUM($A_1 \times \ldots \times A_m$), where A_1 through A_m are aggregation attributes from different tables, the simple approach of creating a new attribute to store $A_1 \times \ldots \times A_m$ requires joining all tables, therefore, violates our security claims. For this case, our private sketching protocol can be used where the sketch is still computed by multiplication of atomic sketches, only that every table containing an aggregation attribute will contribute a S-atomic sketch, and every other table will contribute a F-atomic sketch. It can be shown that such sketch estimator converges to SUM($A_1 \times \ldots \times A_m$), by following a similar analysis as in the previous section where multi-party COUNT(*) queries are analyzed.

SUM($A_1 \pm \ldots \pm A_m$) queries. Now let us consider a join aggregate of the form SUM($A_1 \theta \ldots \theta A_m$), where each θ is either "+" or "–". We can rewrite this query into SUM(A_1)θ…θSUM(A_m) such that each SUM(A_i) is a sub-query over the original join. For each table containing an aggregation attribute A_i, we now construct one F-atomic sketch and one S-atomic sketch; only one ε family needs to be maintained for both atomic sketches. For all other tables, we construct only the F-atomic sketch. The sketch for each sub-query SUM(A_i) is computed using the S-atomic sketch for the table containing A_i and the F-atomic sketches for all other tables, as described above in the paragraph for the basic type of SUM(A) queries. Then the estimator for the entire SUM($A_1 \theta \ldots \theta A_m$) query can be obtained by summing the estimators for all SUM(A_i) sub-queries.

One subtlety, however, is that each party should learn only the final estimator of SUM($A_1 \theta \ldots \theta A_m$), not the estimator of an intermediate subquery SUM(A_i). In order to achieve this, we need to conceal the estimator of SUM(A_i) from all parties. Let us consider two parties Alice and Bob. The ε-phase and the S-phase can be shared by all sub-queries because these phases produce only random shares. At the end of the S-phase, for each sub-query SUM(A_i), Alice and Bob each obtain one random share, and Tim obtains two random shares that are complementary to Alice's and Bob's share, respectively. The subsequent α-phase and β-phase are separately computed for each sub-query, as explained below.

In the α-phase, each party computes SSPs for each sub-query SUM(A_i). Additionally, to conceal the estimator of SUM(A_i), Tim generates a secret random number R_i. R_i is generated independently for each sub-query but stays the same in each repeated α-phase for the same sub-query. We modify the α-phase as follows:

1-4 are the same as in the α-phase of the basic IC-conforming sketching protocol as discussed in the previous section.

5. Tim generates a random number R_i. This is done once for SUM(A_i).
6. Tim sums all his random shares and R_i, and send the result to Alice.
7. Alice adds all her random shares to the sum and forwards it to Bob.
8. Bob adds all his random shares to the sum, divide the sum by α.

Essentially, we use R_i to disguise the average over the α trials for SUM(A_i) in this modified α-phase.

The β-phase for SUM(A_i) then computes the median of these averages. Let TS_i denote the true estimator of SUM(A_i). At the end of the β-phase for SUM(A_i), the result obtained by Bob is not TS_i, but rather a modified estimator that is equal to ($TS_i + R_i/\alpha$). Since R_i is unknown to Bob, Bob cannot deduce TS_i.

With m sub-queries, all following the same modified α-phases, Tim generates m secret random numbers {R_1, …, R_m} and Bob obtains m modified estimators {S_1, …, S_m}. After all sub-queries are done, Tim computes $(R_1\theta\ldots\theta R_m)/\alpha$, where each operator θ follows the exactly same order as in SUM($A_1\theta\ldots\theta A_m$), and sends the result to Bob, who can then compute the estimator of SUM($A_1\theta\ldots\theta A_m$) as $(S_1\theta\ldots\theta S_m)-(R_1\theta\ldots\theta R_m)/\alpha$. Since each R_i is an independently-generated random number and is only known to Tim, $(R_1\theta\ldots\theta R_m)/\alpha$ provides no clue to Bob on the value of individual R_i. Therefore, the true estimator of each SUM(A_i) is concealed from Bob (as well as from Alice).

General expressions. In general, in the aggregate function *agg*(*exp*) of the join aggregate query, the attribute expression *exp* can be any arithmetic of aggregation attributes, as long as there is no division with the denominator that involves addition or subtraction of attributes from different tables. For example, SUM($A_1+A_2\times A_3-1/A_4$) can be separated into sub-queries SUM(A_1), SUM($A_2\times A_3$), SUM($1/A_4$), and the final aggregate is the sum of all sub-queries and can be computed using modified α-phases as described above. Note that SUM($1/A_4$) can be computed by SUM(A_5), where the new attribute A_5 is equal to $1/A_4$. The only case where our private sketching protocol cannot be applied, is when the denominator involves additions or subtractions of attributes from different tables,

such as SUM(A_1/(A_2+A_3)), where A_2 and A_3 are from different tables. Such aggregations cannot be estimated using sketches because the join relationship needs to be maintained at the record level.

Group-by Operations

In the presence of the group-by operator, the join can be considered as being partitioned on the group-by attributes. Each table with a group-by attribute consists of disjoint *partitions* corresponding to the group-by attribute values in the table. Each table without group-by attributes consists of only one partition, i.e., the table itself. Let p(T,c) denote the partition on a table T that satisfies a condition c. In Figure 1, T_1 consists of two partitions: p(T_1,G_1=a1) and p(T_1,G_1=a2); T_3 consists of one partition p(T_3,*) because T_3 has no group-by attribute, where * denotes the trivial "true". All join values are omitted.

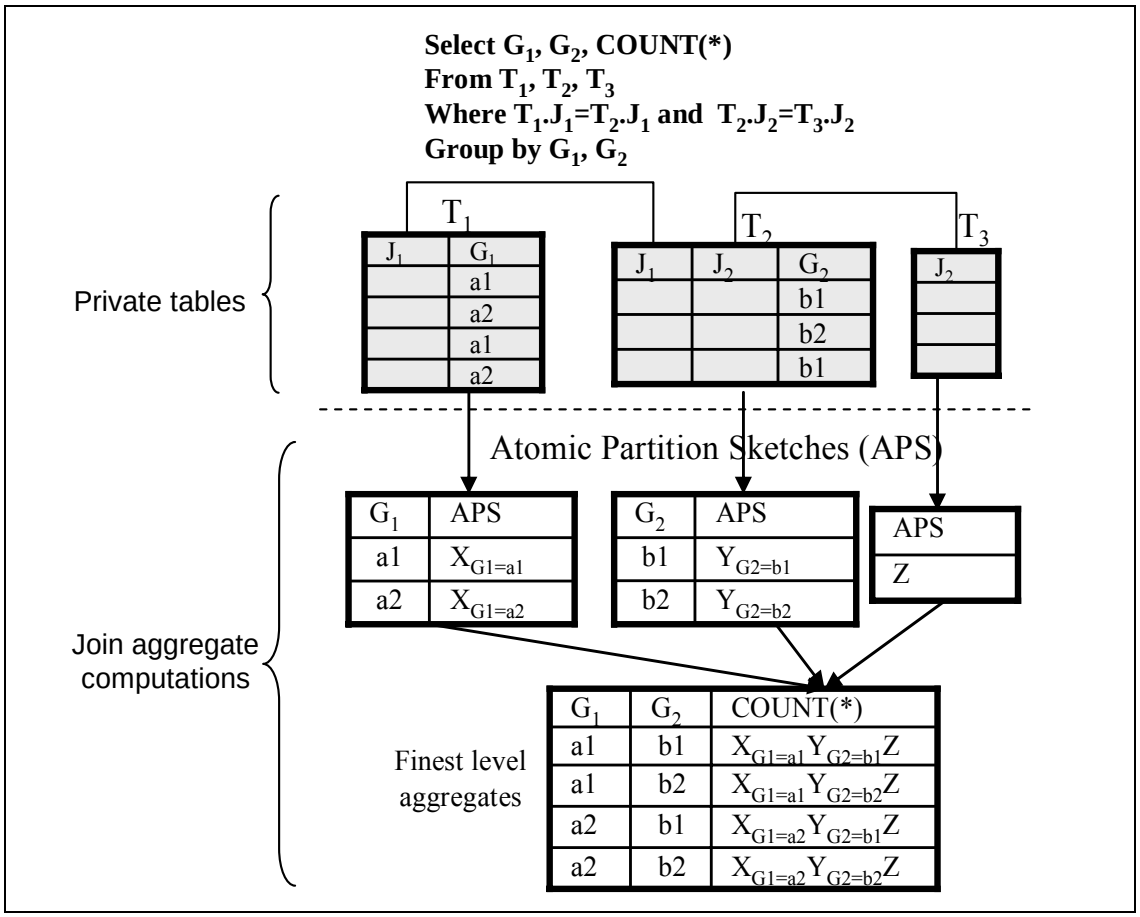

Figure 1. Join aggregates with group-by

The query can be computed by applying the protocol discussed earlier to each combination of partitions from different tables, called a *partition combination*. Each partition combination produces one row in the result. In Figure 1, the protocol can be applied to each of the following partition combinations:

$p(T_1,G_1=a1)$, $p(T_2,G_2=b1)$ and $p(T_3,*)$

$p(T_1,G_1=a1)$, $p(T_2,G_2=b2)$ and $p(T_3,*)$

$p(T_1,G_1=a2)$, $p(T_2,G_2=b1)$ and $p(T_3,*)$

$p(T_1,G_1=a2)$, $p(T_2,G_2=b2)$ and $p(T_3,*)$

The only difference is that the *same* hash function and ε family must be used for all partitions of the same table. This means that the same atomic sketch for a partition, called *atomic partition sketch (APS),* will be used for all partition combinations that contain this partition. For example, with $X_{G1=a1,}$ $X_{G2=b1,}$ $X_{G2=b2,}$ Z being the APSs for $p(T_1,G_1=a1)$, $p(T_2,G_2=b1)$, $p(T_2,G_2=b2)$ and $p(T_3,*)$, respectively, COUNT(*) for the first combination is computed by $X_{G1=a1}X_{G2=b1}Z$, and COUNT(*) for the second combination is computed by $X_{G1=a1}X_{G2=b2}Z$.

There is one sketch for every partition combination and each party obtains one random share of every sketch. Since the random shares at a single party are for different partition combinations, they are non-complementary and do not infer anything. Thus the protocol extended to the group-by operator remains IC-conforming. In the case that all attributes of some table are in the group-by list, the full detail of the table may be leaked in the query result. But this leakage is through the query result, not through computations. A party can always reject such queries or impose a minimum size on the partitions that participate in a query.

Roll-up/Drill-down Operations

To support roll-up/drill-down operations, in a straightforward implementation, the aggregates at the finest level are first computed and subsequently are used for roll-up/drill-down. This means that the parties that maintain such aggregates will know the most detailed aggregates. For preserving the privacy, however, aggregates should be known to a party only at the requested level and only if the participating parties honor the request. In order to achieve this protection, for each partition at the finest level, its owning party maintains one random share of the

APS for the partition, whereas the third party Tim maintains the other random share. No single party can learn the APS from only one of the two complementary random shares.

When a roll-up is requested on some group-by attribute G, the party P that owns G sums up the local random shares for all partitions that differ only on the dimension of G, so does Tim. For each "rolled-up" partition p, the sum at party P and the sum at Tim are the two complementary random shares of the APS for the partition p. The subsequent α-phase will use these random shares in the computation that involves p. In αβ-boosting, each trial maintains a separate set of random shares.

As an example, in the previous Figure 1, suppose that the roll-up is requested by some party P on the group-by attribute G_1, i.e., the roll-up is along G_2 from the finest aggregates on $<G_1,G_2>$. Essentially, it requires computing the sketches:

$X_{G1=a1}(Y_{G2=b1}+Y_{G2=b2})Z$ for G_1=a1

$X_{G1=a2}(Y_{G2=b1}+Y_{G2=b2})Z$ for G_1=a2

where $Y=Y_{G2=b1}+Y_{G2=b2}$ is the APS for the rolled-up partition $p(T_2,*)$. Before the roll-up, the party P_2 has the random shares of $Y_{G2=b1}$ and $Y_{G2=b2,}$ denoted as R_1 and R_2; Tim has the other random shares of $Y_{G2=b1}$ and $Y_{G2=b2}$, denoted as R_1' and R_2'. Note that $Y_{G2=b1}=R_1+R_1$' and $Y_{G2=b2}=R_2+R_2$'. Then, P_2 obtains one random share of Y by R_1+R_2 and Tim obtains the other random share of Y by R_1'+R_2'.

Thus, by maintaining only one set of random shares for the APSs at the finest level, the random shares at a coarser level can be computed incrementally and locally at each party. Essentially, the roll-up/drill-down is now performed on local random shares, not on sketches that involve all parties. The major benefit is that the random shares at a single party provide no clue on the APSs or sketches, as required by the IC-conformity. Only by an honored roll-up/drill-down request, do all parties collaborate to compute the sketches from local random shares (as in the α-phase and β-phase). Therefore, our protocol supports the roll-up/drill-down operations at the same level of privacy protection as specified by the IC-conformity.

It is true that both group-by operations and roll-up/drill operations add more complexity and cost to the basic protocol. But some of this added cost is due to the increase of query power itself even without the privacy consideration.

Experiments

We implemented the basic two-party join size estimation protocol on three PCs that simulate Alice, Bob and Tim, all connected through a LAN with 100Mbps network cables. All PCs have Pentium IV 2.4GHz CPU and 512M RAM and Windows XP. The cryptographic hash function was implemented using the QuickHash library 3.0 ("The QuickHash Library", online source) and the two-party SSP protocol in Du & Zhan (2002) was used.

Tests were done on synthetic datasets with various table sizes and join characteristics: Alice and Bob each own a private table T_1 and T_2 with one join attribute J. The cardinality of J, denoted as |D|, varies from 100 to 10,000. The number of records in T_1, $|T_1|$, was varied from 10,000 to 1,000,000 with the values of J generated following the zipf distribution. T_2 was generated such that for every distinct join value in T_1, a certain B number (B varies from 1 to 10) of records are generated in T_2 with the same join value. Thus, the size of table T_2 is $B*|D|$ and the join size $|T_1 \infty T_2|$ is $B*|T_1|$.

We evaluated our algorithm in terms of both accuracy and efficiency. Since sketching is a randomized algorithm, each reported result is an average over 5 runs, where each run uses exactly the same parameters.

Accuracy

We define the error rate as |F–E|/E, where F is the approximated count computed by our algorithm and E is the true count. In each run, the query result was computed by the αβ-boosting where the number of trials is α×β. In general, a larger number of trials will produce a smaller error rate.

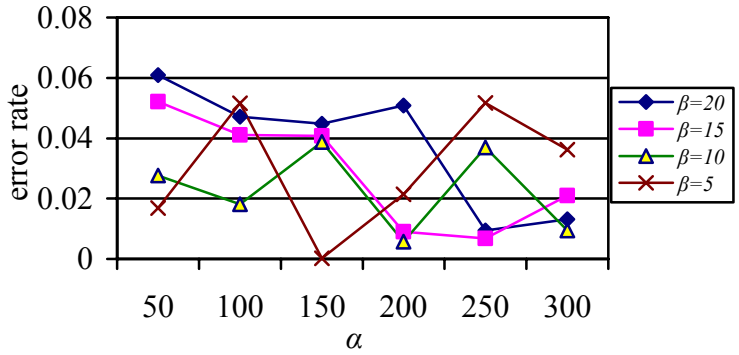

Figure 2. The effect of α and β on error rate

Figure 2 shows the error rate vs. different choices of α and β in the case of $|T_1|$=1,000,000, |D|=100, B=1. The results in other cases show similar trends. Generally, when β is small (<15), the error rate shows random fluctuations with different choices of α, due to the fact that the sketch is constructed on random variables and does not necessarily converge when β is small. As β gets bigger, the error rate tends to be decreased with increasing α, as expected. Note that because sketching is a randomized algorithm, there are always random variations observed; nevertheless, bigger β values generally show smoother trends. We observed that with α ranging from 50 to 300 and β ranging from 5 to 20, the error rate in all runs is no more than 6%. For large α and β, the error is frequently less than 2%. This shows that the approximation provided by sketching is in general sufficient for most data mining applications where the focus is on trends and patterns, instead of exact counts.

With α=200 and β=15, Figure 3 shows the error rate for different $|T_1|$ and B while fixing |D|=100. In general, the error rate is low (<5%). There is no clear trend observed with regard to either $|T_1|$ or B. Figure 4 shows the error rate for different |D| and B while fixing $|T_1|$=10,000. Again, the error rate fluctuates in a small range, i.e., 2%. We believe such randomness is a natural behavior of the sketching algorithm, irrelevant with any other parameter.

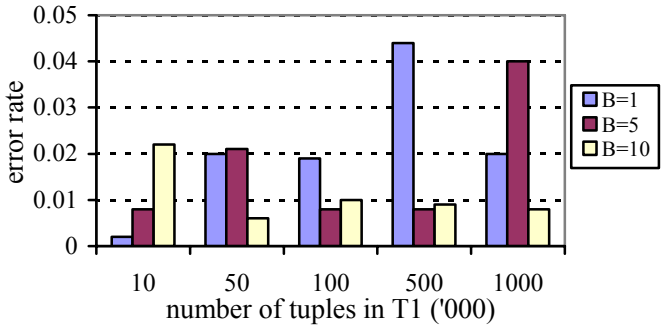

Figure 3. Error rate w.r.t. $|T_1|$ and B (|D|=100)

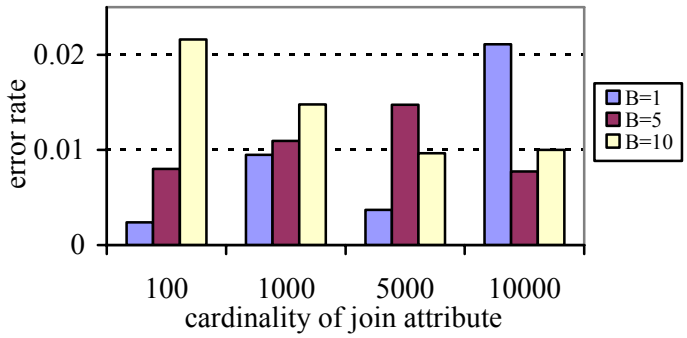

Figure 4. Error rate w.r.t. |D| and B ($|T_1|$=10k)

Efficiency

Our protocol has a linear complexity in terms of the table size, because (1) both the basic sketching algorithm and the SSP protocol have a linear complexity and (2) these procedures are called only a constant number of times. As a representing case, we report the results on efficiency for the case of α=200 and β=15. With other α and β settings, the efficiency analysis stays the same.

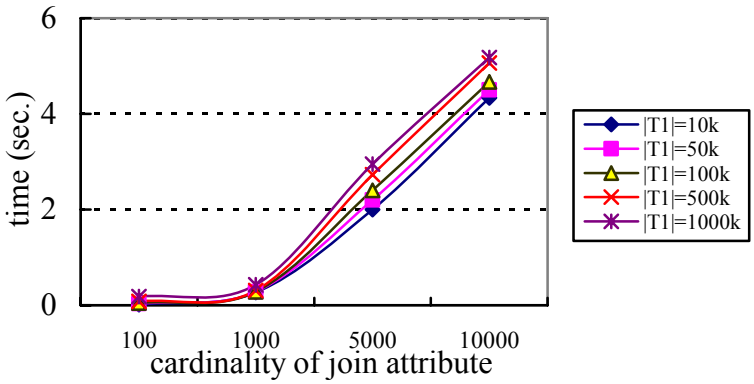

Figure 5. ε-phase running time (B=5)

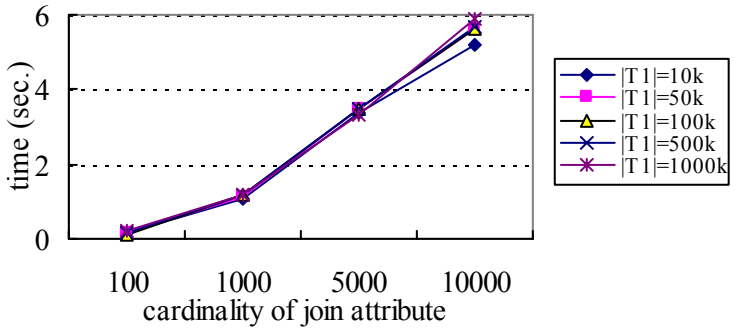

Figure 6. S-phase running time (B=5)

Figure 5 shows the running time of the ε-phase with regard to $|T_1|$ and $|D|$. This time increases linearly with increasing $|T_1|$ and $|D|$. Figure 6 gives the running time of the S-phase. It shows a linear increase with regard to the join cardinality and does not change with $|T_i|$. In all experiments, the running time of α-phase and β-phase is almost ignorable as there are only two tables involved. In general, with more than two tables, the running time will increase, but typically the number of tables, which is the number of parties involved in a join query, is small.

Note that the reported running time is the aggregated time spent on local computations at all parties and does not include the time spent on message transmissions, since the latter depends on transmission media, network protocol, traffic etc. To measure the communication cost in an implementation-independent manner, we collected the total size of messages transmitted among all parties, as shown in Figure 7. We can see that the message size increases with increasing $|D|$. As analyzed in Section 6.1, because we used only one join attribute in our experiments, the number of join value combinations $|C_i|$ in a table is the same as $|D|$. Thus, the communication cost depends only on $|D|$. If there are more join attributes in table T_i, $|C_i|$ may be much larger than $|D|$, but no more than $|T_i|$. Note that the message size is independent of the size of tables or the join size.

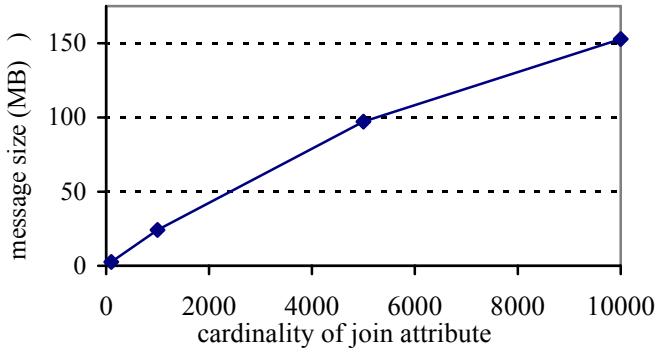

Figure 7. Message size w.r.t. |D| (B=5)

The experiments show that the running time depends mainly on the join cardinality and only the first phase involves scanning the tables. The tables are scanned only once with hash functions applied to each distinct join value, thus this first phase is very efficient. The subsequent phases depend only on the join cardinality and parameters α and β. Thus, the proposed approach is highly suitable for the trend and pattern analysis on large and private databases while scalability and privacy are major concerns. In all our experiments, the total running time is less than 20 seconds for all computations and less than 5 seconds for network communications.

Discussions

The capabilities of computing join aggregate queries over private tables are essential for collaborative data analysis that involves multiple private sources. In this paper, we proposed an efficient privacy-preserving protocol for computing join aggregate queries over private tables. By a novel use of the sketching technique and the private scalar product protocol, we achieve a level of protection not provided by the previous encryption technique. The key idea is locally maintaining random shares of atomic sketches that provide no clue on the data owned by other parties. This protocol can be naturally extended for handling group-by aggregations and OLAP roll-up/drill-down operations, while maintaining the same level of privacy protection.

References

Adam, N. R., & Wortman, J. C. (1989). Security-control methods for statistical databases. *ACM Computing Surveys*, 21 (4), 515-556.

Agrawal, R., Evfimievski, A., & Srikant, R. (2003). Information sharing across private databases. ACM SIGMOD International Conference on Management of Data.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. ACM SIGMOD International Conference on Management of Data.

Agrawal, R., Srikant, R., & Thomas, D. (2005). Privacy preserving OLAP. ACM SIGMOD International Conference on Management of Data.

Alon, N., Gibbons, P. B., Matias, Y., & Szegedy, M. (1999). Tracking join and self-join sizes in limited storage. ACM PODS Principles of Database Systems.

Alon, N., Matias, Y., & Szegedy, M. (1996). The space complexity of approximating the frequency moments. ACM STOC Symposium on Theory of Computing.

Clifton, C., Doan, A., Elmagarmid, A., Kantarcioglu, M., Schadow, G., Suciu, D., & Vaidya, J. (2004). Privacy-preserving data integration and sharing. ACM SIGMOD Workshop on Research issues in Data Mining and Knowledge Discovery.

Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., & Zhu, M. Y. (2002). Tools for privacy preserving distributed data mining. *SIGKDD Explorations*, 4 (2), 28-34.

Dobra, A., Garofalakis, M., Gehrke, J., & Rastogi, R. (2002). Processing complex aggregate queries over data streams. ACM SIGMOD International Conference on Management of Data.

Du, W., & Atallah, M. J. (2001). Privacy-preserving cooperative statistical analysis. Computer Security Applications Conference.

Du, W., & Zhan, Z. (2002). Building decision tree classifier on private data. ICDM Workshop on Privacy, Security, and Data Mining.

Emekci, F., Agrawal, D., Abbadi, A. E., & Gulbeden, A. (2006). Privacy preserving query processing using third parties. ICDE International Conference on Data Engineering.

Ganguly, S., Garofalskis, M., & Rastogi, R. (2004). Processing data-stream join aggregates using skimmed sketches. EDBT International Conference on Extending Database Technology.

Goethals, B., Laur, S., Lipmaa, H., & Mielikainen, T. (2004). On private scalar product computation for privacy-preserving data mining. ICISC International Conference in Information Security and Cryptology.

Goldreich, O. (2001). Secure multi-party computation. *The Foundations of Cryptography* (Vol. 2, Chapter 7). Cambridge University Press.

Gray, J., Bosworth, A., Layman, A., & Pirahesh, H. (1996). Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. ICDE International Conference on Data Engineering.

Jefferies, N., Mitchell, C., & Walker, M. (1995). A proposed architecture for trusted third party services. Cryptography Policy and Algorithms Conference.

Kantarcioglu, M., & Vaidya, J. (2002). An architecture for privacy-preserving mining of client information. ICDM IEEE Conference on Data Mining Workshop on Privacy, Security and Data Mining.

National Institute of Standards and Technology (NIST). (2002). Secure hash standard. Federal Information Processing Standards 180-2.

Stinson, D. R. (2006). *Cryptography: theory and practice* (3rd ed.). Chapter 4. Chapman & Hall/CRC.

The QuickHash Library. (2006). Retrieved from the World Wide Web: http://www.slavasoft.com/quickhash/

Vaidya, J. S., & Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Wang, L., Jajodia, S., & Wijesekera, D. (2004). Securing OLAP data cubes against privacy breaches. IEEE Symposium on Security and Privacy.

Yao, A. C. (1986). How to generate and exchange secrets. Symposium on Foundations of Computer Science.

Zhang, N., Zhao, W., & Chen, J. (2004). Cardinality-based inference control in OLAP systems: an information theoretical Approach. ACM DOLAP International Workshop on Date Warehousing and OLAP.